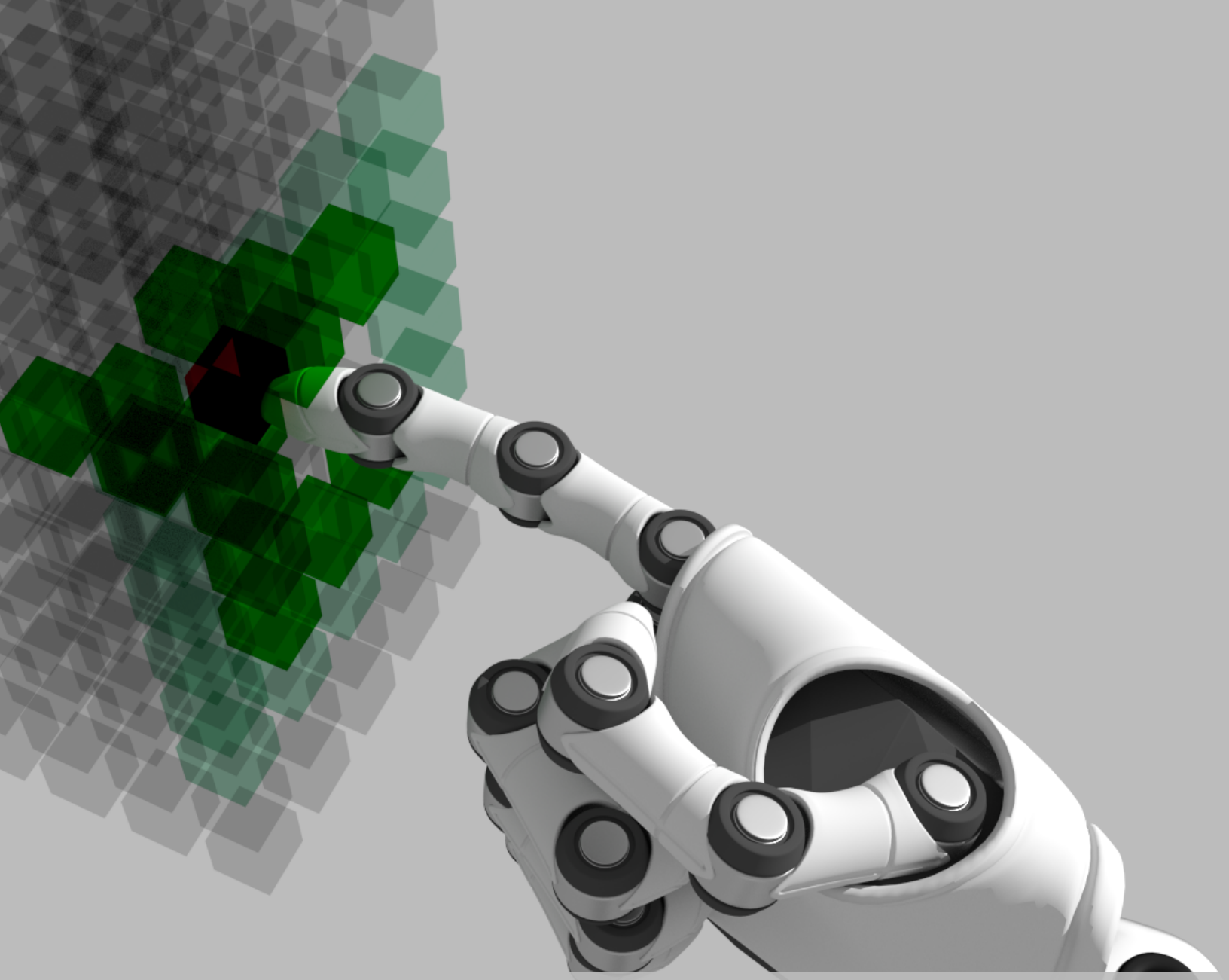


Contact driven robotic grasping

Javier Felip León

UNIVERSITAT JAUME·I

Director: Antonio Morales
Universitat Jaume I
April 2016

UNIVERSITAT JAUME·I
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Contact driven robotic manipulation

JAVIER FELIP LEÓN

Director: Antonio Morales Escrig

Doctoral Thesis in Computer Science

Castellón de la Plana, Spain April 2016

Director	Dr. Antonio Morales Escrig Robotic Intelligence Lab Department of Computer Science and Engineering Universitat Jaume I, Spain
Reviewers	Prof. Dr. Helge Ritter Neuroinformatics Group Technische Fakultät Universität Bielefeld, Germany
	Ao. Univ. Prof. Dipl.-Ing. Dr.techn. Markus Vincze Automation and Control Institute Technische Universität Wien, Austria
Opponents	Dr. Mohamed Abderrahim Robotics Lab Universidad Carlos III, Spain
	Prof. Dr. Helge Ritter Neuroinformatics Group Technische Fakultät Universität Bielefeld, Germany
	Ao. Univ. Prof. Dipl.-Ing. Dr.techn. Markus Vincze Automation and Control Institute Technische Universität Wien, Austria

Universitat Jaume I
Computer Science and Engineering Department
Av. de Vicent Sos Baynat s/n
12071 Castellón de la Plana
SPAIN

The work presented in this thesis has been mainly carried out at the Robotic Intelligence Laboratory at Universitat Jaume I of Castelló. The support for this research has been provided in part by the European Commission's Seventh Framework Programme through the GRASP project, IST-FP7-IP-215821, and by Universitat Jaume I FPI programme through the PREDOC/2010/23 grant. Part of the work presented in this thesis has been carried out in the Institute for Anthropomatics at KIT (Karlsruhe Instiute for Technology), Karlsruhe, Germany under the supervision of Prof. Tamim Asfour.

Acknowledgements

I have spent a long time working on this thesis. There are a lot of people that went in and out during this years and I am grateful that I met all of them. First I would like to thank Antonio Morales, without him it would have been impossible to achieve this goal. He gave me the opportunity to be part of the GRASP project and start my research career. His advice has been a guiding light for every step of this work. It is thanks to him that I learned to think a bit more as a scientist and less as an engineer.

Thanks to all of the GRASP project members, specially to those who I collaborated more with and spent long working nights before deadlines and review meetings: Aitor, Walter, David, Jonna, Ville and Jeannette. The participation in the GRASP project has given me experience, knowledge and expertise, but more importantly it has made me meet some awesome researchers and team workers.

My stay at KIT was a very important experience and a plus to my formation as a researcher. Thanks to Tamim for hosting me at his lab and guiding my research there, to Niko for his technical support, to Paul for fixing ARMAR and to the rest of the team for their warm welcome into the group. I am specially thankful to Stefan and David Gonzalez who really made me feel home and enjoy my time in Germany.

I would also like to thank all the members of the Robotic Intelligence Lab and the Interactive Robotic Systems Lab from Universitat Jaume I. Special thanks to the group of people that everyday join for a snack before lunch. Thanks to Beatriz, Higinio and Jose for their help in the set up of the simulated environment and the work in the GRASP project, to Mario for his brief and effective robotics lessons and for helping me paint Tombatossals, to Eris for his advice, to Angel for developing the software architecture, to Marco for his permanent disagree and seek for perfection and to Juan Carlos, with whom I started this adventure, for his friendship and readiness to help.

Thanks to my family, specially to my mother and my brother for being always a pillar in my life, and for not allowing me to cook. Thanks to Alba and her family. Since I met her everything turned out to be brighter and her daily support has become essential. Finally, thanks to my father, always in my mind. It was his light that made me see when it was dark and his thrust that pushed me forward when I was exhausted.

Abstract

Autonomous manipulator robots could be a key factor for the future of mankind. Robots can help us with mundane tasks in household scenarios, and performing unpleasant jobs. But more importantly, they can also be helpful to operate in hazardous environments or in environments where the humans are still not adapted to live in, such as underwater or in the space. However, more than 40 years after the first six-axis manipulators were built, the problem of autonomous manipulation in unstructured scenarios remains an open scientific question.

Human manipulation skills are flexible, robust and adapt to unknown environments. We believe that the understanding of human grasping may lead to new paradigms and techniques that will advance the current state-of-the-art of robot skills. Our approach to autonomous manipulation in unstructured scenarios, is based on neuroscience studies carried out on humans aimed at the sensorimotor control of manipulation. Thanks to those studies, we have identified the principal components of human manipulation and implemented them on a robotic platform.

Human manipulation is mainly driven by the detection and prediction of contacts. A task (e.g. pick and place) is composed of several simple atomic actions (i.e. grasp, lift, place, release) that are bound by mechanical contact events. Therefore, one of the keys of our approach is contact event detection and prediction. For detection we provide a novel sensor fusion framework that is targeted at contact detection and can integrate any information available. For prediction, a simulation approach is used. We explore the available robotic simulators and implement a virtual surrogate for our robotic platform that will act as a prediction engine to foretell where and when contacts will arise. The other key component of our system, is the implementation of the simple atomic action vocabulary, we have implemented all the atomic actions using a sensor-based approach. Therefore, the atomic actions are endowed with reflexes and corrective movements that, as we experimentally demonstrate, improve the robustness and reliability of the system.

As a result, in this thesis we provide a complete implementation of a manipulation system that can operate in unstructured environments. However, there is room for improvement in all the components presented and many unsolved questions that should be addressed in the future remain.

Resumen

Los robots manipuladores autónomos pueden ser un factor clave para el futuro de la humanidad. Pueden ayudarnos con las tareas cotidianas en escenarios domésticos y con la realización de trabajos desagradables. Lo más importante, es que también pueden utilizarse para operar en ambientes peligrosos o en entornos en que los seres humanos todavía no estamos adaptados para vivir, como por ejemplo, bajo el agua o en el espacio exterior. Sin embargo, tras más de 40 años desde la construcción de los primeros robots manipuladores de seis grados de libertad, el problema de la manipulación autónoma en escenarios no estructurados sigue siendo uno de los principales retos de la robótica.

La habilidad de manipulación de los seres humanos, es flexible, robusta y se adapta fácilmente a entornos desconocidos. Creemos, que la comprensión del agarre humano, puede conducir a nuevos paradigmas y técnicas que hagan avanzar el estado del arte actual de las habilidades manipuladoras robóticas. Nuestro enfoque, se basa en estudios de neurociencia llevados a cabo en seres humanos, para determinar como funciona el control sensorimotor de la manipulación humana. Gracias a estos estudios, se han identificado los principales componentes de la manipulación humana que hemos implementado en nuestra plataforma robótica.

La manipulación humana se basa principalmente en la detección y predicción de contactos. Una tarea (e.g. agarrar y transportar un objeto) se compone de varias acciones atómicas simples (i.e. agarrar, transportar, depositar, soltar) que están conectadas por eventos de contacto. Por lo tanto, una de las claves de nuestro enfoque es la detección y predicción de eventos de contacto. Para la detección, se proporciona un novedoso entorno de fusión sensorial que está orientado a la detección de contactos y que puede integrar cualquier información disponible. Para la predicción, se utiliza un entorno de simulación. Hemos explorado los diferentes simuladores robóticos disponibles e implementado un sustituto virtual para nuestra plataforma robótica. El simulador actuará como un motor de predicción para indicar dónde y cuándo se producirán contactos. El otro componente clave de nuestro sistema, es el vocabulario de acciones atómicas. Todas las acciones atómicas que hemos implementado, utilizan un control basado en sensores. Por lo tanto, están dotadas de reflejos y movimientos correctivos que, como se demuestra experimentalmente, mejoran la robustez y la fiabilidad del sistema.

Como resultado del trabajo presentado en esta tesis, se proporciona una implementación completa de un sistema de manipulación que puede funcionar en entornos no estructurados. Sin embargo, hay margen de mejora en todos los componentes y quedan muchas preguntas sin resolver que deberán ser abordadas en el futuro.

Objetivos

Independientemente de las limitaciones de hardware, hay muchos desafíos de software que necesitan ser abordados antes de que un robot humanoide de tamaño adulto pueda estar disponible comercialmente. Entre los problemas no resueltos, está la manipulación autónoma en escenarios no estructurados. Este problema es el objeto de estudio de esta tesis.

El objetivo principal de esta tesis consiste en dotar a un robot con habilidades de manipulación suficientes para llevar a cabo tareas de manipulación comunes en escenarios no estructurados. El robot tiene que ser capaz de adaptarse a los imprevistos propios de los entornos no estructurados y tratar con objetos desconocidos. Por otra parte, también se proporciona un mecanismo para definir tareas que el robot pueda ejecutar. Por último, las habilidades implementadas deben ser transferibles entre diferentes plataformas con un esfuerzo razonable. Por ello, el trabajo presentado en esta tesis se puede implementar en cualquier robot capaz de realizar tareas de manipulación. La manipulación diestra y otros tipos de manipulación más complejos (e.g. cambios de agarre sin soltar previamente el objeto), están fuera del alcance del trabajo presentado. Sin embargo, dichas habilidades pueden ser fácilmente añadidas como nuevos componentes del sistema gracias a la arquitectura modular.

Metodología

Para la implementación y validación del sistema de manipulación basado en la detección de contactos y la interacción física, en primer lugar nos hemos inspirado en estudios neurocientíficos sobre el control sensorimotor de la manipulación humana. A partir de estos estudios, se han identificado los componentes esenciales de la manipulación:

- Primitivas de manipulación: las tareas de manipulación de objetos se componen generalmente por una serie de fases. Cada una de las fases se encarga de lograr una meta específica, es decir, un objetivo parcial de la tarea. Denominamos primitiva de manipulación a cada una de estas fases.
- Eventos de contacto: codifican la formación y ruptura de contactos entre cualquiera de los dedos y el objeto, o el objeto agarrado y otros objetos o superficies.
- Fusión sensorial: los eventos de contacto pueden ser detectados mediante diferentes sentidos, la visión, la fuerza, el tacto o incluso el sonido.

- Predicción de eventos de contacto: durante la manipulación de objetos, el cerebro predice los eventos sensoriales que indican la consecución de objetivos parciales.
- Acciones correctivas: la diferencia entre un evento de contacto percibido y un evento de contacto previsto, desencadena una acción correctiva que ha sido aprendida junto a la primitiva de manipulación que se esta ejecutando.

En segundo lugar, hemos puesto en práctica cada uno de los componentes identificados y los hemos unificado dentro del mismo marco. Cada uno de los componentes, requiere su propio esfuerzo de investigación, implementación y validación experimental.

Como resultado, hemos puesto en marcha un sistema capaz de realizar tareas de manipulación en escenarios no estructurados. Es también capaz de adaptarse a la incertidumbre y a eventos inesperados. Las habilidades de manipulación se demuestran a lo largo de la tesis en los diferentes experimentos realizados para validar cada uno de los componentes del sistema. En estos experimentos, el robot ha sido capaz, por ejemplo, de vaciar una caja llena de objetos desconocidos sin necesidad de utilizar la visón, sólo se ha utilizado la posición y el tamaño de la caja. Otro experimento ha demostrado la capacidad del sistema para agarrar una botella con una mano y desenroscar el tapón con la otra mano.

Más allá de los robots que se utilizan en esta tesis, el sistema y las técnicas que se presentan, pueden ser implementadas en cualquier robot manipulador a través del mecanismo de abstracción presentado. Además de en las plataformas utilizadas para la investigación y el desarrollo del sistema de manipulación, el sistema, ha sido implementado en el robot Baxter y utilizado para participar y resolver el desafío del Amazon Picking Challenge. En este desafío, el robot debe agarrar de forma autónoma un conjunto de objetos de una estantería y colocarlos en un recipiente.

Contribuciones

Las principales contribuciones de la investigación llevada a cabo en esta tesis se enumeran a continuación:

- La contribución principal consiste en el desarrollo del paradigma de primitivas de manipulación. Una primitiva de manipulación es un controlador reactivo que esta enfocado a conseguir un objetivo concreto. Este concepto permite implementar y especificar un conjunto de acciones atómicas (e.g. transportar, mover, agarrar, deslizar, empujar) que pueden ser utilizadas como bloques para definir tareas más complejas.
- Diseño e implementación de una estrategia reactiva para el agarre de objetos. El valor de las estrategias de control basadas en realimentación sensorial se demuestra experimentalmente.

- Como caso de uso del paradigma de primitivas de manipulación, se presenta la implementación de las primitivas necesarias para permitir al robot realizar la tarea de desenroscar el tapón de una botella. Para ello se ha diseñado una primitiva de desenroscado que, tras su implementación, se ha validado experimentalmente.
- Se proporciona un nuevo método para detectar contactos utilizando el modelo de robot, la información de movimiento del brazo actual e imágenes RGBD.
- Un entorno para la fusión sensorial centrado en la detección y localización de contactos es otra de las aportaciones de esta tesis. También se prevé la integración de varias señales sensoriales, el contexto y la predicción.
- Estudio e implementación de un simulador dinámico para predecir la interacción robot-objeto y los contactos que se derivan de ella.
- Una arquitectura para la abstracción del hardware utilizando el paradigma primitivas de manipulación con el fin de transferir planes entre diferentes robots.
- La arquitectura de control organizada en cuatro capas que coordina todos los componentes del sistema.

Conclusiones y trabajo futuro

La manipulación robótica autónoma en entornos no estructurados sigue siendo uno de los grandes retos de la robótica. Con el fin de manipular un objeto, el robot tiene que realizar varios pasos, cada uno de ellos una área de investigación en si misma. En primer lugar, el robot debe que buscar el objeto en el entorno y determinar su posición. En segundo lugar, tiene que planificar la manera de manipular el objeto teniendo en cuenta las limitaciones de la tarea y el entorno. Por último, ha de ejecutar la acción prevista adaptándose a los errores de predicción y las posibles perturbaciones externas. En esta tesis, se ha hecho uso de los algoritmos más avanzados disponibles en la literatura para los dos primeros pasos y desarrollado estrategias basadas en realimentación sensorial para el último paso.

Hoy en día, hay una gran cantidad de robots que han demostrado buenas habilidades de manipulación. Sin embargo, se ocupan generalmente de pequeños subconjuntos del problema, introduciendo restricciones y realizando asunciones sin resolverlo completamente. Los robots están todavía lejos de alcanzar a los seres humanos en cuanto a destreza y fiabilidad en dichos escenarios. En esta tesis nos hemos inspirado en los experimentos de agarre humanos para implementar un sistema capaz de realizar tareas de manipulación en entornos no estructurados. El sistema de manipulación implementado esta basado en la retroalimentación sensorial, control adaptativo, detección de contactos, predicción de contactos, detección de objetos y reconocimiento de objetos. Aunque

también se realizan algunas asunciones y restricciones sobre entorno, creemos que van a ser eliminadas poco a poco en el futuro.

El desarrollo de esta tesis, ha producido varias publicaciones relacionadas y abordado algunos de los problemas más frecuentes de la manipulación robótica. Cada uno de los aspectos desarrollados, abre la puerta a futuras mejoras. Más allá de dichas mejoras, el trabajo presentado sugiere mejoras para ampliar el alcance de los resultados presentados. En los siguientes párrafos se detallan dichas posibles mejoras y extensiones.

Experimentos de agarre humano

Tras revisar los estudios de neurociencia disponibles en la literatura, hemos extraído como una de las conclusiones, que los seres humanos realizan movimientos correctivos cuando las señales sensoriales previstas y percibidas no coinciden. Sin embargo, no hay experimentos disponibles en la literatura que estudien en detalle cómo se realizan estas correcciones. En esta tesis se han realizado experimentos preliminares para observar cómo los seres humanos realizan correcciones, sin embargo, para entender mejor y determinar cómo los seres humanos se adaptan a las diferentes situaciones inesperadas, es necesario realizar más experimentos.

Primitivas de manipulación

Aunque hemos propuesto e implementado un conjunto de primitivas de manipulación, todavía hay primitivas que deben ser identificadas e implementadas con el fin de aumentar la gama de tareas que pueden ser descritas. Algunas de esas primitivas ya han sido identificados (e.g. empujar, tirar) pero puede haber otras relacionadas con entornos más específicos (e.g. para cocinar) que aún se desconocen.

El paradigma de primitivas de manipulación abre la puerta a la aplicación de técnicas de aprendizaje automático para reemplazar la implementación de primitivas de manipulación. De este modo las primitivas de manipulación utilizadas para describir acciones podrían ser aprendidas en lugar de programadas. Sin embargo, como se sugiere en [Johansson and Flanagan, 2010], los movimientos correctivos se aprenden simultáneamente con cada primitiva de manipulación. Por lo tanto, para utilizar técnicas de aprendizaje automático para adquirir nuevas habilidades, se debe integrar el aprendizaje de los movimientos correctivos también. Además, como las correcciones son activadas por diferencias entre predicción y percepción, es necesario un mecanismo de predicción que debe ser introducido en el sistema de aprendizaje. Una posible solución para aprender este tipo de controladores fue propuesto por [Pastor et al., 2011], donde se utiliza la memoria sensorimotora par aprender a agarrar un objeto. Los movimientos correctivos se llevan a cabo cuando se detectan errores de predicción. Sin embargo, no está claro cómo la estrategia aprendida puede generalizarse a otras tareas de agarre con diferentes entornos, objetos y configuración de la mano.

Percepción

Durante las tareas de manipulación, los seres humanos detectan eventos de contacto utilizando la información procedente de diferentes señales sensoriales [Johansson and Flanagan, 2010]. En esta tesis hemos propuesto, implementado y demostrado una nueva técnica de fusión sensorial que integra información de diferentes fuentes y proporciona estimaciones de los eventos de contacto.

Tras la revisión en detalle de la literatura sobre estudios neurocientíficos de la manipulación humana, no se han encontrado experimentos que proporcionen pistas acerca de cómo los humanos resuelven el problema de la fusión sensorial. Si hay o no precedencia de una modalidad sensorial (por ejemplo táctil) sobre otras (por ejemplo, visual o predicciones) es aún desconocido. Intuitivamente, nuestra implementación del sistema de fusión de sensorial, utiliza un enfoque probabilístico y las prioridades de cada modalidad sensorial dependen del valor de confianza asignado en su generación. Nuevos experimentos centrados en este aspecto de la manipulación humana serían muy valiosos para mejorar los métodos de fusión sensorial y la detección de contactos.

Predicción

En esta tesis se utiliza la simulación dinámica para predecir dónde y cuándo se producirán contactos. Sin embargo, la simulación dinámica requiere mucho tiempo de cómputo y no es posible obtenerla en tiempo real. En el corazón de la simulación dinámica, se encuentra el motor físico. El continuo desarrollo de los motores físicos hace que sea difícil seleccionar el mejor para nuestro simulador. En esta tesis hemos seleccionado Open Dynamics Engine (ODE) pero en el futuro puede haber otros motores con mejor rendimiento. Por lo tanto, la implementación de una capa de abstracción del motor físico sería muy valiosa para el nuestro sistema de predicción. Sin embargo, aunque se seleccione el mejor motor físico y que éste funcione en tiempo real, para obtener simulaciones precisas, se requiere ajustar una gran cantidad de parámetros relacionados con el material de los objetos y sus propiedades inerciales. Determinar de forma precisa dichos parámetros, es hoy en día uno de los problemas más importantes para la utilización de simuladores como sustitución de la realidad.

El problema de la predicción también se puede abordar desde otros puntos de vista, en lugar de utilizar un simulador dinámico, las consecuencias de la interacción entre los objetos pueden ser aprendidas por el robot [Belter et al., 2014]. Otro enfoque posible, es la utilización de la memoria sensoriomotora de una ejecución exitosa para conducir la ejecución [Pastor et al., 2011]. Por lo tanto, un enfoque que combine la memoria sensoriomotora, el aprendizaje y la simulación dinámica podría ser la solución híbrida para el problema de predicción. En primer lugar, los parámetros de simulación física se pueden ajustar a través de interacciones exploratorias con el objeto. En segundo lugar, la simulación se puede utilizar para proporcionar datos de entrenamiento a los

algoritmos de aprendizaje. Por último, la memoria sensoriomotora aprendida se puede utilizar para conducir la ejecución de tareas y activar movimientos correctivos cuando se detecten errores de predicción.

Planificación de tareas a alto nivel

Las definiciones de tareas utilizadas para los experimentos presentados en esta tesis, fueron creadas manualmente. El conjunto de primitivas de manipulación proporcionado, puede ser utilizado junto con primitivas de percepción como los símbolos básicos para un planificador de tareas de alto nivel que puede convertir órdenes semánticamente significativas tales como "limpiar la mesa" o "fregar el suelo" en definiciones de tareas ejecutables por el robot. Como recientemente [Yang et al., 2015] ha publicado, es posible generar representaciones de tareas de forma automática a partir de secuencias de vídeo no etiquetadas. Sin embargo, a partir de un solo ejemplo, es difícil obtener los parámetros para ajustar la tarea a un escenario específico. Una vez que las descripciones de las tareas puedan ser aprendidas de forma automática a partir de ejemplos, un método de adaptación para ajustar la descripción de la tarea aprendida a diferentes escenarios será necesario. Con suficientes ejemplos de la misma tarea, el aprendizaje por imitación proporciona métodos que pueden utilizarse para este fin.

Contents

Chapter 1

Introduction

Compared to the most advanced robots, average human manipulation skills are outstanding. Robots that perform manipulation tasks in daily life, are mainly devoted to execute repetitive, preprogrammed tasks. Their lack of adaptability has relegate them to be deployed in factories in a controlled environment, inside a cage. There are a lot of simple tasks in manufacturing that cannot be automated with current robots. Those tasks are performed by humans, that with much greater cognitive abilities are stuck counting cups, packing and unpacking boxes and performing tedious, repetitive and not knowledge based work.

Developing a robot that has close-to-human manipulation abilities would change the way robots are used in the current society. The potential of such machines could transform the life of most of the developed countries inhabitants. Factories could include such robots into their manufacturing lines, move the workers performing repetitive tasks to a higher level duty, and increase their productivity. In addition, robots could start being deployed in real houses to assist elderly or disabled people, boosting their living standards and giving them back some independence. Maybe in the future, service robots will be present in each house and considered another house appliance like the washing machine or the dishwasher.

Unfortunately the current state of the art in robotic manipulation is far away from that purpose. So far, there has been no real interest by the industry to include robots with human-like manipulation skills into their production lines. Nevertheless, with the recent growth of collaborative robotics the interest of the industry may change in the near future.

In this thesis we study and develop a framework that enables robots to perform manipulation tasks in unstructured and changing environments. To do so, we have taken inspiration from neuroscience studies about the human sensorimotor control of manipulation and the visual stimuli processing. In the implemented reactive contact based manipulation system: sensory feedback, adaptive control, contact detection, contact prediction, object detection and object recognition are key.

1.1 Motivation

Social facts

Advances in medicine and lifestyle are increasing the life expectancy of the world's population. Moreover, birth control is reducing the number of young people that in the future will support the elder ones. This fact is inverting the population pyramid[1]. In the near future there will not be enough working force to support a society with too many old individuals. Service robotics is an answer to this problem. It can reduce the cost to take care of an elderly person and increase the life quality of all the population. A service robot that takes care of us and does the housework would definitely increase the life standards without the need for more workforce.

Moreover, robust, flexible and adaptive robots will be also a solution for unpleasant tasks or rescue missions in hazardous environments (e.g nuclear plant, factories) and space missions. However, the threat of a future without enough workforce to maintain an elderly population is not enough to trigger public and private investment into robotics development. Fortunately, the recent rise of collaborative robots has set up the perfect moment to attract the investment into robots that can work shoulder to shoulder with humans.

The rise of collaborative robots

Collaborative robotics is a branch of industrial robotics that unlike classic industrial manipulators, uses compliant robots that work shoulder to shoulder with humans. This robots are more failure tolerant and robust to environment changes. Moreover, they have the ability of dealing with a determined amount of uncertainty.

A clear example of the growing interest of the industry in compliant and adaptive manipulators is the emergence of companies (e.g. Rethink Robotics Inc.[2], Universal Robots[3] A/S) that design cheap and robust robots to fill in that gap in the industry marketplace. Recently, Amazon has also shown its interest in that kind of robotic manipulation tasks. In May 2015, they organized the first Amazon Picking Challenge (APC) where a robot had to grasp some objects from a shelf and place them in a bin. From the scientific point of view, it looked like a solved task. However, it turned out to be an unsolved problem. None of the 30 teams from all around the world were able to grasp all the challenge objects.

Although more investment is being attracted by robotics, the current state of the art is far from being able to provide a robot companion with human-like abilities. One

[1]Source: United Nations, Department of Economic and Social Affairs, Population Division. World Population Prospects: The 2012 Revision.

[2]Rethink Robotics Inc. `http://www.rethinkrobotics.com/`

[3]Universal Robots A/S `http://www.universal-robots.com`

Figure 1.1: Some state of the art full-size humanoid robots. Upper row: ASIMO [Sakagami and Watanabe, 2002], ARMAR-4 [Asfour et al., 2013] and iCub [Metta et al., 2008]. Lower row: Kenshiro [Nakanishi et al., 2012], REEM-C [Tellez et al., 2008] and HRP-4 [Kaneko et al., 2011]

approach to understand and be able to replicate human-like abilities is to perform neuroscience studies and observe how humans solve the challenges.

Taking inspiration from humans

Nature has always been a source of inspiration for the human being. Going back to the renaissance, Leonardo Da Vinci took inspiration from nature for several of his inventions and tried to mimic the bird's anatomy at human scale to build a flying suit.

Biomimicry only replicates the geometry of the observed plants or animals to take advantage of a nature-designed structure. Often it is not enough to replicate the design if the control and behaviour are not replicated too. A good example is Kenshiro (See Fig. 1.1 lower left), a robot from the University of Tokyo [Nakanishi et al., 2012], which

mimics mostly all the tendons, bones and muscles of a human but cannot walk, grasp, manipulate or interact with its environment.

The main applications of service and rescue robotics are bounded to human engineered environments (factories, houses, cities, vehicles). As the humanoid robots aim to help humans in their own environment, the human form factor is appropriate for such scenarios. As shown in the DARPA Robotics Challenge (DRC)[4] trials, not always the exact human embodiment is the best for specific scenarios. Even though, the winner team presented a robot with two arms and two legs.

Having robots with the same kinematics as humans, makes learning by demonstration and demonstrating tasks to the robots easier. Moreover, in the future it could also work the other way round, allowing robots to teach humans in a more intuitive way. In addition, the human embodiment allows robots to perform easier non-verbal communication and have better acceptance by humans.

There are already many examples of full size humanoid platforms (see Fig. 1.1), unfortunately none of them has a control software stable enough to let such high Degrees of Freedom (DOFs) robots work standalone alongside humans in real environments. Understanding human sensorimotor control could be a key factor to develop software able to control robots with many DOFs like Kenshiro, REEM-C, ASIMO, HRP-4, ARMAR-4 or iCub (See Fig. 1.1) and transform them into multi-purpose, robust service robots.

Recent advances in measuring technologies, such as eye trackers and fMRI, enable the neuroscientists to study the response of the brain under controlled conditions. Although a lot of useful information has emerged from this field, the brain mechanisms are mostly unknown.

1.2 Aims and scope

Regardless of the hardware constraints, there are many software challenges that need to be tackled before a full size commercial humanoid robot can be available. In this thesis we focus on autonomous manipulation under uncertainty in unstructured scenarios.

The main aim of this thesis is to research how to endow a manipulator robot with sufficient manipulation skills to perform the most common manipulation tasks. The robot has to adapt to unstructured environments and deal with unknown objects. Moreover, a mechanism to define tasks that the robot can execute has to be provided. Finally, the implemented skills should be transferred among different platforms with reasonable effort.

Apart of humanoid robots, the work presented in this thesis can be applied to any industrial or collaborative manipulator with potential manipulation skills. Dexterous

[4]Darpa Robotics Challenge: `http://www.theroboticschallenge.org/`

manipulation and in-hand manipulation are out of the scope of the work presented. However, those skills can be easily added as new components to the framework.

1.3 Methodology

In this thesis, we have taken inspiration from neuroscience theories, derived from experiments on humans, that give some clues on how humans perform manipulation tasks. First, we have identified the components of human manipulation from a wide variety of neuroscience experiments available in the literature. Those components are:

- Action phase controllers: object manipulation tasks typically involve a series of action phases. Each phase accomplishes a specific goal or subgoal of the task.
- Contact events: encode the making and breaking of contact between either the fingertips and the grasped object or the object in hand and another object or surface.
- Sensor fusion: contact events could be detected using many sensory cues, vision, force, touch or even audio.
- Contact event prediction: during object manipulation the brain predicts sensory events that signify goal attainment.
- Corrective actions: a mismatch between the expected sensory event and the perceived sensor signals triggers a learned corrective action.

Second, we have implemented each of the identified components and put them together into the same framework. Each of the components, required its own research effort and were implemented and validated experimentally.

As a result, we have implemented a system capable of performing manipulation tasks in unstructured scenarios and adapt to uncertainties and unexpected events. The manipulation skills are demonstrated along the thesis in the different experiments performed to validate the components of the system. In those experiments, the robot has been able to empty a box full of unknown objects without using visual feedback, only knowing the position and size of the box. Another experiment has shown the ability of the presented framework to grasp a bottle with one hand and unscrew its cap with the other hand.

Beyond the robots used in this thesis, the framework presented can be ported to any manipulator robot through the hardware abstraction mechanism presented. The presented framework has been ported to the Baxter robot and used to participate and solve the APC where the robot has to autonomously grab a set of target objects from a shelf and place them into a container.

1.4 Outline

This thesis is structured as follows:

Chapter 2 - Human-inspired sensorimotor control of manipulation

Chapter 2 summarizes the experiments carried out by R.S. Johansson and J.R. Flanagan, regarding human manipulation of objects, and explains their theories about the sensorimotor control of manipulation in humans.

Inspired by neuroscience, the theoretical models from Johansson and Flanagan are converted into a computational model to define a task as a set of actions connected by contact events. Finally, the elements required for a robotic implementation are identified and bound to the neuroscience model.

Chapter 3 - Manipulation primitives

Manipulation primitives are the basic actions that can be combined in order to perform a task. Chapter 3 presents the manipulation primitive paradigm and shows an example of implementation of a set of primitives. Primitives compose the vocabulary of actions used to describe tasks. Examples of complex task definitions using the paradigm are shown.

While executing a task, there are many situations that can cause the robot to fail. In some of those cases it is possible to detect the failure before it happens, thus instead of failing, a corrective action can be taken that can solve the problem and allow the task to be completed successfully. Chapter 3 describes how reflexes are embedded into manipulation primitives to make them more robust and adaptive.

Chapter 4 - Contact perception

Detecting and localizing contacts with objects and the environment is key for the robot to perform a task. Chapter 4 details the use of vision, tactile, force, proprioception, control and prediction to detect and localize contacts. The mechanisms to generate contact events are also shown. Moreover, this chapter proposes a sensor fusion method to combine different sensory cues and convert sensor readings into contact hypotheses providing a common representation for all the sensors.

Chapter 5 - Contact prediction

In Chapter 5 the mechanisms used for contact prediction are described. Using the perceived state of the environment, a dynamics simulation is executed in parallel to the real action. The simulation provides the on-line prediction of contact events, that will be used with the task description to monitor the task status, detect errors and trigger reactive behaviours. Contacts could also be predicted from previous successful executions of the task, this is also discussed in Chapter 5.

Chapter 6 - Integration architecture

All the components of the system must be organized and orchestrated. Chapter 6 presents the layered architecture used to integrate all the components. Beyond task coordination, there is a system-wide architecture that dictates how modules have to be implemented depending on their role and level of abstraction.

Chapter 7 Embodiment abstraction

In Chapter 7, we show how the manipulation primitives paradigm can be a tool for embodiment abstraction enabling the same task definition to be successfully used on different robots.

Chapter 8 - Object perception

Object detection, localization and recognition mechanisms are not studied by the main neuroscience theories that inspired this thesis. Nevertheless, it is an important part of any robotic system. In Chapter 8 we have taken inspiration from neuroscience experiments that study those abilities and implemented a visual recognition system inspired by primate brain mechanisms.

Chapter 2

Human sensorimotor control of manipulation

This chapter presents the neuroscience studies that support and inspire the contact-based robot manipulation framework presented through this thesis. Firstly, a review of the existing human grasping neuroscience experiments is shown. Secondly, the methodology, results and conclusions of some selected studies are detailed. Finally, using the conclusions extracted from the presented studies, the required building blocks for a human inspired manipulation system are outlined and mapped to the components of the contact based robotic framework.

2.1 Motivation

As discussed in the introduction, nature can be a helpful source of inspiration to provide solutions for current engineering problems. Regarding robot object manipulation, a possible solution could be to mimic how humans or great primates address those tasks. Unfortunately, so far there is not enough evidence about how the brain works and how the manipulation is performed at a sensorimotor level.

However, there is a wide variety of neuroscience studies carried out on humans, that can provide some ideas on how human manipulation works. In this thesis we have used the ideas provided by those theories, to structure and implement a system capable of manipulate robustly known and unknown objects, in unstructured environments and adapt to unexpected events.

Apart from the study of human manipulation, there is another important component required: the visual perception that allows humans to detect, localize and recognize objects in order to obtain enough information to generate grasping plans and allow physical interaction. This chapter focuses on sensorimotor control for manipulation, object detection and recognition is discussed in Chapter 8.

2.2 Neuroscience of human grasping

There is a huge amount of research on human manipulation. The studies are usually based on a set of experiments conducted on a reduced number of subjects. Generally the experimental setup consists of an object on a table in front of the subject (see Fig. 2.1 Left), who has to grasp it, manipulate it and place it again on the table. During the different tests, object properties, environment, or perceptual conditions are often altered to observe the effect those alterations have on the task performance. It is also common to ask the subjects to perform the grasp in a specific way or add some constraints to the manipulation process (e.g. do not tilt the object, use index and thumb).

The data recorded depends mainly on the target of the study and can be gathered instrumenting the subject (fMRI, micro-neurography, data gloves), instrumenting the object (markers, force sensors, pressure sensors, distance sensors), instrumenting the environment (cameras, sensors on the table) or by a combination of them (gaze trackers, motion capture systems). Finally the data is analysed, discussed and the conclusions are provided.

The experiments can be classified in four different categories regarding their motivation and study goal: visual perception, motion and grasp planning, physical interaction and grasping. The next subsections provide examples of experiments of each type available in the literature and review papers and books where more details can be found.

2.2.1 Visual perception experiments

One subset of the neuroscience experiments available in the literature, is focused to the study of how visual input is processed in order to enable grasping and dexterous manipulation.

[Singhal et al., 2007] performed a series of manipulation experiments to determine the influence of visual feedback and memory while manipulating objects. After memorizing a list of paired words, the subjects were asked to grasp an object while having to recall a pair from the list. The experiment was repeated asking the subjects to perform a delayed grasp (i.e. look at the target object and grasp it without visual feedback). The results suggest that there is interference between the recall and grasp task supporting that the processing of stored perceptions information is used for the grasping tasks.

A review of experiments related to the neuroscience of visual-based manipulation can be found in [Chinellato and Del Pobil, 2009]. The review introduces the experiments performed on humans and presents a functional model of the brain that is suitable to be implemented on a robot. This studies are reviewed and used in Chapter 8 as the foundations of the implemented visual system.

2.2.2 Motion and grasp planning experiments

The trajectory that the arm follows when approaching an object is nor random neither naïve. The motion planning performed by humans is also studied by neuroscientists because it can be influenced by many factors: environment, object a priori knowledge, subsequent actions, additional constraints or available sensor feedback. The influence of the subsequent actions was studied by [Hesse and Deubel, 2010], they concluded that the subsequent actions have an important influence if the task is easy. On the other hand, if the task is complex the planning does not take into consideration the subsequent actions.

Despite the arm motion planning, the contact points of the finger with the object are also planned before the grasp is executed. There are also many factors that can influence the selection of contact points such as the object position, object shape, center of mass and task. [Gilster et al., 2012] performed experiments to determine the influence of shape when allowing the subjects to use all the fingers, in the introduction they provide a review of the different experiments and the elements that influence human grasp planning.

2.2.3 Grasping experiments

In order to study how humans grasp objects, [Santello et al., 1998] performed an experiment that involved subjects virtually grasping objects of different shapes, the joint angles of the hand were recorded using a data glove. Analysing the results, they noticed that most of the grasps were similar, a Principal Component Analysis (PCA) showed that almost all the variation was accounted by the first two components. This hand synergies were later evolved and implemented as a control software for robotic hands [Ciocarlie and Allen, 2009] and implemented using hardware mechanisms on a real robot hand [Catalano et al., 2012].

In another set of experiments, [Schettino et al., 2003] observed and characterized the evolution of specific hand configurations during the reach-to-grasp movement and their modulation by different amounts of visual feedback. Their results indicate the presence of early mechanisms of hand preshaping dependent on object shape, regardless of visual feedback availability, as well as late "corrective" mechanisms which are thought to be dependent on the availability of vision. For a more detailed review about neuroscience of human grasping refer to [Castiello, 2005]. A detailed analysis of the human hand and how the experimental findings are applied to robotics is detailed in the book edited by [Balasubramanian and Santos, 2014].

2.2.4 Physical interaction experiments

The experiments classified in this group intend to understand how humans interact with objects and what are the internal mechanisms used at the sensorimotor control level. In

a seminal work, [Johansson and Westling, 1984] performed a set of grasping experiments with an instrumented object and used the results to sketch the sensorimotor control of human manipulation. Those experiments were later repeated with some variations (sensors, object shape and texture) to research in the same direction and take advantage of new technologies [Johansson et al., 2001].

Human sensorimotor control of grasping has been deeply studied, for further details, parts I and II of [Nowak and Hermsdörfer, 2009] provide details about experimental methodologies, a review of the experiments performed on humans and the theories derived from the experimental results.

In this thesis we focus on physical interaction, thus the experiments and theories we have used to inspire our work are extracted from the physical interaction experiments. Nevertheless, to have a fully autonomous manipulation system, it is necessary to determine object position and properties. To develop the visual pipeline we have also taken inspiration from the neuroscience studies. The experiments and the development of the visual perception system is discussed in Chapter 8.

2.3 Human manipulation experiments

In this section, the human physical interaction experiments used to inspire the work of this thesis, are detailed and their results and conclusions are presented. In the next section, the conclusions and the resulting ideas extracted, are used to determine the building blocks that are required to build a complete autonomous manipulation system.

The experimental setup is similar among all the experiments. It consists of a table in front of the subject with the target object to be grasped on it, see Fig. 2.1. The object is instrumented with force sensors and its *position*, *grip force* and *load force* are recorded. Usually, during the experiments, the *slip force* is calculated asking the subjects to release slowly the object until it slips. The difference between the *slip force* and the *grip force* is called *safety margin*. The details of one of the devices and its components are shown in Fig. 2.2.

2.3.1 Grasping an instrumented small object

In this experiment 9 subjects were asked to grasp the instrumented object shown in Fig. 2.2, lift it about two centimetres, hold it for 10 seconds and replace it on the table [Johansson and Westling, 1984], Fig. 2.3 depicts the action sequence of one experiment. The variable weight of the object was set to 400g.

The subjects were asked to perform a specific pinch grasp on the object as depicted in Fig. 2.2. The lifting experiments were repeated from 32 to 48 times for each subject.

Three years later, the measurement apparatus was improved by adding a microneurography recording device. This technology allowed the tactile afferent signals to

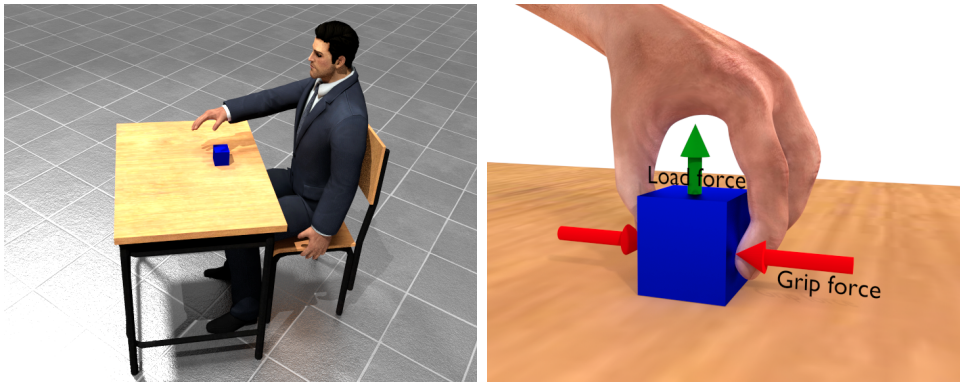


Figure 2.1: Typical experimental setup for the human precision grasping experiments: a table in front of the subject with the target object to be grasped on it. Grip and load forces are recorded together with object position. Tactile signals from the human hand are also recorded using the micro-neurography technique.

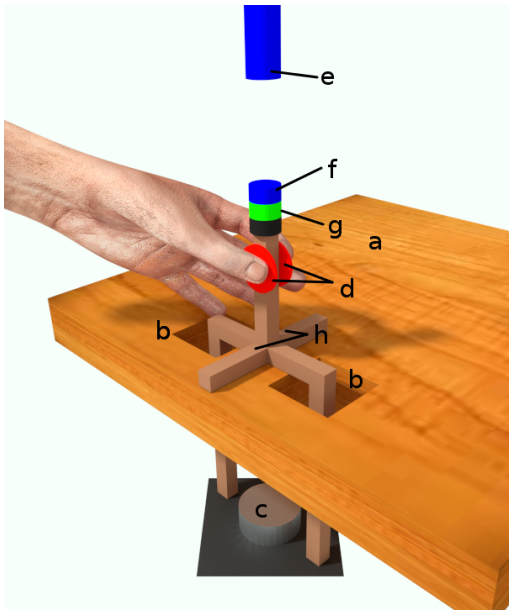


Figure 2.2: Measurement device for the grasping experiments used in [Johansson and Westling, 1984]. a) Table. b) Holes in table. c) Exchangeable weight shielded from the subject's view by the table. d) Exchangeable discs. e) Ultrasonic emitter. f) Ultrasonic receiver for vertical position measurement. g) Accelerometer. h) Strain-gauge force transducers for measurement of grip force and load force.

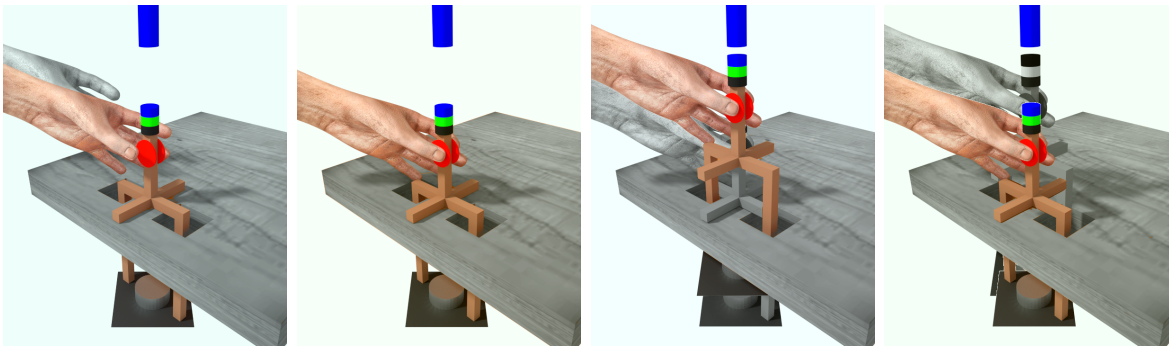


Figure 2.3: Execution steps of a grasping experiment with the instrumented object. The subject reaches and grasps the object by the red discs, lifts it about 2cm, holds for 10 seconds and replaces it on the table.

be recorded. Under the skin there are four different kinds of tactile afferents. Two of them, termed fast-adapting type I (FAI) and fast-adapting type II (FAII) respond only during dynamic phases of tissue deformation. The other two, called slowly-adapting type I (SAI) and slowly adapting type II (SAII) respond to sustained skin deformation with a graded sustained discharge [Johansson and Vallbo, 1983].

With the new technology, the experiments were repeated on 20 subjects and tactile afferent signals were recorded using the micro-neurography technique [Johansson and Westling, 1987].

Results

The averaged results of the recorded object forces and tactile afferent data are shown in Fig. 2.4. When the fingers contact the object, the grip and load forces start to increase simultaneously until the object lifts. All the subjects managed to exert grip forces that were slightly above the slip force, providing a minimal safety threshold and optimizing the time it takes to reach the desired force, reduce muscular fatigue and avoid cracking fragile objects.

The initial contact with the object is detected by the FAI and FAII afferents, but the object lift is detected only by FAII afferents. After replacing the object on the table, the object-table contact is also noticed by FAII tactile afferents and the break of contact with the object is encoded in the FAI and FAII signals.

There was a consistent delay of 0.08s between the tactile detection of the object-table contact in the replace phase, and the reduction of grip forces.

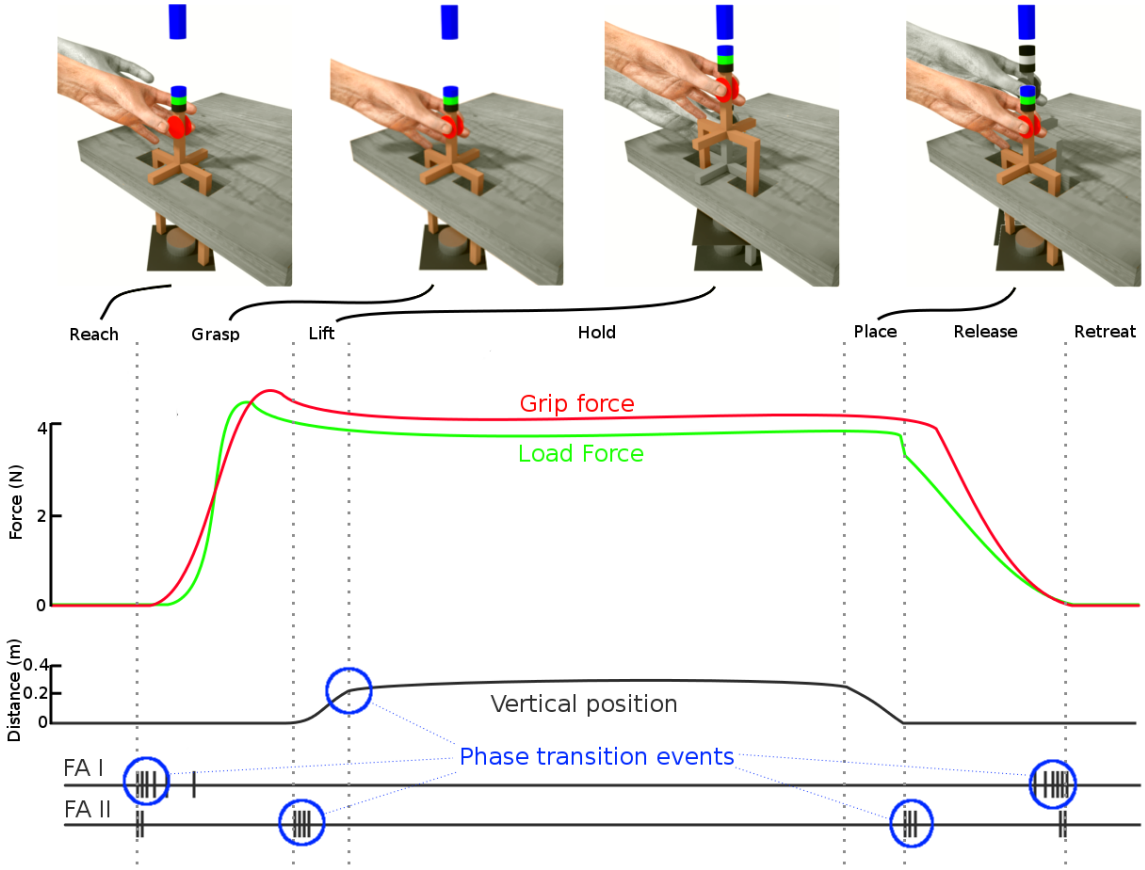


Figure 2.4: Experimental results for the instrumented box (see Fig. 2.2) grasping experiments performed by Johansson and Westling. Grip and load force directions are depicted in Fig. 2.1. Tactile signals from the human hand were also recorded using the micro-neurography technique. This diagram is a partial remake of the one in [Johansson and Flanagan, 2010] page 594, using the original data from [Johansson and Westling, 1984].

Conclusions

In 1991, using the data from these experiments, Johannson and Westling analysed the functional role of tactile signals during manipulation tasks [Johansson and Westling, 1991]. They pointed out that the transitions between phases were mainly driven by contact information and wrote: "tactile input may produce an unambiguous indication that an intended manipulative motor goal has been accomplished".

Moreover they identified several phases during the proposed manipulation task, which were separated by sensory events. Fig. 2.4 depicts the detected tactile signals, the load and lift forces, the identified task phases and the transition events.

The identified task phases were lately coined action-phase controllers and defined as focused controllers that were bound by mechanical events. The action-phase controllers, identified for the manipulation task proposed in this experiments, are listed below:

(a) Preload: The subject establishes the grip.

(b) Load: The load and the grip forces increase in parallel until the load force overcomes the gravity and the object starts to move.

(c) Transition: By wrist and/or elbow flexion the object is lifted to the intended position.

(d) Hold: A static phase where the object is held static.

(e) Replacement: The object is moved down and replaced on the surface.

(f) Delay: There is a short consistent 0.08s delay until the next phase starts.

(g) Unload: Both load and grip forces decrease in parallel until the object is released.

The reaction time to the detection of the object-table contact (delay phase) proved too fast to involve direct voluntary intervention. Hence the authors suggested that the motor commands are preprogrammed and triggered by a particular pattern of sensory information.

2.3.2 Grasping objects with unexpected friction

The first studies about the importance of frictional properties in grasp control, were performed by [Johansson and Westling, 1984]. To conduct the experiments the authors used the measurement device depicted in Fig. 2.2. To produce the change in frictional properties, the grasping pads surface was switched between silk, suede and sandpaper. The room lighting was good enough to see the target object but not to determine the material of the grasping pads. Nine right-handed healthy subjects performed a series of 32-48 trials each. The surface structure was pseudo-randomly varied between consecutive trials. The subjects were not instructed to pay attention to the grip force but to the timing and the positioning of the object in the space.

Almost a decade later, Edin et. al. performed and exhaustive set of grasping experiments targeting the frictional properties of objects [Edin et al., 1992]. Unlike the experiments by Johansson and Westling, the frictional properties of the grasping pads was heterogeneous, each pad had always different frictional properties than the other. For this experiments only sandpaper and silk were used. The frictional properties were changed randomly and recorded the lift and load forces applied independently for each finger. On a first stage 8 subjects performed 18 trials each without being able to see the object. On a second stage the experiment was repeated allowing the subjects to see the object and know in advance the type of surface that they were going to grasp on each trial. This stage analysed 29 trials on 5 subjects. A very similar measurement device to the one shown in Fig. 2.2 was used to record the experimental data.

Results

Johansson and Westling observed that the material in contact with the skin principally influenced the rate of grip force change: the more slippery the material the higher the rate. During the different tests, the subjects adapted their force to the changes in friction caused by finger sweating, indicating that they adapted to friction rather than to texture. To determine when and how the adaptation of the grip force to the surface structure took place, trials carried out subsequent to a change of the surface were analysed. The adaptation to a new surface material occurred generally around 0.1s after the object was contacted. The initial reaction to unexpected material was faster than the simple tactile reaction time: mean reaction times to tactile stimuli are over 0.15s [Lele et al., 1954]. However a comparison with the second trial on the same surface revealed that the adjustment was not complete and the first correction maintained a higher grip force, hence a greater safety margin (See Fig. 2.5).

Short-lasting slips, revealed as vibrations in the object recorded by the accelerometer, were triggering reactions between 60 and 80ms. The slips were rarely noticed by the subject and the corrections appeared to proceed in an automatic fashion without requiring the attention of the subject.The adjustment to a less slippery material is shown in Fig. 2.5 Left. The adjustment to a more slippery material is depicted in Fig. 2.5 Right.

The results of the experiments performed by Edin et. al. in 1992 confirmed those obtained by Johansson and Westling in 1984 regarding grip force adaptation to unexpected frictional properties and slip correction. Moreover Edin et. al. observed that when the frictional conditions were different for each finger, the total grip force was asymmetrically distributed among both fingers. This asymmetry enabled the safety margin to be equal for each finger. When the subjects were able to visually assert the type of material that they were about to grasp, only one out of five subjects seemed to exploit prior experience with the object with respect to the individual contact surfaces.

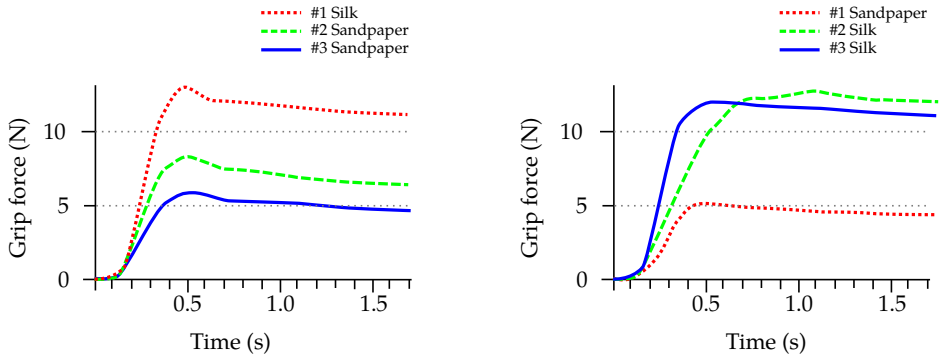


Figure 2.5: Adaptation of motor output to unexpected friction conditions. Grip forces of human grasping experiments with unexpected friction conditions. Left: Evolution of grip forces during a series of three lifts, from a slippery surface (silk) to a rough one (sandpaper). Left: Evolution of grip forces during a series of three lifts, from a rough surface (sandpaper) to a slippery one (silk). Results extracted from [Johansson and Westling, 1984].

Conclusions

The unnoticed corrective actions taken to prevent slips and adapt to different frictional properties, suggest that those reactions are directly encoded in the grasping process without requiring the subjects to be aware of them. Corrections are performed in an automatic and unattended fashion.

The adaptations of grip force are related to frictional properties detected by the tactile afferents rather than memory or other sensory cues.

The safety margin employed by all the subjects was constant among all the experiments with each subject. This suggests that it is memory-based.

The expectation from previous trials determines the initial finger forces applied when an object is lifted. Thus in the Central Nervous System (CNS) there exists a representation of a previously executed lift. This representation refers both to the object representation and to previous commands of lifting tasks [Johansson and Westling, 1991].

The task of providing a stable grasp during manipulation of objects with different shapes, weights and surface characteristics may be reduced to a problem of how to avoid accidental slips at the various digit-object contact locations. This problem seems to be solved by humans by independent digit-specific mechanisms which intermittently adjust the forces applied to an object on the basis of the frictional properties detected at each contact location.

The visual detection of the frictional properties of the object is generally not used by humans to adapt the specific finger forces.

2.3.3 Grasping objects with unexpected weight

There are several experiments that studied how prediction errors in the weight of objects affect the performance of human grasping. Some of them change the weight of the object directly [Johansson and Westling, 1984] or pull the object while it is being grasped [Cole and Abbs, 1988, Johansson et al., 1992].

During this set of experiments the subjects did not know in advance the weight of the object. Moreover, after performing several experiments they had some prior knowledge about the grasps already performed, this was exploited to change the object weight and observe the adaptation of the subjects to prediction errors. The measurement device used was the same shown in Fig. 2.2 upgraded with Force-Torque sensors between each grasping disc and the central pole. It is important to note that the weight of the object was shielded to the subject view trough the holes of the table, thus there was no visual feedback available to guess about object's weight.

Results and conclusions

The results of these experiments are quite similar to the ones detailed in Section 2.3.2. The safety margin, timings and forces applied when the weight of the object was known from previous experiments, is very similar to the results from the other experiments. The corrections required to adapt to an unexpected weight are also executed in a similar fashion, the difference is that the onset of the corrective actions is triggered by the presence or absence of an expected contact event (the break of contact between the object and the table). If the object is lighter than expected, the contact event occurs before it was predicted and the correction is triggered by that mismatch. In the opposite case, the correction is triggered by the absence of a predicted event that should have already happened. The corrective actions are executed around 100ms after the mismatch is detected, suggesting that this corrective actions are also automatic and do not require the subject attention. Figure 2.6 shows two sequences of grasp and lift trials, the left sequence with object weights 800g, 200g and 200g respectively and the left sequence 400g, 800g, and 800g respectively, the adjustment of the grip force can be observed during the second trial of each sequence.

The pushing and pulling experiments show that the tactile information drives the adaptations but also the proprioceptive information can be used to cope with external forces, pointing out the importance of sensor fusion. To confirm this results, the experiments were executed also with fingertip anaesthesia, those experiments are described in Section 2.3.5.

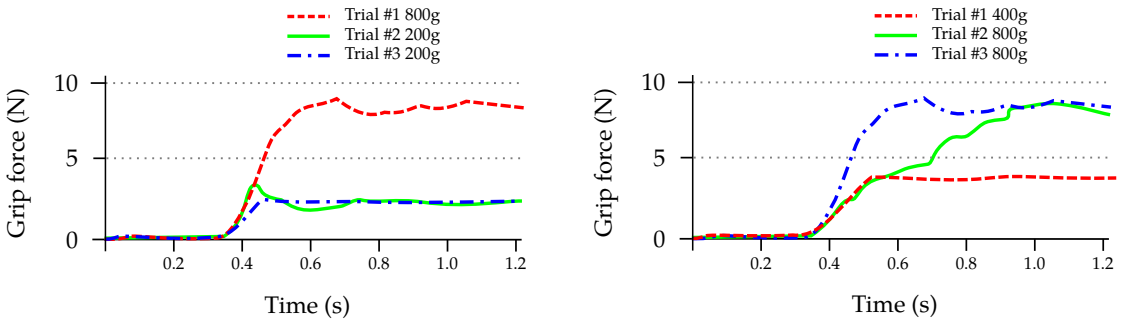


Figure 2.6: Adaptation of motor output to object weight. Grip forces for the unexpected weight experiments. Left: grip forces for a sequence of three trials with object weights 800g, 200g and 200g respectively. Right: grip forces for a sequence of three trials with object weights 400g, 800g, and 800g respectively. The initial delay in grip forces is due to the reaching phase of the experiment when the hand is moving towards the object. Results extracted from [Johansson and Westling, 1984].

From this results, the authors of the studies conclude that the corrective actions are performed in an automatic fashion by the subjects and that are triggered by mismatches of predicted contact events and actual perceived contact events.

2.3.4 Grasping objects with different shapes

To determine the importance of visual cues versus other physical interaction based sensory information, [Jenmalm and Johansson, 1997] performed a series of human grasping experiments with tapered objects, changing the angle of the graspable faces. The experiment consisted on a set of two different trial series with and without visual input. During each series, the angle of the graspable surfaces of the object was randomly changed in steps of 10º from -40º to 40º (see Fig. 2.7 Left). [Goodwin et al., 1998] performed a similar experiment but using concave and convex objects. In this case the sight of the subjects was not blocked, the concave and convex type of objects are shown in Fig. 2.7 Right. A similar device as the depicted in Fig. 2.2 to measure the grip force, load force and object position was used.

Results and conclusions

Despite the huge variation in finger force requirements, subjects automatically adapted the balance between the grip force and the load force to the object shape and maintained a constant safety margin against slips. Thus, visual cues are used to adapt force to object shape in anticipation of the force requirements imposed once the object is contacted. In the absence of tactile information, sighted subjects still adapted the force coordination

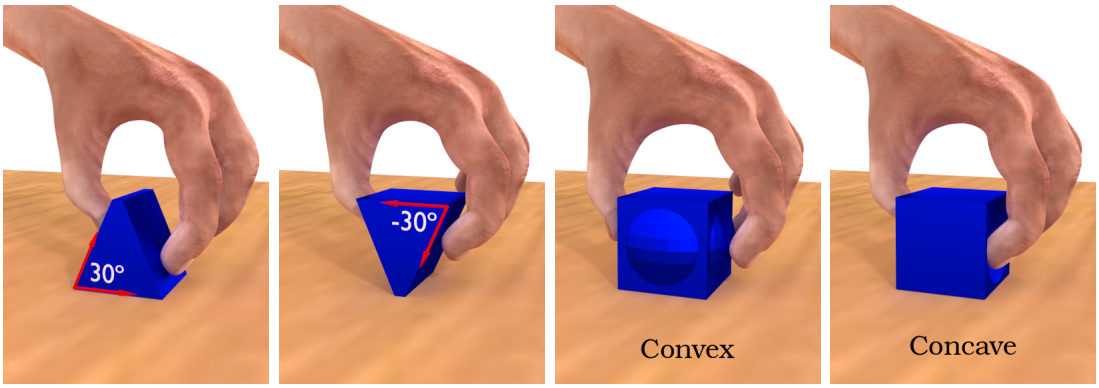


Figure 2.7: Objects used for shape based human grasping experiments. Tapered objects with angles of 30° and -30°. In the experiment the angles of the tapered object were varied in steps of 10° from -40° to 40°. Concave curved objects with radii: 20 and 40mm. Convex curved objects with radii: 20, 10 and 5mm.

to the object shape, but without vision and tactile input the performance was severely impaired. With normal digital sensibility, subjects adapted the force coordination to the shape even without vision (see Fig. 2.8).

The authors conclude that both visual and somatosensory inputs are used in conjunction with sensorimotor memories to adapt force output to the object shape automatically for grasp stability. Unlike for the frictional properties adaptation, the visual cue seems to dominate the force coordination regarding the object shape.

2.3.5 Grasping objects with fingertip anaesthesia

To assess the importance of the tactile sensory cue, there are some grasping experiments that applied anaesthesia to the fingertip tactile afferents and observed the subjects performance at grasping objects [Häger-Ross and Johansson, 1996] and reacting to external perturbations [Johansson and Westling, 1984] and to unexpected frictional properties [Edin et al., 1992].

The experiments performed by [Johansson and Westling, 1984] have been detailed in Section 2.3.1 and the experiments by [Edin et al., 1992] have been shown in Section 2.3.2. For the experiments of [Häger-Ross and Johansson, 1996], 9 healthy right handed subjects were instructed to grasp an object using different arm configurations. The object was pushed or pulled by an external force and the subjects had to keep it steady. Each subject ran 30 trials, 10 with the forearm fixed, 10 with the hand fixed (only fingers were able to move) and 10 with the whole arm free.

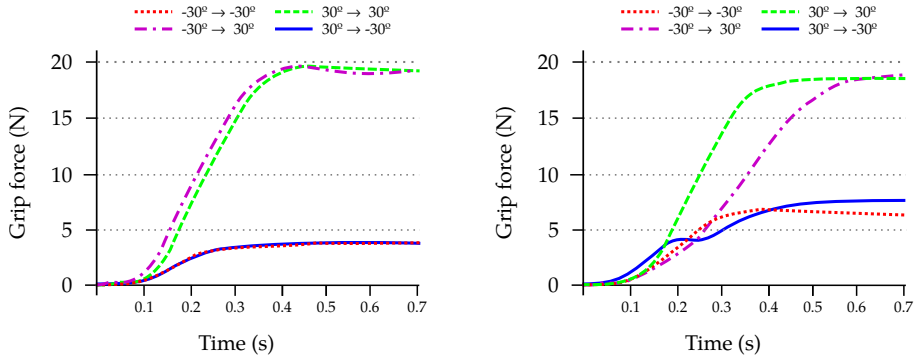


Figure 2.8: Adaptation of motor output to different object shapes. Grip forces of human grasping experiments with different object shapes. Left: with vision. Right: without vision. Results extracted from [Jenmalm and Johansson, 1997].

Results and conclusions

Although the subjects were able to grasp the objects and react to external forces, the grip force profiles were far from being optimal and the safety margins were large. The adjustments shown by the subjects to adapt to the frictional properties of the objects did not occur during finger anaesthesia. Thus the detection of the frictional properties is only tactile based.

Grip force control was dramatically reduced in the absence of tactile information. However the arm free trials showed better results than the wrist and hand fixed trials. That indicates that the proprioceptive information was used in the absence of tactile information. The performance was always worse than with the tactile input available. The authors conclude that the tactile afferents drive grip responses but when the most reliable sensory input is not available, other cues are combined to try to deal with the task as best as possible.

The subjects showed faster and more accurate reactions when the object was pulled away. It reflects preparation of a default response to the slips occurring in that direction. Reaction times when the object was pushed towards the hand were slower and the proprioceptive cue was used in combination to the tactile afferents to detect and adapt to that kind of perturbations.

2.3.6 Human corrective actions experiments

The experiments already presented, have shown that corrective actions are an important part of human grasping. Moreover their authors state that “corrective actions are highly task and phase specific and are presumably learned with the learning of the underlying action-phase controller”. However there is no clue on how the corrective

(a) (b) (c) (d) (e)

Figure 2.9: Human corrective movements extracted from the blind grasping experiment video. a) Hand moves towards the bin to touch and grasp an object. b) Initial palm contact with an object. c) Hand uses the palm contact as a pivot to rotate the wrist and obtain finger contact. d) The hand slides over the object looking for a stable grasp. e) Final stable grasp.

actions are performed. In order to take inspiration about how corrective actions were performed during the grasping phase, we conducted a simple informal experiment. The idea was to observe how humans perform corrective movements to extract ideas for an implementation on a robotic platform.

The informal experiment consisted of a box full of unknown objects that should be emptied by the subject that was standing in front of the table that was supporting the box. The subjects were 1 male and 4 females from 12 to 60 years old. The experiments were performed without any instrumentation on objects or subjects. The only data recorded was video. Figure 2.9 shows a corrective action detected after analysing the video. The figure shows an unpredicted contact and the subsequent motions to adapt, and slide over the object surface to acquire a stable grasp. Despite the inspiration taken from this informal experiments, more experiments with more subjects and proper instrumentation should be performed. Targeting the corrective movements, the mechanisms that allow humans to detect and perform corrections while executing higher level tasks could be modelled.

Results

Through the observation of the video sequences, we realized that the corrective actions taken by all of the subjects were consistent. There is a common strategy that slides over the surface of the object until free space is detected and the fingers can be opposed to grasp. It looks like the hand is reconstructing haptically the surface of the object to look for stable grasps, on the other hand it could be just a reactive strategy and the hand is adapting to the shape of the object. The results of these experiments were the inspiration for the reactive grasp controller presented in Sec. 3.3.1.

2.4 From human to robot manipulation

The experiments presented in the previous section were designed to understand how human sensorimotor control of manipulation is accomplished. After studying the results, the authors of the experiments identified several elements that contribute to human grasping:

- Action phase controllers
- Contact events
- Sensor fusion
- Contact event prediction
- Corrective actions

This five elements, together with object perception, constitute the building blocks of the work presented through this thesis. Each of the elements is detailed in the next sections of this chapter.

2.4.1 Action phase controllers

Object manipulation tasks typically involve a series of action phases in which objects are grasped, moved, brought into contact with other objects and released. These phases are usually bound by mechanical events that are subgoals of the task. Each phase accomplishes a specific goal or subgoal of the task.

A given object manipulation task can be represented as a set of sensory goals in one or more sensory modalities [Flanagan et al., 2006]. The implementation of such a plan requires the selection and execution of a corresponding sequence of basic actions to achieve the sensory goals.

The representation of the task performed by the subjects of the human grasping experiment is depicted in Fig. 2.10. The representation uses the concept of action phase controllers and the contact events to define the whole manipulation task. This concept inspired the development of the manipulation primitives paradigm. A manipulation primitive is a single reactive controller, designed to perform a specific primitive action on a particular embodiment. The manipulation primitive paradigm is detailed in Chapter 3.

2.4.2 Contact events

Contact events encode the making and breaking of contact between either the fingertips and the grasped object or the object in hand and another object or surface. The contact events provide information related to the functional goals of successive action phases. They have a crucial role in the sensorimotor control of manipulation.

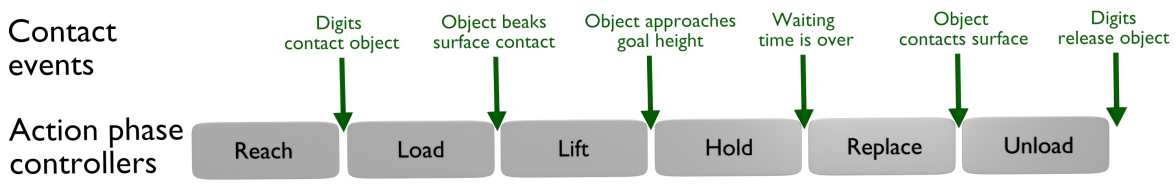


Figure 2.10: Action phase controllers used for the human grasping experiments, consisting on grasping, lifting and replacing the test object on the supporting surface. Arrows represent the event that triggers the transition from one action phase to the next one.

Contact events contribution to sensorimotor control of human manipulation is threefold. First, by comparing actual and predicted contact events, the task can be monitored and, if prediction errors arise, trigger corrective movements to respond to the unexpected events accordingly. Second, contact events give rise to salient sensory signals, they provide an opportunity for sensorimotor integration and sensor fusion. Third, the predicted consequences of contact events can directly furnish initial state information for subsequent phases of the manipulation tasks, this enables smooth transitions between different phase controllers [Johansson and Flanagan, 2010].

2.4.3 Sensor fusion

Contact events could be detected using many sensory cues, vision, force, touch or even audio. Moreover, the contact events can also be inferred on other sensory cues such as vision or proprioception. Despite the use of many different sensory cues, contact events provide a good opportunity for sensor integration, providing a common stimulus to be matched in all the perceptual modalities. The mechanisms used for contact event detection and sensor fusion are detailed in Chapter 4.

2.4.4 Contact event prediction

In object manipulation the brain not only forms action plans in terms of desired subgoals but also predicts sensory events that signify goal attainment in conjunction with the generation of motor commands, see Fig 2.11. By comparing predicted sensory events with the actual sensory events, the motor system can monitor task progression and adjust subsequent motor commands if errors are detected. The implementation of a contact event prediction engine is discussed in Chapter 5.

Contact events can function as sensorimotor control points in both actors and observers. Sensations caused by our own actions are attenuated to increase the salience of sensations with an external cause. Such perceptual cancellation could explain why we cannot tickle ourselves and why externally imposed constant forces applied to the fingertip are

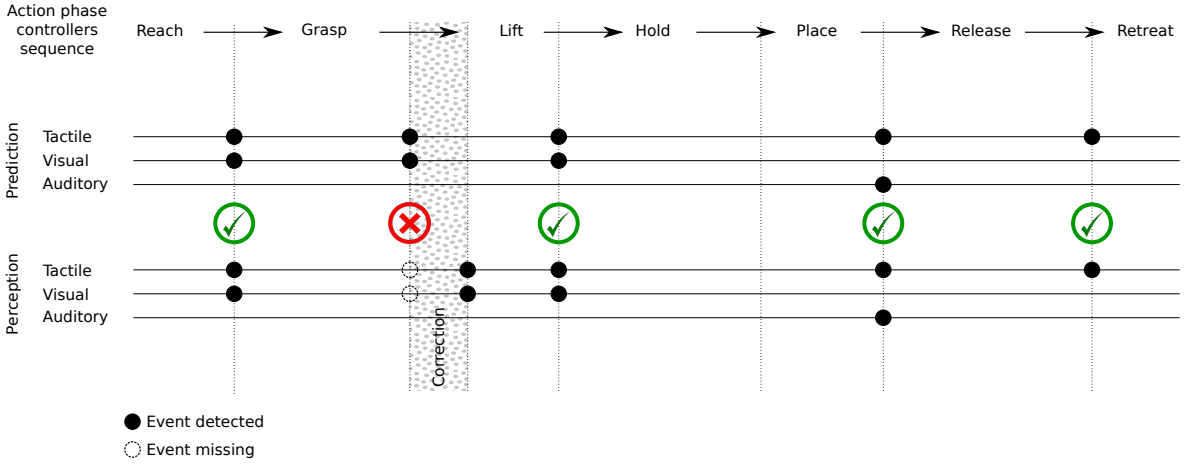


Figure 2.11: Contact events are predicted, and the actual sensory input is compared with the predictions. Prediction errors trigger corrective actions. The figure shows the response of the sensors and the predictions on each sensory cue. There is a prediction error, the contact between the object and the surface does not break when it was predicted, in this case the correction only consists on increasing the load force until the object lifts off and the expected events are detected.

perceived as more intense than the same forces applied by ourselves. Perceptual attenuation is linked to specific contact events arising from movement rather than the movement itself, [Flanagan et al., 2006].

2.4.5 Corrective actions

The resulting mismatch between the expected sensory event and the perceived sensor signals triggers a learned corrective action pattern that depends on the action phase controller and that is learned together with the learning process of the action controller. Moreover, it leads to the updating of the representation of the object properties in memory. Thus in this situation the sensorimotor system reacts quickly to both the presence of an unexpected contact event and the absence of an expected sensory event. In the presence of misleading cues, updating might require repeated action executions or movements of the target object, [Johansson and Flanagan, 2008].

As suggested by the fast reaction times in humans, the adaptation is encoded in the controller itself. All the manipulation primitives implemented in Chapter 3, are able to detect, react and adapt to unexpected sensory inputs performing corrective movements. However, there are some mismatches that should require a higher level response, such as replanning or reasoning. This can be modelled by a hierarchical corrective action schema which is part of the future work of this thesis.

2.5 Conclusions

This chapter presented the neuroscience experiments and theories that are used as the foundations of the contact based robot manipulation system presented in this thesis.

Starting from the huge number of experiments performed by the neuroscience community regarding human grasping and manipulation, we have classified the different types of experiments, selecting those suitable to provide inspiration for the implementation of a robot manipulation framework. Motivated by the lack of experimentation related to corrective movements, we have conducted an informal experiment to take inspiration about corrections in order to provide ideas for the implementation of such movements in a robotic setup. However, more experimentation on that direction should be conducted in order to provide better models and more details about how human corrective movements are performed. The results could be used to enhance the computational models that are used on the presented framework.

After analysing the results of the reviewed experiments, we have highlighted the importance of contact detection and contact events during human manipulation. Thus, we have focused our development on a contact based manipulation framework. Moreover, the building blocks of the framework were identified. Each of them is detailed in a chapter of this thesis where they are implemented and validated on different robotic platforms.

Although from the neuroscience experiments we were able to extract the main components necessary to implement a manipulation system, the results of the experiments analysed do not provide any hints or guidelines about how to implement them. Whether there is a set of latent abilities (action-phase controllers) that are refined during the development of the subject or everything is learned from scratch is an unknown. In this thesis we have implemented each building block without using a learning approach. However, learning techniques can be applied for the implementation of manipulation based controllers as discussed in Chapter 3 and also for prediction as discussed in Chapter 5.

Chapter 3

Manipulation primitives

As detailed in Chapter 2, neuroscience studies concluded that humans likely perform tasks as a set of different action-phase controllers that attain task subgoals. Experimental results from Chapter 2, suggest that humans learn a set of corrective reflexes that are triggered automatically during the execution of action-phase controllers in order to adapt to unexpected events.

This chapter, inspired by the action-phase controllers, proposes the paradigm of manipulation primitives, a tool for modelling and execution of reactive manipulation actions. Manipulation primitives constitute a vocabulary of atomic sensor-based actions, which can be coordinated using graphical methods to describe complex tasks.

We define a *manipulation primitive* as a single reactive controller, designed to perform a specific primitive action on a particular embodiment. Each primitive is parametrized to allow it to be used in different situations. A focused control policy, which uses the available sensor feedback, is then used to achieve predefined success or failure conditions.

The strength of the manipulation primitives paradigm is demonstrated by developing a set of primitives for object transport and manipulation. After providing the implementation and testing of several basic primitives, two examples of a complex task composed by those primitives are shown.

The embodiment independence that this paradigm enables, is detailed and discussed in Chapter 6 together with the main system architecture.

3.1 Related Work

3.1.1 Manipulation primitives

The idea of control primitives is not new in robotics, and particularly in robot grasping. Earlier works propose individual control primitives for different problems such as to control a hand [Speeter, 1991], to define object movements [Michelman and Allen, 1994] and its relations [Morrow and Khosla, 1997] and to control a manipulator [Hasegawa et al., 2003]. Despite different definitions of primitives, all of them present a common

trend, discretizing and reducing the complexity of controlling a robotic setup by reducing the search space for planning. Other similar approaches include Object Action Complexes [Krüger et al., 2011] and the physical interaction framework of [Prats et al., 2010].

An apparently similar concept are Dynamic Movement Primitives (DMPs) introduced by Schaal et al. [Schaal, 2006]. DMPs describe motion trajectories by means of differential equations, which can be adapted to several situations by adjusting a few of the equation parameters and are deeply interlinked with motor control. This framework has proved to be very effective to represent standardized arm movements, which can be learned by human observation or demonstration [Schaal et al., 2005], and can even be adapted reactively as the scene is observed to change. However, they are basically different idea of primitives presented in this chapter. DMPs are focused on motion description in a lower level, while our primitives describe robot manipulation skills.

3.1.2 Sensor based control

As suggested in Chapter 2, corrective reflexes are a mechanism used by humans to adapt to unexpected events while manipulating objects. An approach to address the problem of unknown environments is to use sensors, for example vision, to build the necessary models. Vision has been used to obtain the shape of unknown target objects [Morales et al., 2006, Aarno et al., 2008] and to determine the location and pose of objects [Azad et al., 2006]. In both cases, visual input was used to plan feasible grasps. On the other hand, visual feedback can also be used on-line during reaching for an object. [Murphy et al., 1993] uses visual techniques to correct the orientation of a four-finger hand while approaching an object to improve contact locations.

Once contact between object and robot has been reached, tactile and force sensors can be applied. Tactile measurements can be used to estimate the quality of grasps [Coelho Jr. and Grupen, 1997, Platt et al., 2002, Mouri et al., 2007, Bekiroglu et al., 2011] or the shape of an object [Allen and Roberts, 1989] with the purpose of reaching better contact locations through a sequence of grasping/regrasping actions. Contact information can also be used to program complex dexterous manipulation operations like finger repositioning while holding the object [Coelho Jr. and Grupen, 1997, Huber and Grupen, 2002]. Several works have combined the use of several sensors to complete the process of grasp planning and execution [Allen et al., 1997, Grzyb et al., 2008].

3.2 Manipulation primitives framework

A *manipulation primitive* is a single reactive controller, designed to perform a specific primitive action on a particular embodiment. However, there are primitive actions that do not involve physical interaction but perception. Hence, we define *perceptual primitives* as sensor based processes focused to obtain information from the environment or

detect events. *Events* represent the detection of a specific perceptual or internal condition. Primitives are together with events the elementary symbols of a vocabulary that is used to describe tasks. *Tasks* are defined as a cyclic, directed, connected and labelled multi-graph where the nodes correspond to primitives (or other tasks) and the edges to events. It is important to highlight that a task, can also have as nodes, not only primitives but other tasks. A *task* is usually a semantically meaningful goal, such as emptying a grocery bag or clearing a table. An example of task definition for clearing a table is depicted in Fig. 3.1.

The main concepts of the manipulation primitives paradigm are summarized in the following list:

Manipulation primitive A reactive controller, designed to perform a specific primitive action on a particular embodiment.

Perceptual primitive A sensor based process focused to obtain information from the environment or detect events.

Events Represent the detection of a specific perceptual or internal condition and are triggered by manipulation or perceptual primitives.

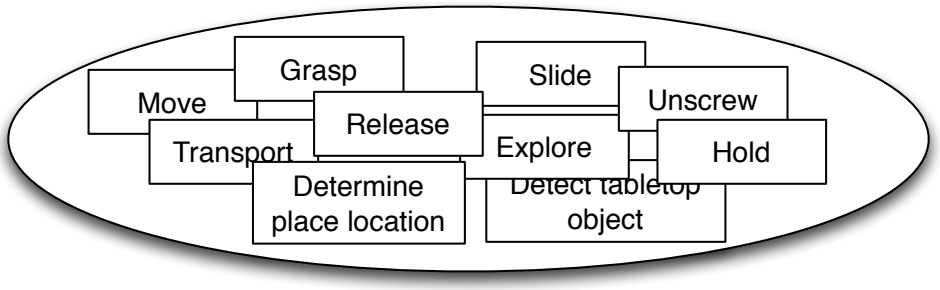
Task A cyclic, directed, connected and labelled multi-graph where the nodes correspond to *manipulation primitives* or *tasks* and the edges to *events*.

Plan An instance of a *task* with the parameters set for a specific execution in a determined scenario and context.

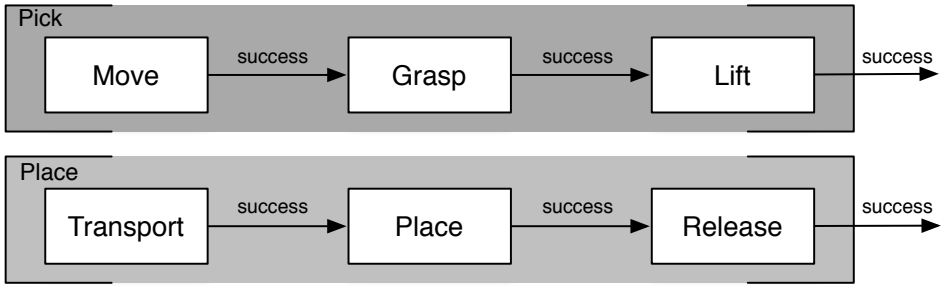
Manipulation primitives are parametrizable, thus, a task planner requires some information (i.e. parameters) to tune and instantiate tasks for a specific scenario. The description of such a planner is out of the scope of this thesis but its outcome must be a composition of parametrized primitives to address the target scenario (See Fig. 3.1). Finally, primitives can finish with several degrees of accomplishment, which in its more simple expression would be an event from the pair *Success/Failure*.

Although some of the parameters are set by the task definition itself, there are still some others that must be defined for a specific scenario and environment state. Moreover, the parameters can be defined and changed on-line according to the result of other primitives or internal conditions. An instantiation of a task with defined parameters according to the current scenario is called a *plan*.

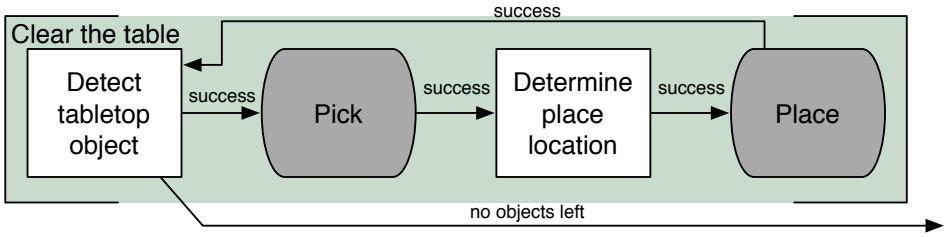
In this thesis, the focus is not in pure perception but in sensor based manipulation. However, a set of *perceptual primitives* is necessary for high level task definitions. For example, an important role of the perceptual primitives is the object detection, recognition and scene understanding that can lead the parameter tuning for other subsequent primitives during the execution of a plan.



(a) Primitives are the elementary symbols of a vocabulary used to compose tasks.



(b) Simple tasks can be composed combining manipulation primitives connected by events.



(c) Definition of the clear the table task using pick and place sub-tasks and perceptual primitives.

Figure 3.1: Composition of the clear the table task. Starting with the vocabulary of primitives, the primitives are connected by events to form simple pick and place tasks. Then the pick and place tasks are used in combination with more primitives to define the clear the table task.

Primitives are embodiment specific. However, embodiments with similar capabilities allow the definition of primitives with similar behaviour and purpose, which can be thought as *abstract manipulation primitives*. The focused purpose of primitives simplifies the development of equivalent primitives on several embodiments. This equivalence also enables the transmission and execution of plans between different embodiments. The abstract manipulation primitives can then be used to describe *abstract actions*. The abstraction mechanisms are described in Chapter 7.

3.3 Sensor-based primitives for manipulation

Table 3.1 shows the implemented vocabulary of primitives, their description and some examples of tasks where the primitives could be used. This set of primitives is sufficient to deal with many of the manipulation tasks a robot may encounter in household, real-life scenarios. However, there are potentially many other manipulation primitives that could be implemented, to enable a robot to perform a wider variety of tasks, or improve the performance of some of them. Some examples to extend the presented primitive vocabulary are push, pull and button pushing. Moreover, focused primitives for specific tool use, would further extend the tasks that the robot is able to perform.

Primitive	*Short description*	*Task examples*
Grasp	Secure a stable grasp that rigidly attaches an object the hand	Clear the table, empty a box
Transport	Move a grasped object from the a starting to a target position	Clear the table, bring water
Place	Move the grasped object towards a supporting surface to place it	Set the table, load the dishwasher
Release	Release a grasped object opening the hand and moving the arm if necessary	Pour water, load the dishwasher
Slide	Slide an object over a surface towards the target position	Wipe the table
Explore	Move the hand towards a specified direction and stop when a contact is detected	Empty a grocery bag
Unscrew	Perform a series of grasp and twist movements on an object cap to unscrew it	Pour water

Table 3.1: Description of the implemented vocabulary of manipulation primitives.

Name	*Parameters*	*Control and sensor requirements*
Robust grasp	Pregrasp size, grasp preshape	Arm control, FT and tactile sensors
Transport	Obstacles, trajectory, constraints	Arm control
Place	Contact threshold	Arm control, Force-torque sensor
Release	Hand position	Arm control
Slide	Slide force threshold	Arm control, Force-torque sensor
Explore	Direction, Contact threshold	Arm control, Force-torque sensor
Unscrew	Grasp force, pull force threshold, unscrew angle	Arm control, FT and tactile sensors

Table 3.2: Implemented manipulation primitives, parameters and requirements.

The primitives are described independent of a particular hardware platform but a set of control and sensor requirements are needed to implement each one of them (See Table 3.2).

All the primitives are parametrizable, requiring one common parameter: an approach vector (6D pose). It will be used conveniently depending on the primitive purpose. All the implemented primitives, together with some of their optional parameters are listed in Table 3.2. The primitives detailed in this section were implemented for either the right (Barrett Hand) or left (Schunk SDH2 Hand) hands of the UJI Humanoid torso: Tombatossals. See more details about the robotic platform in Appendix A.1.

3.3.1 Grasp primitive

Object manipulation requires direct or indirect interaction with objects. Although objects can be manipulated using tools, non-prehensile manipulation or caging grasps, it usually requires to establish a rigid relation between the target object and the robot end effector. In that scenario, the manipulated object is usually attached to the robot end effector after a grasp execution.

Regardless of the grasping taxonomy considered (e.g. [Feix et al., 2015], [Cutkosky, 1989]), there is a wide variety of grasp types. However, this primitive provides a generic grasping ability that can be used for the main grasping requirements of a robot. For other types of non-prehensile manipulation, specific grasps for tool use or control panel interaction, other focused primitives should be used.

The simplest implementation of a grasp primitive would consist of closing the robot hand. It has, however, been demonstrated that by using sensor based methods the success rate of this primitive can be increased significantly [Felip and Morales, 2009]. We propose a novel sensor based controller that performs several corrective movements in order to achieve a stable grasp. The purpose of the corrective movements is to place the hand in a position that is more likely to produce a stable grasp than the initial position. For the implementation of corrections, inspiration was taken from the human experiments presented in Sec. 2.3.6.

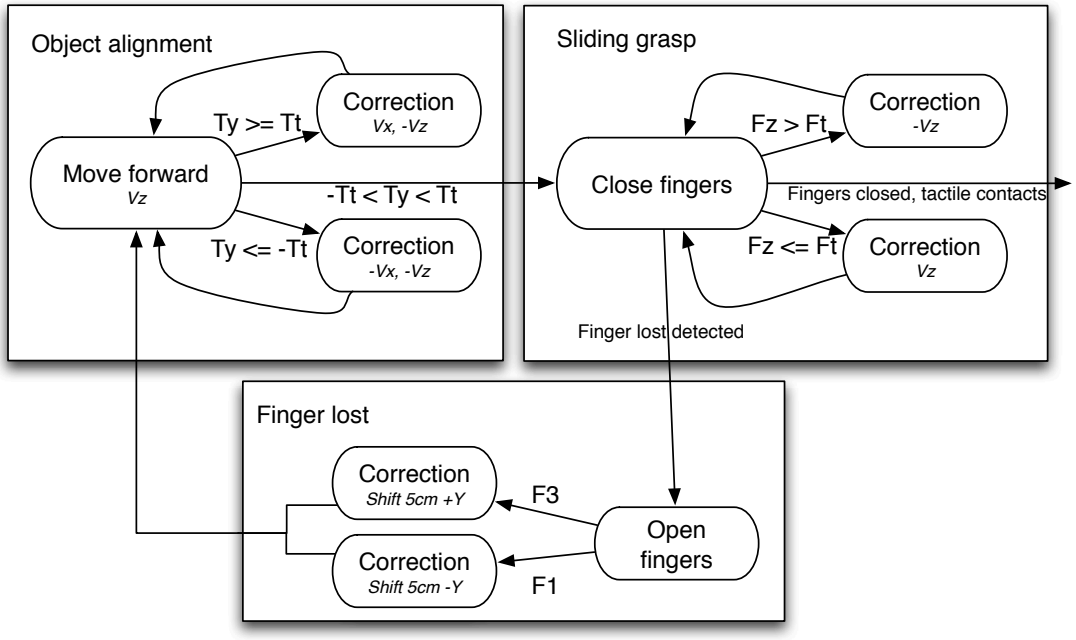


Figure 3.2: Robust grasp primitive. Fz is the force in the Z hand axis, Ty is the torque on the Y hand axis, Tt is the Y torque threshold, Ft is the Z Force threshold, Vz is a velocity in the Z hand frame. Vx is a velocity in the X hand frame. Important reference frames are depicted in Fig. 3.3.

Primitive description

The initial assumption of the grasp primitive is that the hand is near the object, at a close distance. If the starting position of the hand has been carefully planned and the positioning in the vicinity of the object executed accurately, the hand should only move towards the specified grasp direction and close to obtain a stable grasp of the object. Unfortunately, this is not often the case, a series of corrective movements are thus performed in order to obtain a robust grasp. These corrections are executed sequentially and divide the primitive execution into three main phases: alignment, sliding grasp and force adaptation (See Fig. 3.2).

A previous version of this primitive was published in [Felip and Morales, 2009]. A similar strategy is used by [Kazemi et al., 2012] also showing robustness and better grasp performance under uncertainty than non-reactive approaches.

The corrections are performed depending on the estimation of the location of the detected contact. The implementation of this primitive is preshape independent, thus the

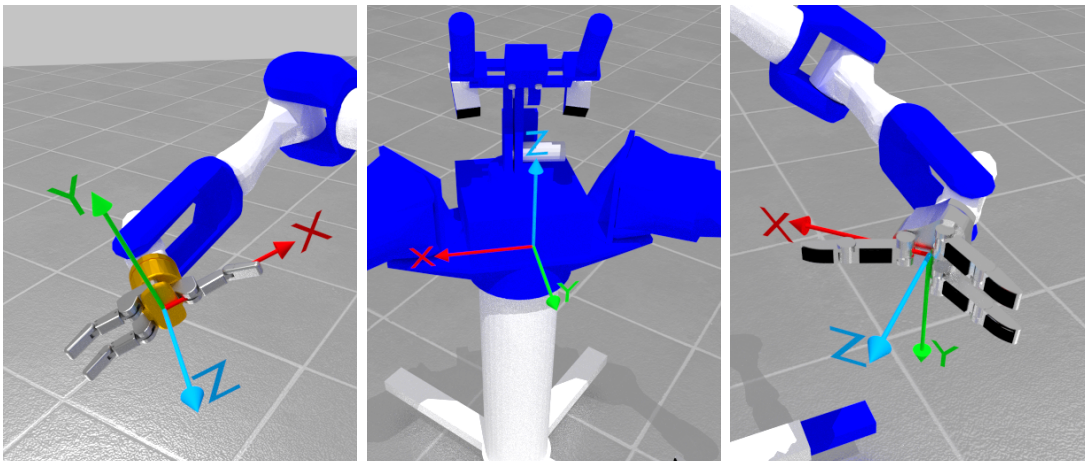


Figure 3.3: Robot important reference frames. Left: Right hand reference frame. Center: Robot base reference frame. Right: Left hand reference frame.

grasp type used (if not set by the PREGRASP_TYPE parameter) will depend on the hand configuration in the instant when the primitive takes control.

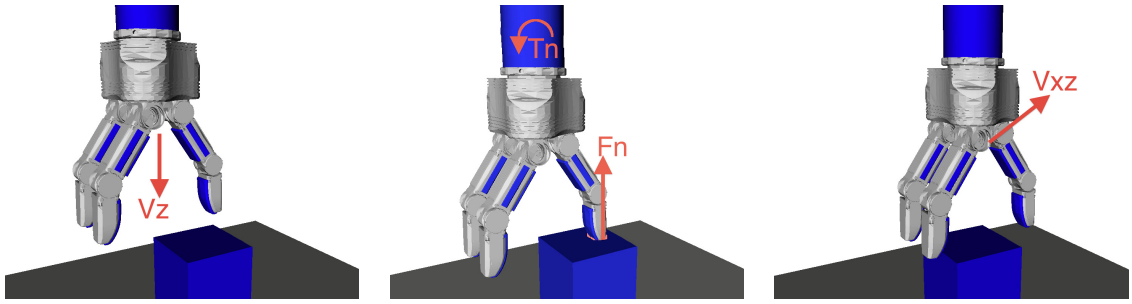
Alignment

In some situations, the initial starting position and the grasp direction result in the hand not correctly facing the center of the object (Fig. 3.4a), and thus there is a premature collision during the approaching (Fig. 3.4b). This contact is detected using a wrist mounted force-torque sensor. Using the torque, the contact point is estimated and a correction is performed to center the object (Fig. 3.4c).

$$\vec{v} = \begin{cases} \vec{v_x} - \vec{v_z} & \text{if } T_y > T_{treshold} \\ -\vec{v_x} - \vec{v_z} & \text{if } T_y < -T_{treshold} \\ \vec{v_z} & \text{otherwise} \end{cases} \tag{3.1}$$

Eq. (3.1) shows the controller that performs the alignment phase of the robust grasp primitive. Where $\vec{v}$ is the resultant velocity that is applied by the controller to the hand (w.r.t hand frame) and is a combination of $\vec{v_x} = (V_x, 0, 0)$ and $\vec{v_z} = (0, 0, V_z)$. Where V_x and V_z are the parameters that control the speed of the corrections produced by the controller movements. T_y is the torque around y-axis (w.r.t. hand frame). The alignment is finished when a contact is detected and $-T_{treshold} \leq T_y \leq T_{treshold}$

An example of an execution of this phase is depicted in Fig. 3.4. The contact can be also detected using the tactile sensors available. Alignment correction improves grasping of objects with location uncertainty by allowing the hand to align its center with the



(a) (b) (c)

Figure 3.4: Grasp primitive: Alignment phase. (a)Arm moving towards the object. (b)Contact generates torque in the wrist. (c)Correction movement is performed.

object. The effects of this alignment phase are analysed in Sec. 3.4 where a thorough evaluation and comparison is conducted and the results are discussed.

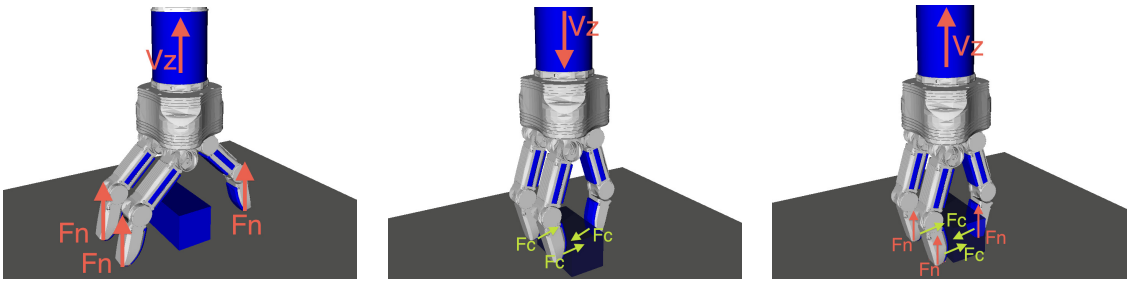
Sliding grasp

When approaching, the hand makes contact in occasions with the supporting surface instead of the object (See Fig. 3.5(a)). In this case, closing the hand can result in unsuccessful grasps especially for small objects. To counter this problem, a sliding correction is used. The corrective movement consists of moving the hand forward or backward depending on the force sensed along its Z axis. Concurrently the fingers are closing (see Fig. 3.5) to maintain stable, light contact with the supporting plane. When the fingers are no longer able to close, because the object is grasped or the fingers reach their joint limits, the sliding grasp control ends. The correction allows grasping small objects by sliding the fingers on the supporting plane until the object is securely grasped.

It is important to highlight that this phase is also suitable to grasp objects laid out on other surfaces different from tables or workbenches, such as handles on doors, drawers, dishwashers and so on.

Equation 3.2 describes the arm cartesian velocity control used meanwhile the fingers are closing where $\vec{v}$ is the velocity control sent to the arm, $\vec{v_z}$ is a velocity in the Z axis of the hand. Fz_{sensor} is the current force in Z axis of the hand read by the sensor and $Fz_{threshold}$ is the force threshold.

$$\vec{v} = \begin{cases} -\vec{v_z} & \text{if } Fz_{sensor} <= -Fz_{threshold} \\ 0 & \text{if } -Fz_{threshold} < Fz_{sensor} < Fz_{threshold} \\ \vec{v_z} & \text{if } Fz_{sensor} >= Fz_{threshold} \end{cases} \tag{3.2}$$



(a) (b) (c)

Figure 3.5: Grasp primitive: Sliding grasp phase. (a)The fingers contact the table while closing and the normal force Fn is detected by the force sensor. Thus the controller sets the velocity to $-\vec{v_z}$ in order to move the hand back. (b)The fingers are closing and although the fingers touch the object exerting the contact force Fc, the contact with the table is lost. Therefore, $\vec{v_z}$ is set forwards. (c)The hand contacts the table again but the object is already grasped and the fingers can no longer keep closing, hence the sliding grasp phase ends.

The behavior of this correction phase is shown in Fig. 3.5. The hand starts closing and when the fingers make contact with the surface, the force they are applying is detected in the wrist, thus the arm moves back (Fig. 3.5(a)). The fingers continue closing and because no contact force is detected, the arm moves forward (Fig. 3.5(b)). In Fig. 3.5(c) the fingers are not able to close anymore and the sliding grasp ends.

Finger lost correction

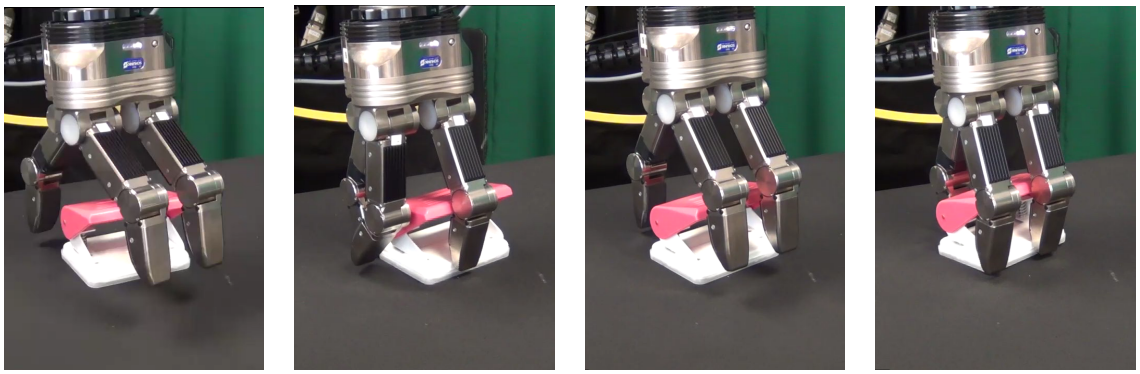
It may happen, if the error is big enough, that the hand closes and one of the external fingers lose contact with the object. In this case, the *finger lost* corrective movement is triggered, the hand opens and shifts towards the detected contacts in order to place all the fingers on the object. Fig. 3.6 shows the execution of such a corrective movement in a real experiment.

Force adaptation

The force of the fingers is increased to improve grasp stability. The primitive ends with a success if at the end the object is still in the hand, detected by the joint configuration or contact information.

Primitive parameters

- PREGRASP_TYPE (default:none): Encodes the starting configuration of the hand, for example: cylindrical, spherical or hook. If no value is specified the current hand configuration is used.



(a) (b) (c) (d)

Figure 3.6: Grasp primitive: Finger lost correction. (a) Grasping position after the alignment phase. (b) The hand closes and one finger does not contact the object. (c) The hand opens and performs a corrective movement. (d) Finally a good grasp is achieved.

- PREGRASP_SIZE (default:none): Can take a value from 0 to 1 and encodes the hand opening prior to the grasp motion.
- FORCE_THRESHOLD (default:2N): Force threshold for the contact detection using the force-torque sensor. This value has to be set depending on the force-torque sensor sensitivity.
- TORQUE_THRESHOLD (default:5Nm): Torque threshold for the contact detection using the force-torque sensor. This value has to be set depending on the force-torque sensor sensitivity.
- GRASP_DIRECTION (default:[0,0,1]): Direction to move the hand to grasp the object using the hand frame. The important frames of the robot used for the implementation are depicted in Fig. 3.3.
- MAX_DISTANCE (default:0.2m): Max distance that the hand will move towards GRASP_DIRECTION looking for a contact. When this distance has been covered, the primitive will switch to the sliding grasp step. This parameter has to be set depending on the grasp planner and the hand size.
- CORRECTION_VELOCITY (default: 0.005m/s): Velocity used to perform the corrective movements during the alignment and sliding grasp phases of the grasping primitive.

3.3.2 Transport primitive

The purpose of the transport primitive is to move the arm to a specified target position while the hand holds an object. The primitive can also be used to move the arm without an object.

The problem of controlling redundant manipulators has been widely studied in robotics [Waldron and Schmiedeler, 2008, Chiaverini et al., 2008] and nowadays there are open source tools that can be tuned to provide advanced control for custom manipulators [Smits, 2015, Corke, 2011, Sucan and Chitta, 2015]. Controllers can be classified depending on the space they are controlling (e.g. joint, Cartesian) and the control type (position, velocity or effort). In order to ease the programmer's task, it is quite common that commercial robots provide and API with different joint control modes (position, velocity, effort). However, all the controllers are finally built on top of an effort controller that is converted to a voltage sent to the motors.

Moreover, on top of the controllers used, motion planning algorithms can be used to plan collision free trajectories and perform optimal movements from point A to point B.

Primitive description

This primitive implements a Cartesian velocity controller that allows the trajectory to be constrained by specifying optional parameters. A trajectory can be specified in joint space as a list of joint positions that define the state of each joint during all the transport primitive execution. However, a less restrictive definition can be used by specifying the trajectory as a list of end-effector Cartesian positions. Instead of defining the exact trajectory that the robot must follow, it is also possible to specify position, velocity or acceleration limits in the Cartesian space.

Equation 3.3 shows how the force-torque threshold parameter is used to stop the movement if a collision is detected where x_{target} and $x_{current}$ are the target and current 6D pose vectors. n is a normalization term used to keep the resulting velocity inside the velocity limits. The resulting velocity is sent to the Cartesian velocity controller that converts the desired Cartesian velocity into joint velocities using the Jacobian pseudo-inverse approach. We assume that the robot provides a joint velocity control and the final conversion to joint efforts is done by the robot's internal controller.

$$\vec{v} = \begin{cases} n(x_{target} - x_{current}) & \text{if } ||F_{sensor}|| < F_{threshold} \\ 0 & \text{if } ||F_{sensor}|| >= F_{threshold} \end{cases} \tag{3.3}$$

Optional parameters can also be used to describe environment obstacles as an obstacle point cloud, in which case a force-field [Khatib, 1985] based collision avoidance strategy

is used to generate a collision free trajectory from current to target position maintaining the hand orientation.

For instance, if the task is to transport a mug full of water without pouring the liquid, acceleration should be constrained to a low value on all axes and the rotation velocity of the table plane axes should be set to 0 to prevent tilting the mug. If the target position cannot be reached without breaking the specified constraints, the primitive ends with a failure. In Fig. 3.7 an example of a position constrained trajectory is shown. The convex hull of the box is defined as forbidden space to define position constraints.

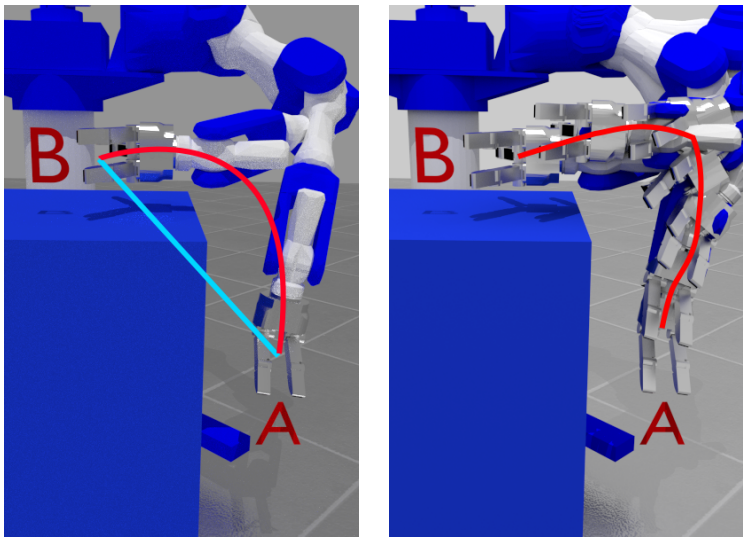
Jacobian pseudo-inverse approaches for Cartesian velocity control, are fast and can work real-time, hence they are very useful for closed loop control. However, they are gradient descent approaches and the trajectories generated may fall into local minima quite easily if the start and end positions are separated enough or the trajectories travel outside the workspace of the robot. Therefore, this control approaches are suitable for close distance movements, fine manipulation and closed loop control. For long distance movements inside the robot workspace, other approaches should be used.

As an alternative method, we have implemented an interface to well known path planning libraries such as MoveIt! [Sucan and Chitta, 2015]. In this case, instead of using an online gradient descent strategy, the trajectory is calculated in advance and a set of waypoints is provided. The plan is executed taking into account the force-torque threshold to satisfy collision security constraints. Figure 3.7(b) shows the result of calculating a motion plan with MoveIt! path planning. There is a wide variety of planning algorithms that can be used, the plugin architecture of the planning library used by MoveIt! makes it simple to switch between planners and use the best for each application. For this primitive implementation we have used the KPIECE planner [Sucan and Kavraki, 2012], it is a tree-based planner that uses a discretization to guide the exploration of the continuous space. Their authors claim that this planner has been shown to work well consistently across many real-world motion planning problems.

The selection of the method used to move the robot, can be automatically selected depending on the distance to the target. The PLANNING_DISTANCE_LIMIT parameter is used to decide which method to use, targets further that the configured distance will trigger the planning algorithm. Moreover, if the target is close but a local minima is detected while executing the movement, the control is switched to the motion planning in order to recover, if the planning fails the primitive throws an error event.

Primitive parameters

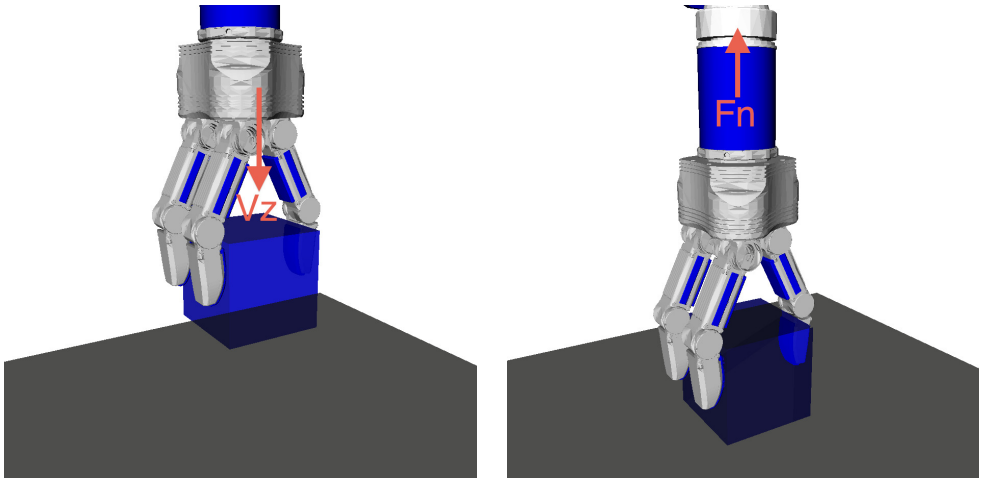
- FORCE_THRESHOLD (default: 15N): Force threshold to consider an unexpected contact.



(a) (b)

Figure 3.7: Transport primitive. (a) Example of execution of the constrained transport primitive from the starting point A to the target point B. Blue line: Standard trajectory. Red line: Position constrained trajectory. (b) Example of a calculated plan using MoveIt! planning library.

- ACCELERATION_LIMITS (default: [0,0,0,0,0,0]): End effector linear and angular acceleration limits in m/s^2 and rad/s^2. If all the values are zero, no constraints are taken into account.
- VELOCITY_LIMITS (default: [0,0,0,0,0,0]): End effector linear and angular velocity limits in m/s and rad/s. If all the values are zero, no constraints are taken into account.
- OBSTACLES (default: none): Point cloud representing obstacles. Obstacles can also be specified by geometric primitives.
- WAYPOINTS (default: none): End effector 6D pose waypoint list. The primitive will move from the starting position to the target position through the specified waypoints.
- PLANNING_DISTANCE_LIMIT (default:0.03m): Minimum distance to the target that will trigger the use of a planning approach to move the arm instead of the on-line Cartesian velocity control.



(a) (b)

Figure 3.8: Place primitive. (a)Arm moving the object towards the surface with velocity $\vec{v_z}$. (b)The object contacts the supporting surface and the normal force Fn is detected by the wrist force-torque sensor.

3.3.3 Place primitive

Placing an object involves a supporting surface that usually is a horizontal plane. However, that is not always the case, the plane can be slanted or be vertical if the task is, for example, to stick the object to the wall. Moreover, the supporting surface could not be a plane (e.g. bowl, dish). The place primitive is used to gently place an object on a supporting surface asserting the success using on-line sensor feedback.

Primitive description

The arm moves towards the supporting surface until a contact is detected. Equation 3.4 describes the control action taken depending on the force sensor readings where $v_{\vec{place}}$ represents a constant velocity defined by the VELOCITY parameter and the SURFACE_NORMAL parameter, $v_{\vec{place}} = \vec{n} \cdot v$. The primitive ends when the arm stops. This primitive can be configured with an optional parameter $F_{threshold}$ defining the force threshold needed to detect a contact. An example execution of this primitive is shown in Fig. 3.8.

$$\vec{v} = \begin{cases} -v_{\vec{place}} & \text{if } ||F_{sensor}|| < F_{threshold} \\ 0 & \text{if } ||F_{sensor}|| >= F_{threshold} \end{cases} \tag{3.4}$$

Primitive parameters

- FORCE_THRESHOLD (default: 5N): Force threshold to consider that the supporting plane has been contacted.
- SURFACE_NORMAL (default: [0,0,1]): Normal of the point of the supporting surface where the object will be placed. This will be used to determine the motion direction.
- VELOCITY (default: [0.005 m/s]): Linear velocity towards the supporting surface.

3.3.4 Release primitive

The purpose of this primitive is to release the object from a previous grasp. Releasing an object can be difficult if the fingers, while opening, collide with the supporting plane or other parts of the object (see Fig. 3.9(a)). To handle this problem, the release primitive opens the hand smoothly while the arm moves back if a contact with the environment is detected.

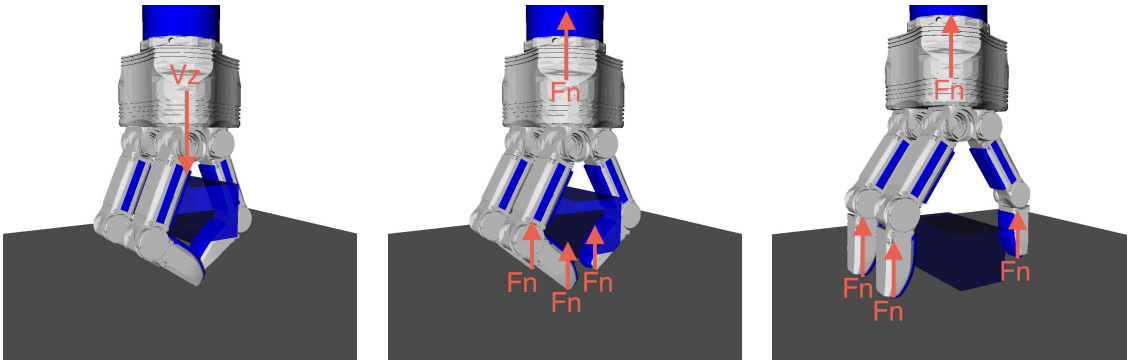
Primitive description

The movement of the arm is force-controlled and the arm only moves back if there is a contact detected between the opening fingers and the surface (see Equation 3.5). The sequence of movements that this primitive performs is shown in Fig. 3.9. This primitive can be configured by setting the target hand position after release and the $F_{threshold}$ parameter.

$$\vec{v} = \begin{cases} -\vec{v}_z & \text{if } Fz_{sensor} < -F_{threshold} \\ 0 & otherwise \end{cases} \tag{3.5}$$

Primitive parameters

- HAND_TARGET_POSITION(default: 1.0): Defines the target hand opening to consider the release primitive finished. 0.0 = fully closed, 1.0 = fully open.
- FORCE_THRESHOLD (default: 5N): Force threshold to consider that the fingers contact the supporting plane while opening.
- SURFACE_NORMAL (default: [0,0,1]): Normal of the supporting plane. This will determine the motion direction if the fingers collide with the surface in order to move the hand away from the contact.
- VELOCITY (default: [0.001 m/s]): Linear velocity used to move the hand away from the supporting surface.



(a) (b) (c)

Figure 3.9: Release primitive. (a)Hand before opening the fingers. (b)The hand cannot release the object, the fingers are blocked by the surface and cannot continue opening. The normal force F_n in each finger propagates to the wrist. (c)The hand moves back and continues opening the fingers. The object is released successfully.

3.3.5 Slide primitive

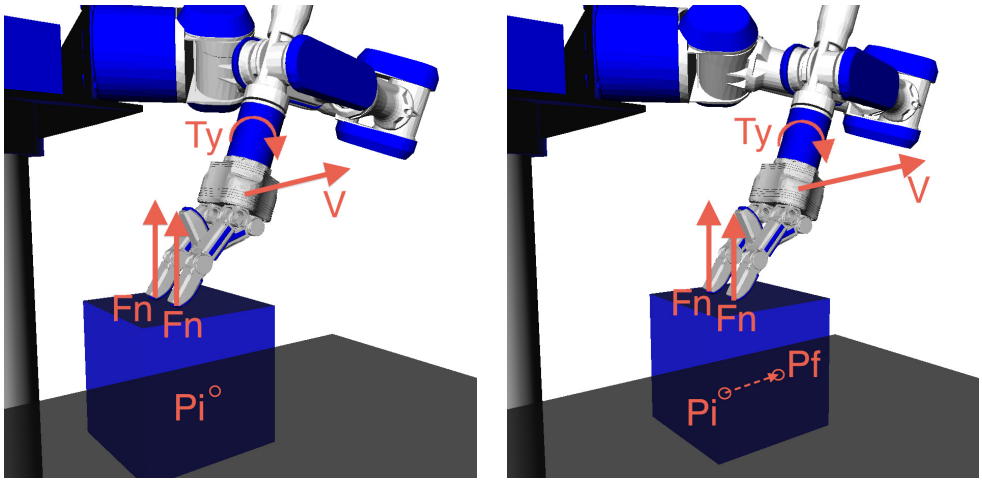
A very frequent type of non-prehensile manipulation is sliding objects on a supporting plane. The purpose of the slide primitive is to provide the robot with such an ability by exerting a constant force on an object against its supporting plane and sliding it to a target position.

Primitive description

Using force control the arm applies a force (F_n in the desired range $F_{min} < F_n < F_{max}$) to the object, then moves towards the target, keeping the applied force constant (Fig. 3.10(a)). The constant force F_n the arm and object allowing the robot to slide the object on the surface from the starting to the target position (Fig. 3.10(b)).

Equation 3.6 shows the control law that keeps the force applied to the object in a desired range while it moves the arm towards the target position. Only the target position is a required parameter, but the applied force can be configured by setting a desired force range defined by F_{min}(LOWER_FORCE_THRESHOLD parameter) and F_{max} (UPPER_FORCE_THRESHOLD parameter). This primitive uses a predefined hand preshape (see Fig. 3.10).

$$\vec{v} = \begin{cases} x_{target} - x_{current} & \text{if } F_{min} < ||F_n|| < F_{max} \\ -\vec{x_z} & \text{if } ||F_n|| <= F_{min} \\ \vec{x_z} & \text{if } ||F_n|| >= F_{max} \\ 0 & otherwise \end{cases} \tag{3.6}$$



(a) (b)

Figure 3.10: Slide primitive. (a)From the starting position with a hook preshape, the arm moves down until it touches the object, then it starts moving towards the target. (b)The object slides over the table from P_i to P_f with velocity V. The primitive keeps the applied force stable between the lower and upper thresholds.

Primitive parameters

- UPPER_FORCE_THRESHOLD (default: 9N): Upper force threshold to consider that the hand is exerting enough pressure on the object. If the value is higher the controller will loosen the pressure moving slightly away from the object.
- LOWER_FORCE_THRESHOLD (default: 5N): Lower force threshold to consider that the hand is exerting enough pressure on the object. If the value becomes smaller, the controller will increase the pressure moving towards the object.
- SURFACE_NORMAL (default: [0,0,1]): Normal of the supporting plane. This will determine the direction used to push the object against the surface.
- TARGET_VELOCITY (default: [0.005 m/s]): Linear velocity used to move the hand from the starting position to the target position.
- CONTACT_VELOCITY (default: [0.005 m/s]): Linear velocity used to push the object against the surface and to loose the contact in case the force applied becomes to high.

3.3.6 Unscrew primitive

To have a robot able operate autonomously in household scenarios, the ability to open containers is necessary. For example, in order to pour liquids from a container, the lid

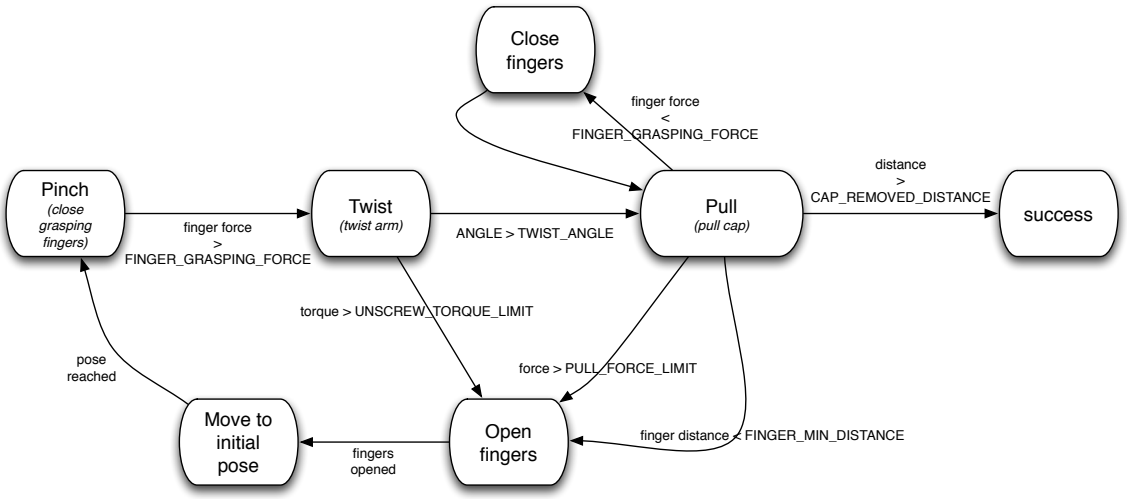


Figure 3.11: Unscrew primitive execution diagram. The execution begins with the pinch phase. This diagram shows the operation of the unscrew primitive, it does not depict a task formed of several basic primitives.

needs to be opened first. Although not all the containers have a screwed lid it is one of the most common cases.

The purpose of the unscrew primitive is to provide the robot the ability to open containers that have a screwed lid. Unlike the other primitives presented in this chapter, the unscrew primitive focuses on solving a specific manipulation interaction that would be very difficult to define as a task composed by other primitives.

This manipulation primitive assumes that the hand is over the cap ready to grasp it. Moreover, it uses several parameters to adapt its behaviour to the target object.

Primitive description

The unscrew primitive is divided into three phases: pinch, twist and pull. The phases and the transitions of the primitive are depicted in Fig.3.11. Note that the diagram shows the internal states of the unscrew primitive in order to explain its operation and does not depict a task formed of several basic primitives.

Pinch

Closes the selected fingers until the desired FINGER_GRASPING_FORCE is achieved on each fingertip. The force applied is determined using the tactile sensors on each finger.

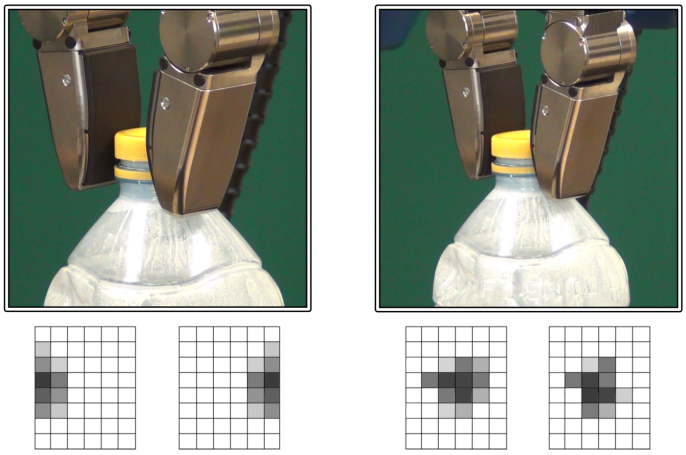


Figure 3.12: Cap grasping and tactile sensor readings. Left: Initial cap grasping, grasping center is not aligned with the cap. Right: Cap grasping after correcting the pose.

Twist

Twists the cap for the specified TWIST_ANGLE and transitions to the pull stage. The hand fingers are still controlled and in the case that the contact with the cap is lost, the fingers keep closing until the contact is detected again. This stage terminates prematurely if the torque detected is over the UNSCREW_TORQUE_LIMIT parameter or the fingers are closed without grasping the cap (inter-finger distance below FINGER_MIN_DISTANCE).

Pull

On this stage the arm pulls back to test if the cap is free. While pulling, the primitive is monitoring the tactile contacts and the force sensor. As in the previous stage, the fingers keep closing if the contact with the cap is lost. If the detected force is over the PULL_FORCE_LIMIT threshold or the fingers get closer than the FINGER_MIN_DISTANCE: the fingers open, the arm moves back to the initial position and the primitive starts again from the grasp stage. On the other hand, if the arm pulls the cap further than the CAP_REMOVED_DISTANCE the primitive terminates successfully.

Reactive adaptation

Due to the intrinsic uncertainty of real environments and systems, it is not possible to guarantee that the position of the fingers when pinching the cap is exactly aligned with the center of the cap. Moreover it is also not possible to produce approach vectors that are perfectly aligned with the cap normal vector (see Fig.3.12). Thus the cap will

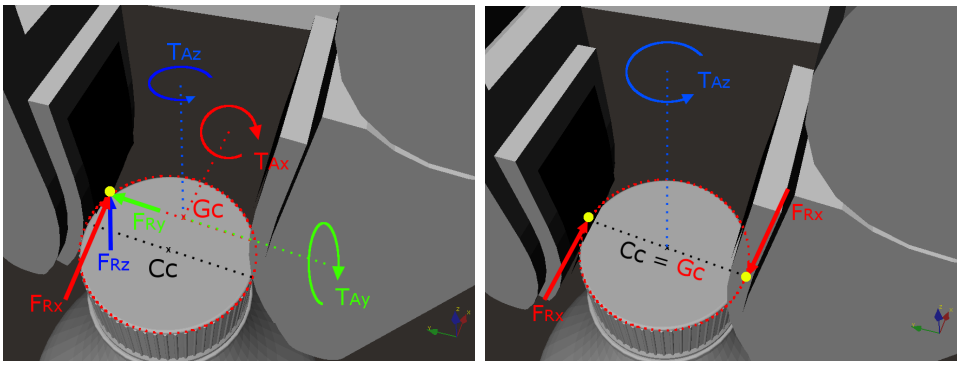


Figure 3.13: Left: Forces and torques produced when twisting the cap with an unaligned grasp point, F_{Ry} and F_{Rz} are undesired components that do not contribute to unscrew the cap but produce undesired torques. Right: Forces and torques produced with an aligned cap. C_C: Cap center. G_C: Grasp center. F_R: Reaction Force. T_A: Torque applied.

hardly be pinched by its center and the rotation axis of the hand will not be aligned with that of the cap.

The twist movement performed to unscrew the cap has the rotation center in the middle of the grasping fingers. It means that the fingers rotate around their middle point assuming that the cap axis is centred. If the cap axis does not match with the twist movement center, the unscrew movement will apply undesired forces on the object (see Fig.3.13). That fact can cause the unscrew motion to move the cap or the object away from the fingers. As the cap position is not tracked by the primitive, the object displacement can cause the subsequent grasps to fail. Thus, the cap misalignment can decrease the robustness of the primitive.

In order to solve this issue, a sensor based strategy is used. It dynamically aligns the fingers center with the cap center to improve the robustness of the primitive. During the grasp and unscrew phases, the contacts on the fingertips are recorded and averaged over time. Before moving the arm back to the starting position, if the recorded contacts are not aligned with the grasping fingers center frame, the distance to that frame is used to correct the position, see Fig.3.12. As for the robust grasp primitive, we have conducted experiments to evaluate the improvement of the reactive approaches. The experiments and results are discussed in the next section of this chapter.

Primitive parameters

- UNSCREW_TORQUE_LIMIT: Default 10 N/m. Defines the torque limit that will be applied by the robot to unscrew the cap. If higher torque is detected, a collision is assumed and the unscrew movement stops and moves back to the starting position to grasp the cap again.

- PULL_FORCE_LIMIT: Default 7N. This parameter defines the force limit when pulling from the cap of the object. If the robot applies more than the defined force when trying to remove the cap, we assume that it is not completely unscrewed, hence it stops pulling and moves back to the starting position to grasp the cap again and perform another unscrew movement.
- GRASPING_FINGERS: Default: 2. Defines the number of fingers to be used for pinching and unscrewing the cap. This parameter is used to automatically configure the preshape.
- FINGER_GRASPING_FORCE: Default: 5N. Defines the target force for each finger. Fingers will keep closing until target force is reached.
- TWIST_ANGLE: Default: 30 degrees. Defines the angle that the primitive will twist the cap on each twist stage.
- FINGER_MIN_DISTANCE: Default: 12mm. Minimum inter-finger distance that will allow an object to be in hand. If the distance is shorter, the controller assumes that the hand is closed but empty.
- CAP_REMOVED_DISTANCE: Default: 40mm. Distance that the cap has to be separated from the container to consider it free to move away.

3.4 Experiments

Three experiments have been implemented to validate and illustrate the usefulness of the manipulation primitives paradigm: validation of the robust grasp primitive, validation of the unscrew primitive and completion of a manipulation task using a set of the primitives described. The two first experiments are focused in illustrating the design of a manipulation primitive and the importance of reactive strategies. The last case is focused on showing how primitives can be combined to solve more complex manipulation tasks.

All the experiments have been tested on real robot systems. In all of them the main experimental platform is *Tombatossals*, an anthropomorphic torso with 29 DOF. The platform is composed of two 7 DOF Mitsubishi PA10 arms. The right arm has a 4 DOF Barrett Hand and the left arm a 7DOF Schunk SDH2. Both hands are endowed with Weiss Robotics tactile sensors on the fingertips. Each arm also has a JR3 force-torque sensor mounted on the wrist. The visual system is composed of a TO40 4 DOF pan-tilt-verge head unit with two Imaging Source DFK 31BF03-Z2 cameras and a Microsoft Kinect. Further information about this robotic platform is given in Appendix A.1.

3.4.1 Validation of robust grasp primitive

As explained in Chapter 2, humans perform reactive movements to adapt to unexpected sensor input. Intuitively, reactive movements should increase the grasp success, especially in real and uncertain environments. This experiment is carried out to validate the intuition and quantitatively provide information about the importance of such a reactive strategy when grasping objects.

Experimental setup

In order to compare the robust grasp primitive with a non-adaptive grasp controller, we have designed a naive grasping primitive. The experimental procedure, for each primitive, consisted on grasping 10 different household objects (see Figure 3.14) 20 times. 10 using the approach vector given by the visual system and 10 introducing a random uniform error to the approach vector generated by the visual system. Thus, each of the evaluated primitives has tried 200 grasps.

A trial is considered successful if the grasp obtained after the execution is considered stable by a human observer. If the object, when lifted, does not move in the hand, the grasp is considered stable by the observer.

Naive grasping primitive

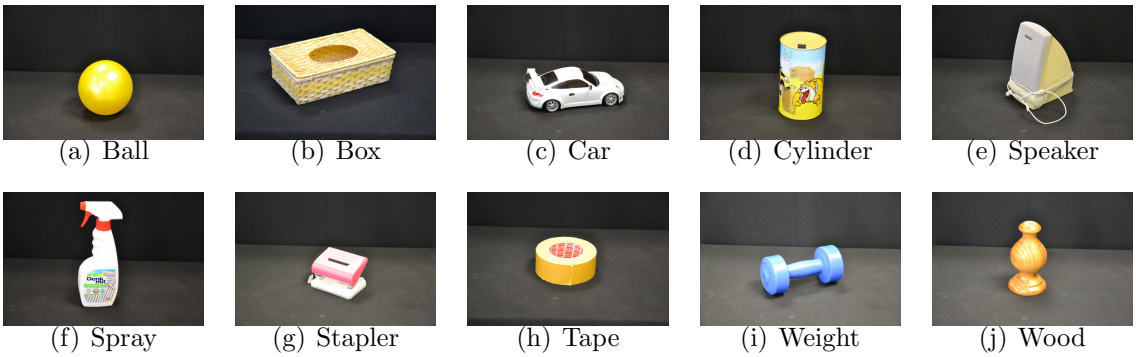
We have designed a grasping strategy that does not use tactile and force sensors to perform corrections. The naive grasping primitive behaves as follows: The arm moves forward 10cm or until a contact is detected. Then the hand closes stopping each finger that detects contact. Finally force is slightly increased on the distal phalanxes to establish the final grasp.

Environment

The experimental environment consists of the robot in front of a table. On the table there is a single test object in any position that can be reached using a top grasp by the left hand.

Test objects

In order to represent the different objects in household scenarios, an heterogeneous set of objects has been selected. The object test set is composed of ten different household objects (see Figure 3.14). There are objects with primitive shapes (cylinder, ball) and symmetries (tape, wood) but there are also more complex objects (spray, stapler). The objects are completely unknown to the robot, the primitives have no prior information about them.



(a) Ball (b) Box (c) Car (d) Cylinder (e) Speaker

(f) Spray (g) Stapler (h) Tape (i) Weight (j) Wood

Figure 3.14: Set of 10 objects used for testing the robust grasp primitive.

Assumptions

The objects have to be placed on the table inside the workspace of the arm used to perform the experiments. Moreover, the objects used, have to have lambertian properties in order to be visible by the visual system used. No other assumptions about shape, weight or other physical properties are made.

Parameter settings

There are no specific parameter settings for the primitives used in this experiments. Both the naive and robust grasp primitives are configured with their default values as presented in Section 3.3.1.

Object detection

In order to grasp the objects, the robot needs to determine their pose and plan a trajectory. For this experiment we have implemented a simple approach that is well known, fast, stable but not very precise. Using this object pose estimation, will allow the controllers to show its performance under some uncertainty. The methods used in this experiment to detect objects are detailed in Chapter 6.

To determine the object position, Kinect RGBD images in combination with algorithms from the Point Cloud library [Rusu and Cousins, 2011] have been used: The table plane is segmented out and the remaining points are clustered, the target object position is determined by the centroid of the leftmost cluster. A detailed description of the visual pipeline is provided in Section 6.3.3.

Approach vector calculation

As stated in Section 3.2, any primitive requires at least an approach vector as an input parameter. Usually, the approach vector used as the starting point for a grasping procedure, is determined by a grasp planner. In this experiment, we have used a grasp planner based on the object point cloud eigen vectors to calculate the approach vector.

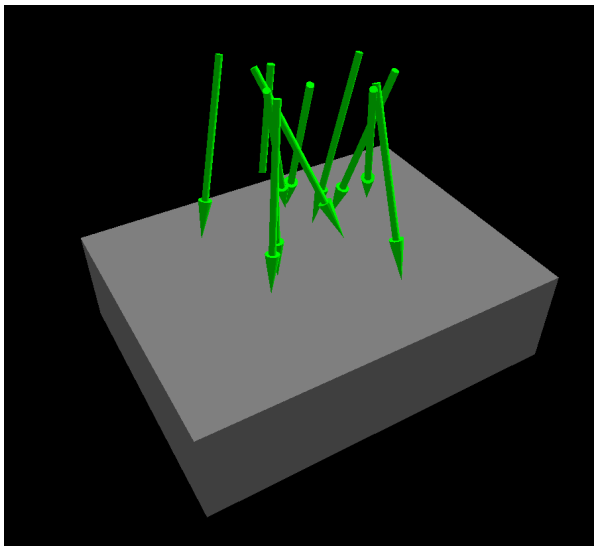


Figure 3.15: Set of 10 approach vectors after adding the random uniform error.

The input of the grasp planner is the segmented point cloud containing the object to grasp. The approach vector is calculated using the *simple grasp generator* described in Sec. 6.3.5 and the result is filtered by orientation to keep only vectors that approach from the top.

In order to test the controller under rough error conditions, we have performed tests adding a random uniform error to the approach vector calculated by the system.

The error that we have introduced to the approach vectors follows an uniform distribution, $\mathcal{U}(0,5)$cm in each axis and $\mathcal{U}(0,15)$degrees on each axis. Figure 3.15 shows a set of 10 approach vectors for an object after adding the uniform error.

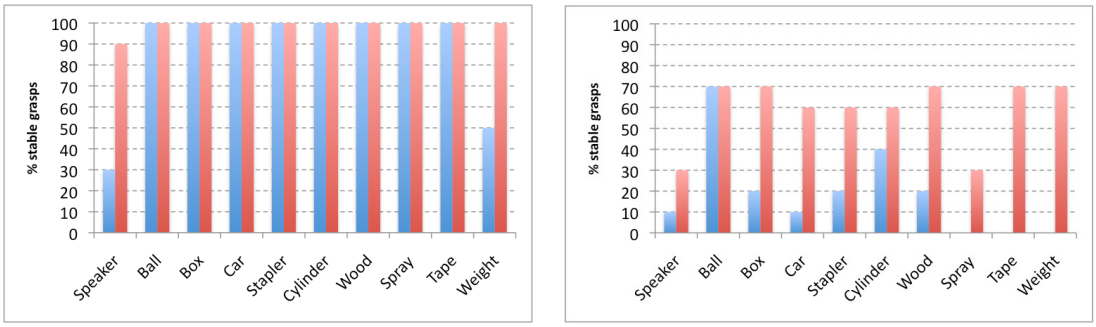
Results and discussion

Table 3.3 shows the overall performance of each controller with and without the additional error. The performance of the robust controller has shown to be better especially under error conditions. Although the results without additional error are quite good for both controllers, the robust grasp primitive outperforms the naive one by more than 10% of success rate.

Figure 3.16 shows the performance of each controller, with and without error, for each object of the test set. Under controlled conditions both controllers are able to perform 10 successful grasps out of 10 attempts on eight of the objects. On the other hand, for some objects, e.g. speaker and weight, the performance of the naive controller is dramatically reduced (5/20) while the robust controller keeps its good performance almost intact (19/20). The main reason of that low performance is that those objects

	stable grasps	failures	%success
naive	88	12	88%
robust	99	1	99%
naive + error	19	81	19%
robust + error	59	41	59%

Table 3.3: Grasping experiments overall results after 100 attempts.



(a) Grasping performance without error.

(b) Grasping performance with error.

Figure 3.16: Grasping performance after 10 trials per object. Blue: Naive controller, Red: Robust controller

have some properties, asymmetry and thinness, that make them difficult to grasp. On the other hand the robust grasp primitive is able to adapt to those properties that cause the naive controller to fail.

Under uncertainty conditions, the robust primitive is able to perform better than the naive one. As shown in Table 3.3 under error conditions the performance of the naive controller drops to 19% while the robust primitive holds a 59% of robust grasps achieved.

3.4.2 Validation of unscrew primitive

The purpose of this experiment is to validate the implementation of the unscrew primitive, testing it on different objects. Furthermore, this experiment also validates that the sensor-based reactive behaviour improves the performance of the primitive by comparing the results of tests with and without corrective movements.

Experimental setup

To validate the unscrew primitive and the impact of the reactive adaptation on its performance, an object test bench composed of eight objects has been set up and the task has been executed 10 times for each object with and without the reactive adaptation enabled. Overall the unscrew task has been executed 160 times, 80 with tactile corrections and 80 without them.

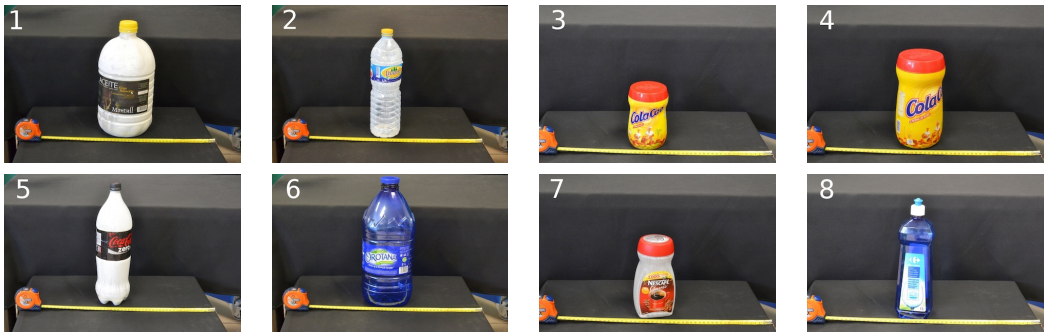


Figure 3.17: Object test bench for the cap unscrew experiments. Object id on the top right of each picture. The lenght of the measuring tape in the pictures is 40cm.

Environment

The experimental environment consists of the robot in front of a table. On the table there is a single test object in any position that can be reached by the right hand. The object is standing upright.

Test objects

The object test set is composed of eight different household objects (see Fig. 3.17). In order to represent the different types of bottles and containers, an heterogeneous set of objects with different shapes, textures, sizes, weight and with different caps was chosen. Moreover, it is important to note that the cap thread is also different from one object to another, hence they require a different number of turns to fully detach the cap from the container.

Assumptions

Although the visual system is able to deal with more than one object, to simplify the experimental process, only one object is standing in front of the robot. The objects are unknown to the robot, the only assumption is that each object has a cap that can be unscrewed and they are laid out upright on the table and inside the workspace of the left arm. Objects' surface has to have lambertian reflectance properties in order for the visual system to detect them. No other assumptions about the objects are made.

Task definition and metrics

The same visual system and pipeline used for the robust grasp validation is used to detect the objects (see Section 6.3.3.). Moreover, as the objects are standing on the table, it needs to be grasped with one hand to fix it before the other hand can perform the unscrew attempt. The grasping procedure is the same as specified in Sec. 3.4.1 but using a side approach vector instead of a top one. Once the object is grasped it is moved to a manually determined vantage position. Then the cap detection starts. It consists

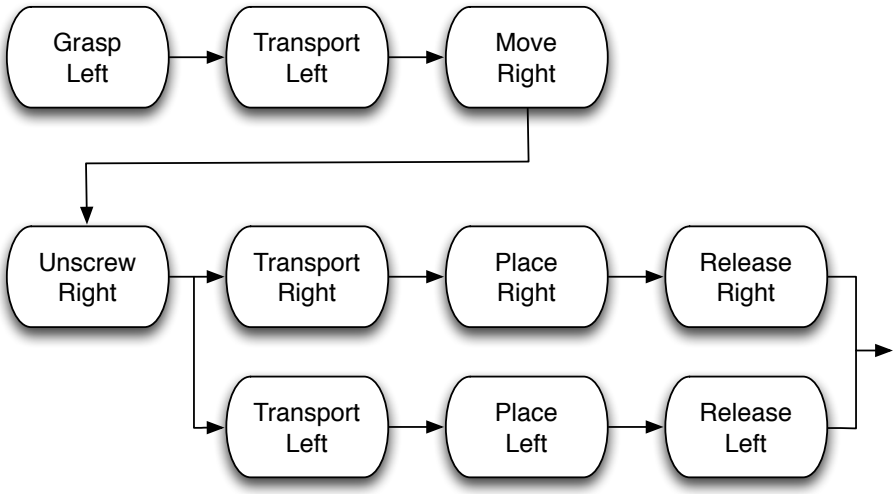


Figure 3.18: Unscrew task definition using the manipulation primitives framework. All the depicted transitions are triggered by a success event. For the sake of clarity, the error transitions are not depicted in the figure; all the primitives have a transition to an error state.

basically on determining the centroid of the upper part of the object which is assumed to be the cap. That centroid is used to generate an approach vector from the top that will be used by the unscrew primitive as the starting position. The whole grasp and unscrew task is described using the manipulation primitives framework as depicted in Fig 3.18.

For each task execution, the following values were measured:

- Time: The time it takes to complete the unscrew primitive execution.
- Twist moves: The number of twist movements required to open the container.
- Cap unscrewed: It will be considered that the robot has succeeded unscrewing the cap if at the end of the task execution the cap is separated from the recipient or can be separated from the object without turning it.
- Success: It will be considered that the whole task has succeeded if the robot removes the cap from the object and places it on the table by itself. This does not include situations where the cap falls while lifting it or as the result of a twist movement.

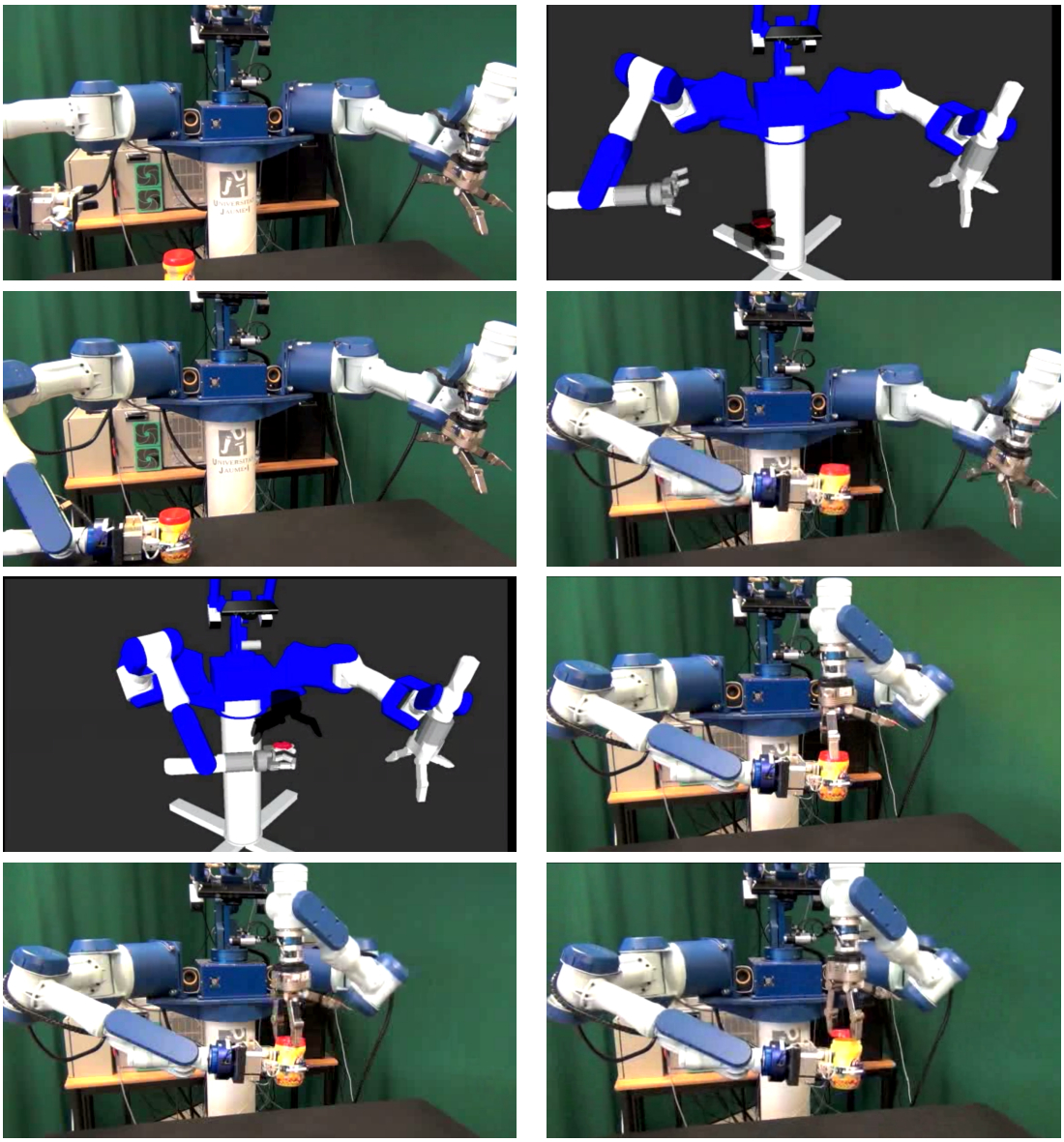


Figure 3.19: Unscrew task execution sequence.

Parameter settings

Like for the other primitives, it is recommended to set the parameters of the primitive depending on the object and the environment, nevertheless the default parameters have been used over the object test set with good results. Adjusting the parameters to each object may improve the performance of the primitive.

Force-torque limit parameters were empirically obtained for our robotic platform. The sensor readings were monitored during several executions and the parameters were set manually.

Grasp preshape and grasping fingers were determined by the task and robot embodiment. These parameters are designed for future improvements, for example to allow selecting the grasp preshape and the grasping fingers according to the cap geometry.

Distance parameters were selected regarding the smallest cap diameter possible (FINGER_MIN_DISTANCE) and the maximum cap height (CAP_REMOVED_DISTANCE).

Results and discussion

The overall results are shown in Table 3.4. After 80 task executions for each primitive configuration (with and without corrections enabled), the success rate of both primitives is very high, almost 90%. Although the primitive with the reactive adaptation enabled has slightly better success rate, it cannot be considered meaningful from this results. The main reason seems to be the low error and noise in the estimation of the bottle cap center in our setup. To test other scenarios with less precision and more noise, 15 more grasps were performed on Object 2 adding some uniform random noise ($\pm 1.5cm$) to the approach vectors calculated to grasp the cap. Object 2 was selected for this extra tests because, as can be seen in Fig. 3.20, it is has been the most problematic object for the strategy. The results obtained after 15 task executions with both primitive configurations is shown in Table 3.5. In this case, when the approach vectors are less precise, is where the robustness of the reactive strategy arises.

Fig. 3.20 depicts the success rate of the whole task for each object in the test bench. Meanwhile Fig. 3.21 shows the cap unscrewed rate. In some cases the cap unscrewed rate is higher than the task success rate, this is because although the cap was unscrewed, the robot failed to grasp it correctly and move it away from the object. Regarding the elapsed time required to perform the task, Fig. 3.22 shows that, for almost all the objects, the reactive strategy requires a bit more time. As expected, performing corrections takes more time than moving back always to the same position.

On the other hand the same twist movements by both primitives could be expected because both of them turn the cap the same TWIST_ANGLE degrees, but as shown in Tables 3.4 and 3.5 the reactive primitive requires more twist movements to open the object.

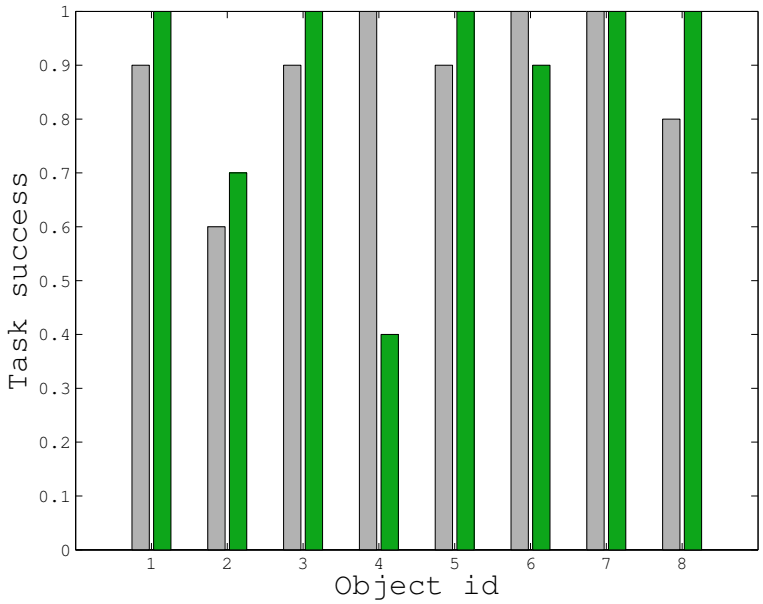


Figure 3.20: Grey: Task success rate for each test object after performing the unscrew task 10 times per object with the reactive adaptation enabled. Green: Reactive adaptation disabled.

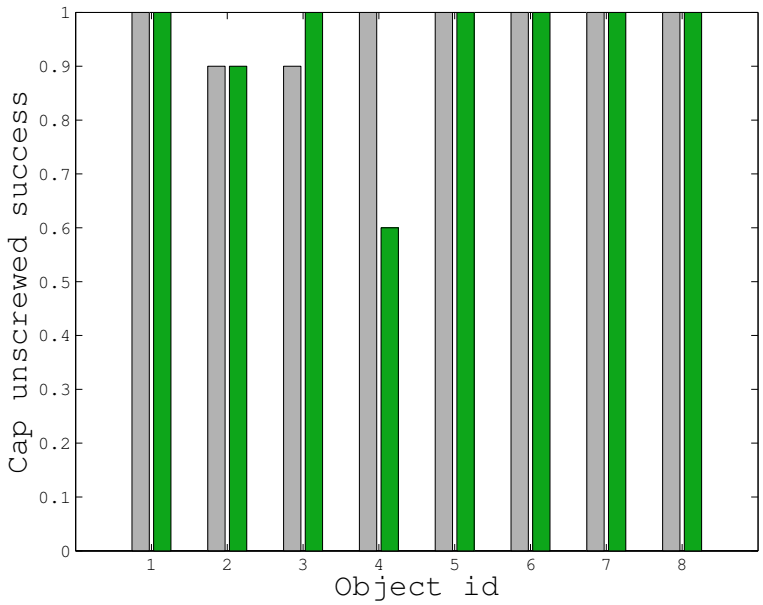


Figure 3.21: Grey: Cap unscrewed rate for each test object after performing the unscrew task 10 times per object with the reactive adaptation enabled. Green: Reactive adaptation disabled.

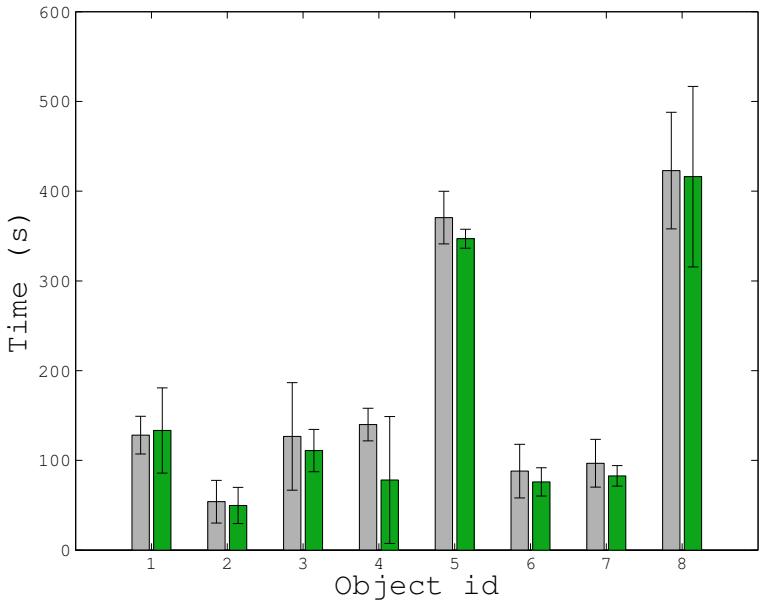


Figure 3.22: Grey: Unscrew task elapsed time for each test object after performing the unscrew task 10 times per object with reactive adaptation enabled. Green: Reactive adaptation disabled.

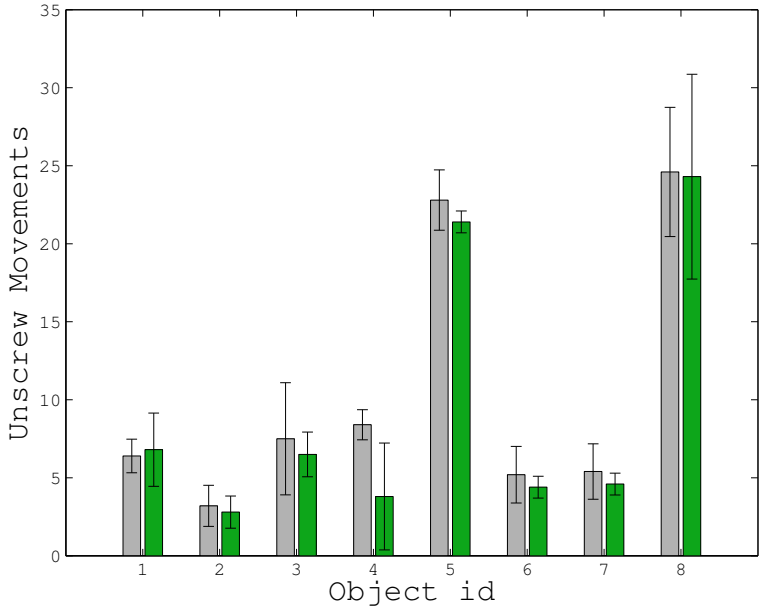


Figure 3.23: Grey: Twist movements performed for each test object after performing the unscrew task 10 times per object with reactive adaptation enabled. Green: Reactive adaptation disabled.

Reactive	Success rate	Cap unscrewed rate	Time	Moves
No	70/80 (87.5%)	75/80 (93.8%)	156.47	8.98
Yes	71/80 (88.6%)	78/80 (97.5%)	172.42	10.06

Table 3.4: Averaged results after 80 executions of each primitive

Reactive	Success rate	Cap unscrewed rate	Time	Moves
No	9/15 (60.0%)	10/15 (66.7%)	57.5	3.4
Yes	12/15 (80.0%)	13/15 (86.7%)	66.9	3.77

Table 3.5: Averaged results after 15 executions of each primitive over Object 2 with extra noise

Thus the difference in the elapsed time is caused by the highest amount of unscrew movements performed by the reactive primitive. The time taken (Fig. 3.22) and the number of movements performed (Fig. 3.23) are highly correlated. This would be expected if each unscrew movement took the same time, but as depicted in Fig. 3.11 the twist movement can be finished by other conditions, however the high correlation of time and twist movements suggests that the twist movement always ends with the TWIST_ANGLE condition instead of UNSCREW_TORQUE_LIMIT or FINGER_MIN_DISTANCE.

3.4.3 Emptying a box: Execution of a complex task

The purpose of this experiment is to demonstrate that the manipulation primitives paradigm is valid for describing and executing complex sensor-based tasks. To do so, we choose the task of emptying a box with no previous information about the number, identity, location and pose of the objects inside.

The task, depicted in Figure 3.24, is described using the manipulation primitives presented in Section 3.2 and a task specific perceptual primitive. The primitives are laid out in a loop that consists of pick and place sub-tasks that are repeated until there are no objects left in the box.

Three different methods were implemented for the approach vector generation and their impact in the task performance was measured.

Experimental setup

In order to validate the task implementation we carried out a total of 30 experiments of emptying a box filled with five unknown objects (see Fig. 3.26(b)). As can be seen in Figure 3.24, a key part of the task execution is the generation of the initial approach vectors. Three strategies were implemented: *random blind*, *blind exploration* and a *vision-based* method. 10 experiments were performed for each approach vector generation method.

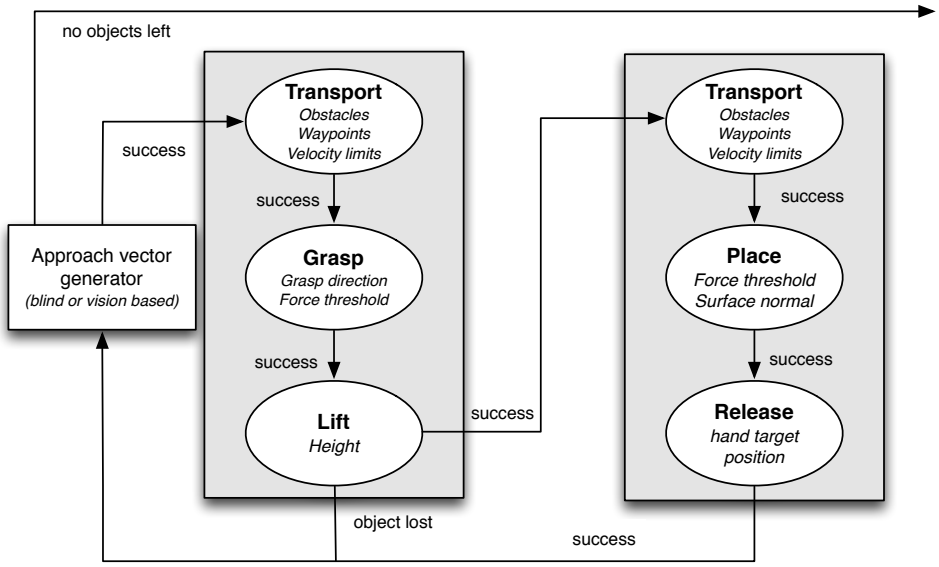


Figure 3.24: Task definition for a pick and place task. Primitives are represented by circles. Perceptual primitives are depicted using white boxes. Grey boxes group primitives into sub-tasks. Inside each primitive, some examples of parameters are written in italics.

To compare the performance of using different approach vector generators, different metrics were used:

- Task success: If all the objects from the box are removed, the task execution is considered successful.
- Grasp attempts: Every time the robot tries to grasp an object from the box, the grasp might not be successful and the robot can fail to grasp the object. In that case several attempts may be needed to grasp an object.
- Time: Time taken to empty the box.

Environment

The experimental environment consists of the robot in front of a table. On the table there is a box full of objects that can be in any position, even stacked (see Fig. 3.26(b)).

Test objects

There are no assumptions about the shape, material or texture of the objects. However, the objects play an important role on the performance of the grasping strategies, hence it is important to have the same objects for all the tests even if they are not exactly in

the same position for all the tests. For the experiment, we have selected five household objects with different primitive and compound shapes, see Fig. 3.26(b).

Assumptions

Objects' pose and geometry are unknown. The pose and size of the box that contains the objects are known. The object positions inside the box are not restricted, objects can be in any position and orientation inside the box, except that there is some clearance between the objects and the sides of the box. The object size limits are defined by the robot hand dimensions so that the objects fit inside the hand and are thus graspable. The box is set on an even plane inside the arm workspace.

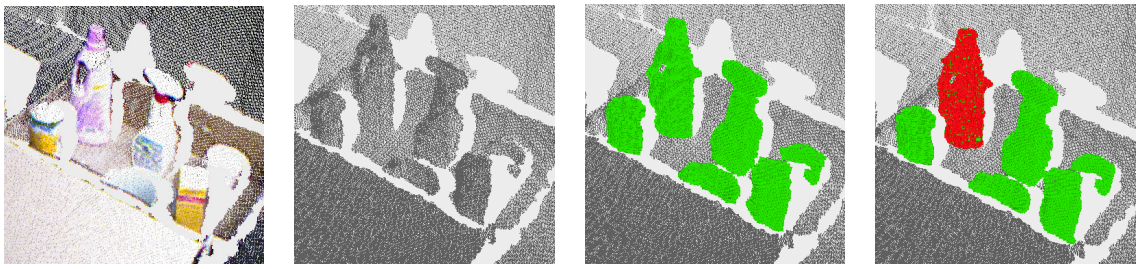
No further assumptions on the objects are made, except for the visual based approach vector generator. It requires the material of the objects to be visible by the active RGBD camera.

Parameter settings

The required parameters are: the starting approach vector to a target object and the target position to place it. The approach vector for the grasp primitive is generated by the approach vector generator, on the other hand the target position to perform the place sub-task is manually set to a pose outside the box. Due to the reactive implementation of the place primitive, objects can be placed on top of each other, the primitive will detect the contact and place them smoothly.

Approach vector generation

- *Random blind* strategy: top-grasp approach vectors are generated uniformly at random inside the known location of the box. In this case, there are no means to determine the number of objects left in the box, instead of using a large timeout, to save time, the end of the whole process is determined by a human supervisor.
- *Blind exploration* strategy: the arm moves down until a contact is detected. If the contact is an object, the approach vector is generated over that contact location. If the contact is the box bottom, the hand starts moving along the box until it detects a contact using the tactile and the force-torque sensor. As the position of the box is known, proprioception is used to determine whether the contact is with an object or with the box bottom. The exploration trajectory followed by the hand is shown in Fig. 3.26(a). The task ends after completing an exploration trajectory without finding an object.
- *Vision-based* strategy: the Kinect sensor is used in the same fashion as in Sec 3.4.1 and 3.4.2. Objects are segmented from the environment by a pass-trough filter using the known box boundaries and clustered as shown in Fig. 3.25. The approach vector is determined to approach the centroid of a randomly chosen cluster from the top. The task ends when there are no clusters left.



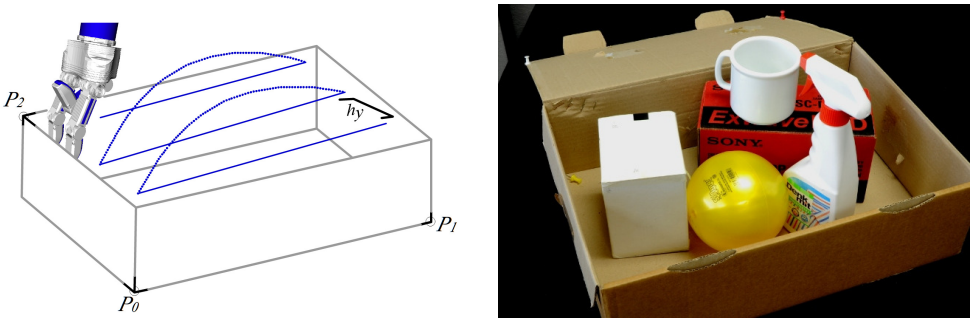
(a) Original 3D image.

(b) Original 3D point cloud read from Kinect sensor.

(c) Virtual box background filtering. Background points are colored in gray and objects are in green.

(d) Object clustering and selection. Background points are marked in gray, objects in green, and the selected cluster is labeled in red

Figure 3.25: 3D point cloud segmentation phases for the visual-based approach vector generation.



(a) Hand preshape for exploration and exploration trajectory.

(b) A possible object layout

Figure 3.26: Exploration trajectory and object layout.

Method	Attempts	Time	Task success
Random	30.5	284.1 s	100%
Exploration	32.8	334.8 s	100%
Vision	7.1	101.4 s	100%

Table 3.6: Average results for each method after 10 task executions.

Results and discussion

Table 3.6 shows the averaged results for each approach vector generation method. All the methods were able to empty the box successfully 10 times out of 10. Regarding the number of attempts and time consumed, the vision based method outperforms the other two methods. However, the interesting result is that the blind methods were also able to complete the task successfully every time. In addition, blind methods are more general as they do not require any assumption about object material reflectivity properties.

Surprisingly, the exploration method requires almost the same attempts than the random method, this result is related with the density of objects. The impact of object density is depicted in Fig. 3.27 where the average number of required attempts to grasp one object, depending on the number of objects remaining in the box is shown. When there is only one object left (low object density) the random method requires way more attempts. As expected, the exploration method takes more time (for a similar number of attempts) than the random method because it has to perform the exploration trajectory for every attempt.

It is important to note that the reactive grasping primitive plays a crucial role because the generated approach vectors are quite inaccurate, especially in the blind approaches.

3.5 Conclusion

This chapter presented the manipulation primitives framework. Starting from the idea of action-phase controllers, provided from neuroscience studies, we have implemented several basic action-phase controllers to form a vocabulary of basic actions that has been used to define more complex tasks.

As the studies detailed in Chapter 2, corrective actions are also present in the human grasping process, following this idea, we have extended the manipulation primitives paradigm with reactive capabilities, implemented them and validated their importance.

From a practical point of view, the main contributions are threefold. 1) A robust reactive grasp primitive was presented. Experiments verified that the reactive control was able to recover successfully from significant planning errors. 2) An unscrew primitive was also implemented and verified as well supporting the reactive control approach already shown with the grasp primitive. 3) It was shown that the combination of sev-

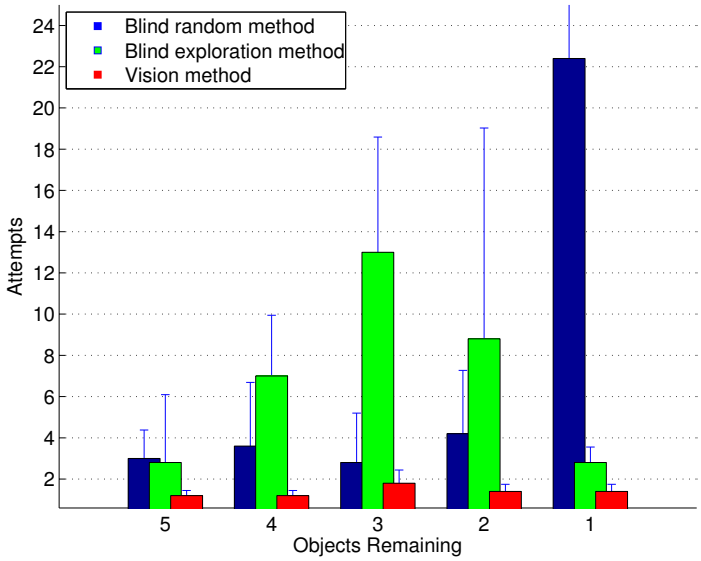


Figure 3.27: Average and standard deviation of required attempts depending on the number of objects remaining. The standard deviation for the blind random method when there is only one object left is truncated in the picture, its value is 39.32

eral manipulation primitives could be used successfully to complete a complex task, emptying a box of unknown objects. The experimental results showed, not surprisingly, that increased perceptual capabilities improve the performance. However, a more interesting finding is that even under the worst conditions, in the blind grasping approach with only tactile feedback, the combination of reactive primitives was usually able to complete the task successfully, even though the time required was increased.

The results support the paradigm based on reactive manipulation primitives as a good way to generate and execute plans in unstructured and uncertain scenarios. The manipulation primitive approach is also suitable as an abstraction layer to provide a way to share plans, and more generally, knowledge, between different embodiments. The results encourage us to believe that manipulation problems can be solved in complex, unstructured scenarios while retaining hardware independence on a higher level. However, immediate feedback capabilities seem essential in coping with the complexity of the world.

Many interesting open issues remain for the future. Firstly, the embodiment specific primitive controllers currently require careful design for each embodiment. Procedures which could automatically at least bootstrap the building of the controllers, or even construct the controllers, would be very valuable. It seems that the use of machine learning techniques would be an interesting and possibly profitable avenue of research in this direction. This approach would most likely require high quality simulations of the

embodiment in order to provide training data for the learning approaches. In Chapter 5 the available simulation tools and the implementation of dynamic simulation for robotic manipulation is presented and discussed.

Secondly, unstructuredness and uncertainty can appear at different levels and in different aspects. The primitives presented here are mostly related with tolerating uncertainty in object pose and shape. In order to design primitives for other types of uncertainties and unexpected events, other primitive designs would be necessary, and most importantly a scheme to coordinate and group different strategies, for example hierarchically, would be necessary.

Over the years, the approaches of robot grasping have split into two groups of approaches. On one hand, object and planning based robot grasping focuses on considering a grasp as a set of contact locations on the object shape, through which manipulation forces are exerted on the object. On the other hand, hand and control based approaches rely on the capabilities and constraints of the robot embodiment, focusing on control aspects. The proposed manipulation primitives paradigm belongs to the latter approach, considering grasps as starting conditions for the action and letting the control loop and the real world itself guide the execution. It is the author firm belief that the inclusion of reactive capabilities is essential in coping with the whole scope of complexity present in the real world.

The research and experiments presented in this chapter have been previously published. The design of the robust grasp primitive was published in [Felip and Morales, 2009] and it was improved and presented together with the manipulation primitives paradigm in [Felip et al., 2012]. The empty the box experiments were presented in [Felip et al., 2013]. Finally the unscrew primitive and the related experiments were published in [Felip and Morales, 2014].

Chapter 4

Contact event perception

Neuroscience studies presented in Chapter 2, point out that human grasping is driven by the creation and breaking of contacts with the environment. Perception of contact events is performed by humans using multiple sensor cues: visual, tactile, proprioceptive and audio to name a few. Moreover, the prediction of contacts is also an important source of information for error detection and recovery while performing manipulation tasks.

Regarding robots, perception of physical interaction can also be achieved using many sensor modalities: tactile, force-torque, proprioception, accelerometers; even sonar, laser and vision could be used as well. Is a problem in robotics to fuse all the available data to provide the robot with enhanced perception capabilities. Sensor fusion has not been applied so far on contact localization for manipulation.

In this chapter, a sensor fusion framework for contact detection and localization is proposed. It is able to use any knowledge available to detect and localize contacts and improve the robustness and precision of the detected contacts. The presented approach allows the integration of multiple sensors, environment, context and predictions. On top of it, we have implemented a contact event detector that will provide the necessary contact events required by the task description to trigger transitions between states, as shown in Chapter 3.

The framework is divided in three main parts: contact hypotheses generation, hypotheses fusion and contact condensation. Firstly, the contact space and the other basic concepts for the contact hypothesis framework are presented. Secondly, the guidelines for the integration of sensors into the contact space, together with some examples of implementation are shown. Thirdly, the fusion algorithm to combine the different sensory cues and how to perform contact detection and localization from the fusion result is shown. Finally, the approach is validated through several experiments on Tombatossals and a simple use case of a contact driven controller on ARMAR-IIIb. For a detailed description of the robotic platforms used see Appendix A.1 and A.2.

4.1 Related work

In the last decade, contact sensing has become a key element on all the robots with manipulation capabilities. Besides tactile sensors, there are other devices that can measure physical interaction, such as force-torque sensors or joint-torque sensors. Moreover, there are other sensor modalities that can be used like vision or audio [Lacheze et al., 2009]. Unfortunately, each data source has its own representation and the information from different sensors cannot be easily compared with that from other sources. In addition, information about the environment can also be very useful to constrain where physical interaction can occur.

Data fusion from different sources has been a largely studied problem in robotics. In the early 90's the theoretical basis of the current techniques were already settled [Hackett and Shah, 1990]. More recently, the evolution of parallel computation enabled the use of high computational cost probabilistic approaches (e.g. particle filters) [Thrun et al., 2005]. On real scenarios fusion is often performed with a defined goal: fusion of audio and visual input to track a talking person [Fermin et al., 2000], to track an object [Meeussen et al., 2006] or to recognize it [Lacheze et al., 2009]. [Prats et al., 2013] developed a framework that presents sensor fusion for robotic manipulation, where each sensor handles a controller that contributes to the resultant control applied to the robot. In this chapter, instead of focusing on control, we provide a common representation for contact detection and localization.

Using vision and force, [Ishikawa et al., 1996] proposed a method to detect contacts between a known grasped object and the environment. In that work the fusion method is task specific and the contact detection method is embodiment specific. Other works that perform contact localization, either use only one sensor modality [Meier et al., 2011] or process and fuse the data with an ad-hoc non scalable method. [Hebert et al., 2011] presented a probabilistic sensor fusion method to estimate the pose of a grasped object, although they obtained a very good precision (5mm) contacts were considered only on the fingertips.

4.2 Contact hypothesis framework

The sensor fusion framework is composed of two independent parts; the contact hypotheses generators and the integrator (Fig. 4.1). Each generator creates contact hypotheses based on a defined criteria (e.g. force sensor, simulator) and sends them to the integrator. The integrator receives those hypotheses, combines them and uses the result to determine the likelihood of a contact at a location.

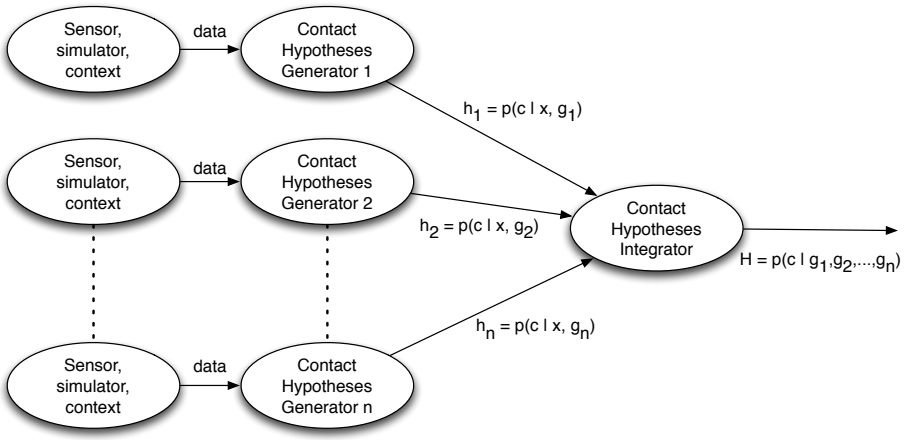


Figure 4.1: System overview. Contact hypotheses generators obtain data from sensors, simulation, kinecmatics, control and environment to produce contact hypotheses that are sent to the integrator. The integrator performs the fusion and contact detection and outputs the resulting contacts.

Contact hypothesis Represents the likelihood that a contact happened at a specified location.

Contact hypotheses integrator Updates the hypotheses space fusing the input clouds of hypotheses from the contact hypotheses generators and provides as a result the estimated contacts.

Contact hypotheses generators Use an information source (context, simulator, sensors) to produce a cloud of contact hypotheses.

Hypothesis space Is a 3D Cartesian space discretized in voxels of a fixed size. The state of the HS is determined by the occupied voxels and the likelihood of each one.

4.2.1 Contact hypothesis and hypothesis space

A contact hypothesis represents the likelihood that a contact happened at a specified location. The hypothesis space HS is a 3D Cartesian space discretized in voxels of a fixed size. The state of the HS is determined by the occupied voxels and the likelihood of each one. The state of the HS is updated by the integrator.

The integrator receives sets of contact hypotheses h_n from the generators $(g_1, ..., g_n)$ that represent the probability p that a contact c happened at a specified voxel $x \in \mathbb{R}^3$. Thus, the set of hypotheses generated by g_n is $h_n(x) = p(c|x, g_n)$ (see Fig. 4.1).

After performing the hypotheses fusion, the output of the integrator is a set of contact hypotheses H representing the probability that a contact happened at a specified voxel combining all the received inputs $H(x) = p(c|x, g_1, ..., g_n)$.

A contact hypothesis must have information about its location and likelihood. Beyond the required information, in the proposed framework, a contact hypothesis is composed by the following elements (required fields are **bold**):

- **Location**: Specifies the 3D position of the hypothesis.
- **Likelihood**: List of likelihoods of each source that contributed to this hypothesis.
- Timestamp: List of timestamps when each likelihood was generated.
- Force magnitude.
- Force direction.
- **Type**: Can take one of this values: Regular, Support or Null.
- **Source id**: List of sources that contributed to this hypothesis.

Contact hypotheses types

It is possible to implement generators that are not based on physical evidence of contacts (e.g. predictions). To avoid detecting contacts without having physical evidence, contact hypotheses are labelled with a type that will be used by the integrator to determine how to proceed for the hypotheses fusion. The possible types are: regular, support and null.

Regular hypotheses

Regular hypotheses are those produced by real sensors from perceptual evidence.

Support hypotheses

Support hypotheses are those produced by generators that do not have perceptual evidence of a contact. Support hypotheses are used to add contextual data or predictions to the estimation of the contact locations. Therefore if the sensors detect a real contact and generate hypotheses, those hypotheses that fuse with support hypotheses will increase their likelihood. On the other hand, support hypotheses that are not fused with any other hypothesis from perceptual evidence will be discarded.

Null hypotheses

Null hypotheses represent the locations of the contact space where there are no contacts. There are cases, where there is relevant information about where contacts cannot happen, this data can be reflected in the contact detection framework generating null hypotheses on the locations where contacts cannot happen. Null hypotheses are used

to draw a null space for the contact detection. Any hypothesis that fuses with a null hypothesis will be discarded.

4.3 Hypotheses generators

The role of a contact hypotheses generator is to convert any suitable information available to the robotic system (context, simulator, sensors) in a cloud of contact hypotheses that will be added by the integrator to the contact space. In Section 4.4 the implementation details of the contact hypotheses generators used in our system are provided. This section gives the guidelines for the implementation of contact hypotheses generators based on sensors and other information sources.

A contact hypotheses generator uses the data from a sensor, a controller, a simulator or any other source that provides information about a perceived or a predicted contact located in the space. The output is a cloud of contact hypotheses (that determine the possible contact locations), the likelihood of each generated contact hypothesis and its type.

4.3.1 Implementation guidelines

For the implementation of a new generator, the first step is to determine whether the generator can detect one or more contacts at a time. On the one hand, single-contact generators can only detect one contact at a time (e.g. bumpers or force sensors). On the other hand, multi-contact generators are able to detect multiple contacts simultaneously (e.g. tactile sensor arrays, simulation engines).

For single-contact hypothesis generators, the probability of the detected contact has to be distributed among the generated contact hypotheses, see Eq.(4.1). The likelihood can be distributed uniformly or with any other distribution, depending on the nature of the data used. For multi-contact hypothesis generators, we will allow the sum of likelihoods to be equal or greater than one, see Eq.(4.2), because more than one contact can be detected at a time.

$$\sum^{i} p(c|x_i, g_{single}) = 1 \tag{4.1}$$

$$\sum^{i} p(c|x_i, g_{multi}) \geq 1 \tag{4.2}$$

The second step is to calculate the likelihood of each generated contact hypotheses. To do so, the nature of the data that the generator uses has to be identified. Although there may be more types, in this thesis the following types were identified:

- Binary: (contact / no contact) it gives no clue about the contact distribution, in that case a fixed value for all the hypotheses is used. The constant value can be determined by the sensor model or experimentally probing the sensor and determining the likelihood of a contact to be detected. A bumper or a deterministic simulator are examples of binary data sources that can provide information about the existence of a contact but not a related value. See implementation examples in Sec. 4.4.3 and Sec. 4.4.5.

$$p(c|x, g_{binary}) = constant \tag{4.3}$$

- Value: If the value of the input data is directly related to the contact likelihood. The final probability for each hypothesis can be calculated using Eq.(4.4) where $argmax(data)$ is used for normalization and represents the maximum value of the current reading only if $argmax(data) > 0$. This approach assumes that the sensor does not provide false positive readings, otherwise if the sensor output is different from zero when there is no contact (e.g. due to noise or hysteresis effects) those values will be considered with high likelihood, in this cases the sensor input has to be previously filtered. An example of this kind of data are tactile or pressure sensors, in those cases the higher the value the more likely that there is a contact on the location where the sensor is at. An implementation example is provided in Sec. 4.4.2.

$$p(c|x, g_{value}) = \frac{data_x}{argmax(data)} \tag{4.4}$$

- Distance: If the data used to generate hypotheses is a distance (e.g. from the sensor to an object). An inverse square law like Eq.(4.5) can be used to determine the likelihood of each hypothesis. Where λ determines the distance at which the likelihood will be 0.5, this has to be tuned depending on the precision of the sensor and its calibration. An implementation example is provided in Sec. 4.4.4.

$$p(c|x, g_{distance}) = \frac{\lambda^2}{\lambda^2 + distance^2} \tag{4.5}$$

Finally, the last step is to determine where in the contact space the hypotheses will be generated. If the location of the hypotheses can not be determined by the data used to generate the hypotheses, other information available can be used for that purpose such as the sensor's geometry and location, robot geometry, joint configuration, sensor sensitive area, etc.

For example a bumper based contact hypothesis generator, would be single-contact and will use the sensor geometry to place the contact hypotheses on. On the other hand, a LiDAR combined with the robot geometry would be multi-contact, use the robot

geometry close to the range data to place the hypotheses and use the distance between the range data and the robot model as the likelihood of each contact hypothesis. Some more examples are given in Section 4.4 where the implemented hypotheses generators for our perceptual system are detailed.

Summarizing, to implement a new contact hypotheses generator: first, the location where the hypotheses will be generated has to be determined. Second, the type of the generator (single-contact or multi-contact) has to be specified. Finally, the type of data and the law for the generation of each hypothesis likelihood has to be selected.

4.4 Implemented regular hypotheses generators

In this section, we show the implementation of several contact hypotheses generators based on sensors that are often available in robotic manipulators.

The sensors embedded on nowadays robots, are usually platform dependent and require a specific processing depending on each embodiment. In this chapter we have only proposed the implementation of platform dependent hypotheses generators. However, the guidelines for the implementation of other types of contact hypotheses generators are given in Section 4.3. On the other hand, the implementation of the integrator is platform agnostic.

4.4.1 Experimental platforms

We have implemented contact hypotheses generators for two robotic platforms, Tombatossals and ARMAR-IIIb.

Tombatossals, is a humanoid torso with 29 DOF with advanced sensing capabilites such as tactile sensors, force-torque sensors, cameras, and a kinect, detailed information about this robot is provided in Appendix A.1. ARMAR-IIIb is a humanoid robot with 33 actuated DOFs. For the experiment we have only used its right arm (7DOF) and hand (7DOF). It has a force-torque sensor on the wrist and tactile sensor pads on the palm and fingertips. More details about this robot can be found in Apendix A.2.

4.4.2 Tactile sensor hypotheses generator

One of the main sensory cues that can provide information about contacts between the robot and the environment are tactile sensors. This kind of sensors typically produce an array of pressure values with measurements from a grid of sensing cells. In combination with the joint positions and the robot model it is possible to determine the spatial location of contacts.

If there are multiple contacts at a time, tactile sensors are able to detect them and provide sensor readings accordingly, hence this will be a multi-contact generator. In this kind of sensors, a single contact may generate marginal readings in the nearby taxels

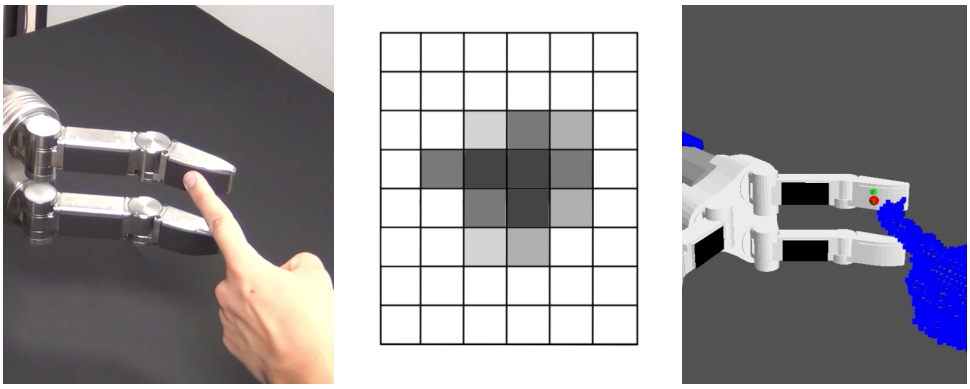


Figure 4.2: Example of the hypotheses generated by the tactile sensor generator. Left: Real scenario. Center: Tactile sensor readings. Right: Green voxels show generated hypotheses and red sphere the result of contact localization.

(see Fig. 4.2 center). However, the contact point will have a higher pressure value, thus we consider that the taxel pressure value is related with the contact likelihood at that taxel.

The hypotheses location x is determined by the activated taxels of the sensor, see Fig.4.2. The likelihood of a hypotheses located at a voxel x and with a tactile value of z_x is calculated using the following equation:

$$p(c|x,z) = \frac{z_x}{argmax(z)} \tag{4.6}$$

The tactile hypotheses generator was implemented both for ARMAR-IIIb and Tombatossals.

4.4.3 Force-torque hypotheses generator

Nowadays, service robotics oriented manipulators, often provide the external forces and torques applied to their end effectors. A straightforward solution to provide information about contacts is to use the force-torque readings. That data can be obtained from a force-torque sensor or computed using the joint efforts and the robot dynamic model. This sensory data is very valuable for contact detection because it can provide information about physical interaction.

In this subsection we present an implementation of a contact hypothesis generator based on the force-torque data detected at the end effector.

As explained in Sec. 4.3, the first step to implement a new hypotheses generator is to determine its type. It is not possible to determine the location of multiple contacts with

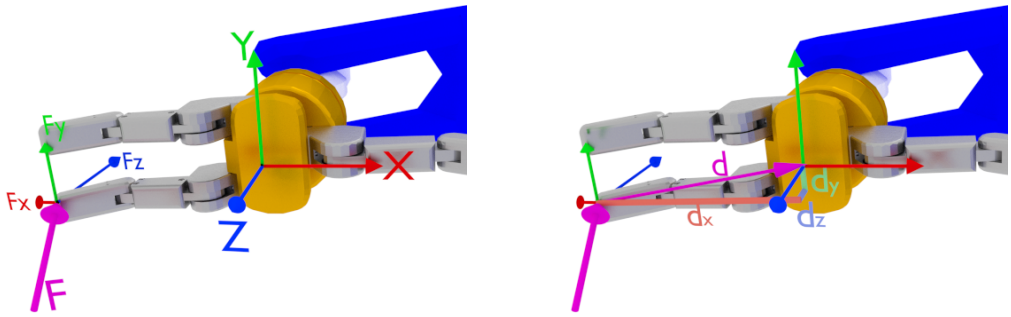


Figure 4.3: Left: A force F is applied to the finger of the robot, that force could be applied by an external actor or be a reaction to the robot interaction with the environment. The sensed force is decomposed in f_x, f_y and f_z by the force sensor. Right: The perceived torque τ depends on the distance vector d where the force F is applied from. Each component of the vector τ can be obtained from the sum of the two tangential forces.

a F/T sensor. Hence, this is a single-contact hypothesis generator because it can only detect one contact at a time.

Regarding the data obtained from the sensor, the values provided give no information about the contact likelihood, thus the likelihood of the detected contact will be uniformly distributed among all the generated hypotheses to satisfy Eq. (4.1).

In order to determine the location where the contact hypotheses will be generated, we have to take a look on how the force-torque sensor works. These type of sensors produce a 3D force vector $f \in \mathbb{R}^3$ and a 3D torque vector $\tau \in \mathbb{R}^3$. The perceived torque τ depends on the distance vector $d \in \mathbb{R}^3$ where the force F is applied from (see Fig. 4.3 right). Each component of the vector τ can be obtained from the sum of the two tangential forces that act on that component, see Eq.(4.7).

$$\tau_3 = f_{\perp 1} \cdot d_{\perp 2} + f_{\perp 2} \cdot d_{\perp 1} \tag{4.7}$$

Applying Eq.(4.7) for each of the axes, we obtain the system of equations shown in Eq.(4.8).

$$\begin{cases} \tau_x = f_z \cdot d_y - f_y \cdot d_z \\ \tau_y = f_x \cdot d_z - f_z \cdot d_x \\ \tau_z = f_y \cdot d_x - f_x \cdot d_y \end{cases} \quad \text{w.r.t. sensor frame} \tag{4.8}$$

Solving the linear equation system with 3 equations and 3 unknowns shown in Eq.(4.8), will provide the contact point ($d = [d_x d_y d_z]$). Unfortunately, the equations are linearly dependent and the system has no single solution. However, the result of intersecting the three equations is a line that can be used to generate contact hypotheses.

In order to generate a set of possible solutions, we will use only two of those equations at a time (see Eq.(4.9)) and solve the leftmost group for a set of possible values for d_x (if $f_x \neq 0$), the center group for d_y (if $f_y \neq 0$) and the rightmost group for d_z (if $f_z \neq 0$). To select the set of values used for d_x, d_y and d_z, the bounding box of the hand is expanded and used to limit the sampling space. After this, we have three lines of contact hypotheses, see Fig.4.4 center. The resulting lines are theoretically equal but experimental validation has shown that the resulting lines are not always equal, it might be due to sensor noise and may depend on the nature of the sensor.

$$\left\{ \begin{array}{l} d_y = \frac{f_y \cdot d_x - \tau_z}{f_x} \\ d_z = \frac{\tau_y + f_z \cdot d_x}{f_x} \end{array} \right. \left\{ \begin{array}{l} d_x = \frac{\tau_z + f_x \cdot d_y}{f_y} \\ d_z = \frac{f_z \cdot d_y - \tau_x}{f_y} \end{array} \right. \left\{ \begin{array}{l} d_x = \frac{f_x \cdot d_z - \tau_y}{f_z} \\ d_y = \frac{\tau_x + f_y \cdot d_z}{f_z} \end{array} \right. \tag{4.9}$$

Finally, using the distance of the generated hypotheses to the spherical model of the robot (details about the robot spherical model are presented in Appendix B), those hypotheses that are not close to the robot geometry will be filtered out (See Fig.4.4 right). The likelihood of the detected contact will be uniformly distributed among all the generated hypotheses, therefore the likelihood of a hypothesis will be $p(c|x, g_f) = 1/n$ where n is the number of generated hypotheses.

Another approach to determine the contact point regardless of end effector geometry is shown in [Karayiannidis et al., 2014]. The Force-torque hypotheses generator was implemented both for ARMAR-IIIb and Tombatossals.

4.4.4 Range sensor hypotheses generator

Tactile sensing technologies, provide accurate data about contacts and are very sensitive. However, it is not possible to cover the whole robot hand with tactile sensors and usually there is a lot of surface that cannot sense contacts using this modality. That gap is covered by force-torque sensors which can sense contacts occurring at any point of the end effector. Unfortunately, their sensitivity is not good enough to detect contacts with light objects like an empty cardboard box. As a consequence, even in heavily sensorized systems, it is common that a contact goes unnoticed which jeopardizes the performance of a contact-based manipulation system.

This subsection proposes a contact hypotheses generator based on range data provided by an RGBD active camera. This generator can be used to fill in the gap left by tactile and force-torque sensors regarding light objects and improve contact detection

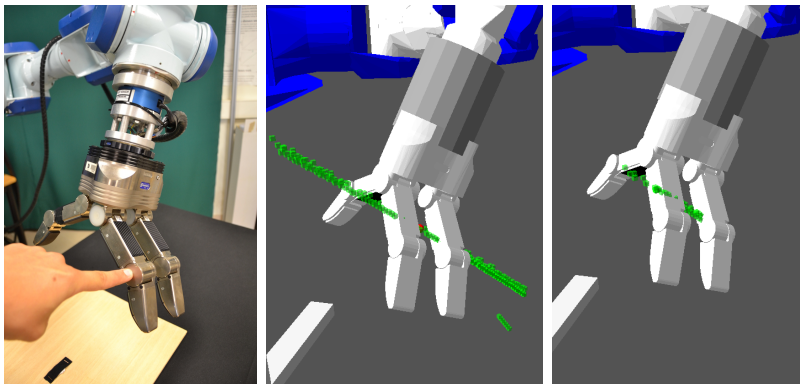


Figure 4.4: Example of the hypotheses generated by the force-torque generator. Left: Real contact. Center: Generated hypotheses before hand geometry filtering. Right: Generated hypotheses.

and localization on those situations. However, the implementation of a method able to detect contacts using an RGBD camera, is not as straightforward as using a sensor that directly measures physical interaction.

One of the problems to address is that contacts can occur on non visible parts of the object, or most likely be occluded by the robot itself. To deal with the occlusion problem, this contact hypotheses generator utilizes an Occupancy Grid Map (OGM) in order to determine where collisions can happen (Fig. 4.5).

The steps performed by the range sensor hypotheses generator are depicted in Fig. 4.6. 1) Before the manipulation task starts, the OGM of the scene is initialized. 2) While the robot moves to manipulate the object, the OGM is updated using the robot spherical model and the space traversed by the robot is removed from the possible contact locations. At the same time, the object is tracked to detect its movement. 3) Once object motion is detected, the contact location is estimated using the combination of the OGM and the spherical model of the robot (details about the robot spherical model are presented in Appendix B).

The generator assumes that if the manipulated object moves, the motion is caused by the physical interaction of the robot, no external agents are acting on the environment.

OGM initialization

The OGM is initialized by projecting each point of the initial object point cloud along the direction of the camera until it intersects with the table plane. To project the points, a perfect pin-hole model of the camera is considered. The initial object point cloud is obtained before any manipulation action is taken. To detect the table and segment out

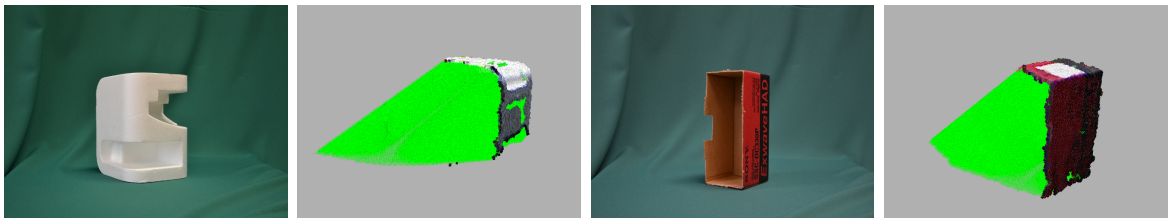


Figure 4.5: Real objects and its initial Occupancy Grid Map (OGM).

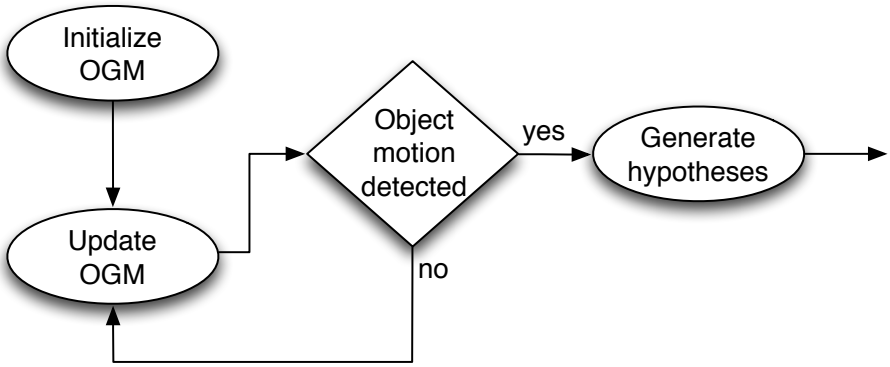


Figure 4.6: Range hypotheses generator diagram. After initializing the OGM, the algorithm keeps updating the OGM until object motion is detected. Then the contact hypotheses are generated.

the object, the same visual pipeline used in the experiments presented in Chapter 3 is used. The details of the visual pipeline are given in Section 6.3.3.

Then, the occluded area is discretized in voxels of $1mm^3$ in each direction (x, y, z). The voxels that are already present in the point cloud provided by the RGBD sensor, have a probability of being occupied $P(c_i) = 1$. Meanwhile, there is no information about the voxels that are not being seen, hence its starting likelihood of being occupied is $P(c_i) = 0.5$. The initial state of the OGM is depicted in Fig. 4.5 where the green voxels correspond to occluded space and the color voxels correspond to voxels that are seen by the camera and are occupied.

Virtual contact sensor model

To update the OGM a probabilistic sensor model is required. As there is no real sensor that can provide readings about contact likelihood, a virtual sensor based on the robot spherical model is used. The model provides the probability function of having a contact z given a voxel state c_i. The function is defined depending on the distance between the robot model and the voxel d, see Eq. 4.10.

$$P(z|c_i) = \begin{cases} 0.0 & \text{if } d < 0 \\ 0.5 & \text{if } d > 0 \\ 0.9 & \text{otherwise} \end{cases} \tag{4.10}$$

If the voxel is inside the model then $P(z|c_i) = 0$, we can be sure that the voxel is free of contact. If the voxel is outside the hand model, there is no information about the contact state of that voxel, hence $P(z|c_i) = 0.5$. Finally, if the voxel is on the surface of the hand model it is likely that the contact happened there, thus $P(z|c_i) = 0.5$.

Object motion detection

To detect the motion of the object produced by a contact, an object tracking approach is implemented. In the literature there are algorithms for model-less object tracking based on image descriptors such as SIFT [Lowe, 1999], SURF [Bay et al., 2008], or Harris corners [Harris and Stephens, 1988]. However, these methods make the assumption that the objects have texture, a regular shape or that the descriptors are visible all the time. As we do not make any assumption of shape or texture and the hand of the robot may occlude parts of the object, we have chosen the Iterative Closest Point (ICP) algorithm [Zhang, 1994] as our approach to object tracking. In particular, we use the standard ICP algorithm implemented in PCL [Rusu and Cousins, 2011]. The ICP does not require any features of the object and exploits 3D data.

ICP is applied to the object point cloud at instant t^k and t^{k-1}. The algorithm provides as a result a homogeneous transformation matrix that is decomposed in a translation

vector $T(x, y, z)$ and a quaternion $q(x, y, z, w)$ from which an angular rotation Θ is obtained.

When the magnitude of the translation vector $\|T\|$ or the angular rotation Θ are grater than a configurable threshold ($\|T\| \geq T_{max}$ or $\Theta \geq \Theta_{max}$), the object has moved and therefore a contact has happened. The thresholds should be tuned regarding the noise of the sensor used and the noise introduced if the hand occludes the object.

OGM update

The OGM is updated each time step. If no object movement is detected, the voxels that are inside or on the surface of the model are updated. The new likelihood value for this voxels is 0, as can be obtained replacing $P(z|c_i) = 0$ in Eq. 4.11. Voxels that are outside the model and not on the surface, keep the same value. Finally, if object movement has been detected the surface voxels are updated according to the following rule:

$$P(c_i|z)^k = \frac{P(c_i|z)^{k-1} \cdot P(z|c_i)}{\sum_{c_i} (P(c_i|z)^{k-1} \cdot P(z|c_i))} \tag{4.11}$$

Where $P(z|c_i)$ is the probability of a measure given a occupied voxel (c_i), in other words, the virtual sensor model. $P(c_i|z)^{k-1}$ is a priori probability to be occupied given a measure z and $P(c_i|z)^k$ is the a posteriori probability to be occupied given a measure z.

Contact hypotheses generation

When object movement is detected, contact hypotheses are generated. First, the voxels from the OGM with high likelihood are selected. Second, contact hypotheses are generated on the location of those selected voxels. Finally, the likelihood assigned to the hypotheses is uniformly distributed among all the generated hypotheses.

The range sensor hypotheses generator was implemented only for Tombatossals, although it can be implemented also for ARMAR-IIIb using its stereo head as a range data sensor.

4.4.5 Finger pose feedback hypotheses generator

This generator exploits the compliance of robotic hands. Usually, contacts on compliant fingers will move the finger joints without any control command being applied to them. The involuntary variation of compliant hand finger positions can be detected by the hand encoders and used to detect a contact on the fingers.

If there are multiple contacts at a time on the same finger, the encoders are not able to detect that situation. Thus, this is a single-contact hypotheses generator. To determine

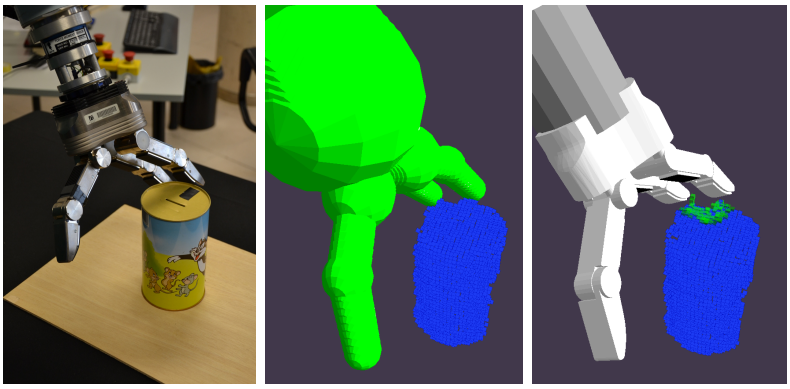


Figure 4.7: Example of the hypotheses generated by the range sensor generator. Left: Real scenario. Center: Segmentation using spherical model. Right: Generated hypotheses.

the location where the contacts will be generated, as there is no information available on the direction or position of the contact, it has to be inferred from the joint readings. Using the robot model, the contacts will be generated all over the finger's surface.

Before starting the detection, the current pose of each finger joint is stored. When a variation on a joint is detected, the finger geometry together with the proprioception and joint values is used to place the contact hypotheses. Like for the force-torque generator, the likelihood is uniformly distributed among all the generated hypotheses, see Fig. 4.8.

This generator assumes that the hand joints are not actuated, if the hand is moving or actively applying forces to the environment this generator is disabled. After moving the hand joints to a different position, the generator is reset to get the reference values updated.

The finger pose feedback hypotheses generator was implemented only for ARMAR-IIIb. Tombatossals does not have compliant hands.

4.5 Implemented support hypotheses generators

In this section we present the implementation of support contact hypotheses generators based on simulation, kinematics and control. It is important to remind that, as explained in Sec. 4.2.1, *Support* hypotheses cannot be considered standalone, *Support* hypotheses that are not fused with *Regular* hypotheses will be discarded. The final effect of *Support* hypotheses is to narrow down the contact localization projecting predictions into the *Regular* hypotheses provided by other generators. The use of *Support* hypotheses, allows us to project predictions or beliefs into the contact space. The benefits of using this approach are experimentally validated in Section 4.7.

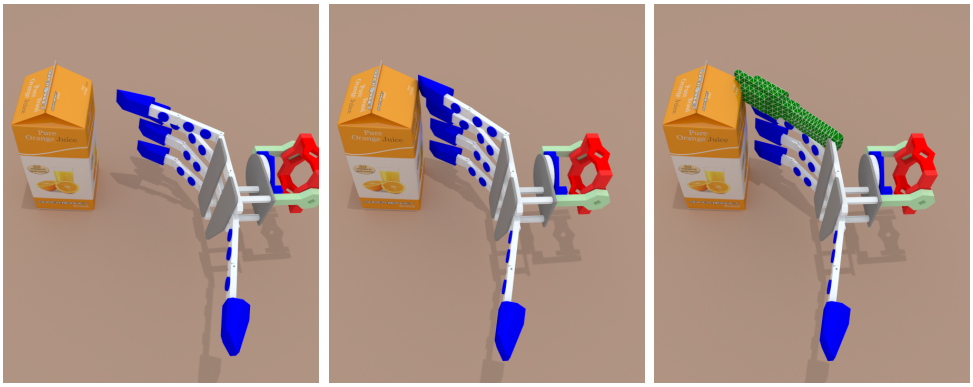


Figure 4.8: Example of the hypotheses generated by the finger pose feedback hypotheses generator. Left: Initial scene, hand moving towards the object. Center: The hand contacts the object and the compliant finger bends back. Right: Generated hypotheses on the compliantly bended finger.

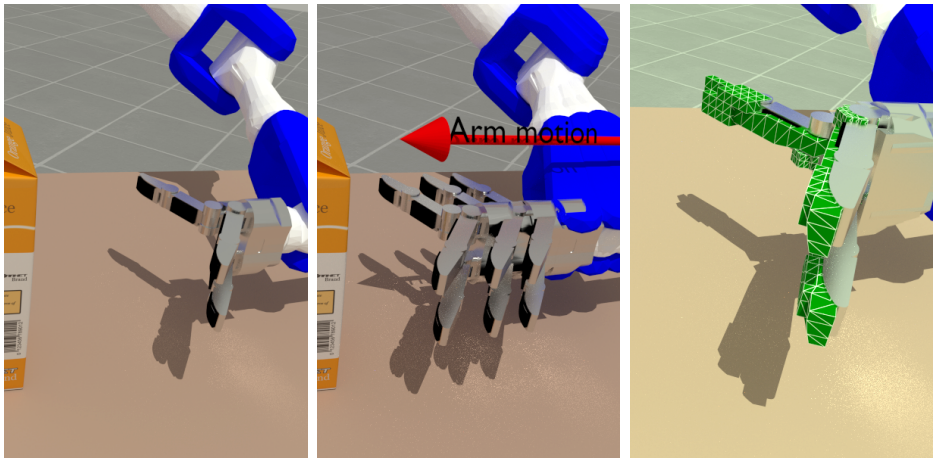


Figure 4.9: Example of the hypotheses generated by the motion estimation support hypotheses generator. Left: Initial scenario. Center: Arm motion Right: Generated support hypotheses.

4.5.1 Motion estimation support hypotheses generator

This generator exploits the current motion of the robot to produce support contact hypotheses in the locations where the robot will be in the next time step. Using the current joint positions and their velocity, contact hypotheses are generated on the next predicted hand position. As this generator is not based on physical evidence, it produces only *Support* hypotheses.

This generator estimates the position of the robot joints in the next time step using the previous joint positions. The predicted position of the robot joints $q(t+1)$ is calculated using Eq.(4.13)

$$\Delta q = q(t) - q(t-1) \tag{4.12}$$

$$q(t+1) = q(t) + \Delta q \tag{4.13}$$

With the predicted joint positions, the generator uses the robot model to produce support contact hypotheses on the space that will be occupied by the robot in the next time step as depicted in Figure 4.9.

There are no constraints about the number of contacts that could be detected using this method, thus this is a multi-contact generator. Moreover, there is no value that can be related to the contact likelihood and the hypotheses are generated with a fixed likelihood value. The likelihood of the generated hypotheses depends on the weight we want to give to this support generator. During the experiments we found that a good value is 0.3. Motion estimation support hypotheses generator was implemented both for ARMAR-IIIb and Tombatossals.

4.5.2 Simulator predictions support hypotheses generator

Another opportunity to narrow down the contact localization problem, is the use of simulators to estimate where the contacts are more likely to happen. Using the simulator as a prediction engine, together with the model of the environment and the objects, allows us to generate support hypotheses at the locations where the simulation engine detects collisions between the robot and the environment or objects.

This generator produces *Support* hypotheses because it is not directly measuring physical interaction between the robot and the environment. It uses the integrated OpenGRASP simulator as a prediction engine to detect where contacts are supposed to happen. In the simulator, the model of the robot, the environment and the objects in the workspace are previously loaded. The simulator implementation as a prediction engine is detailed and discussed in detail in Chapter 5.

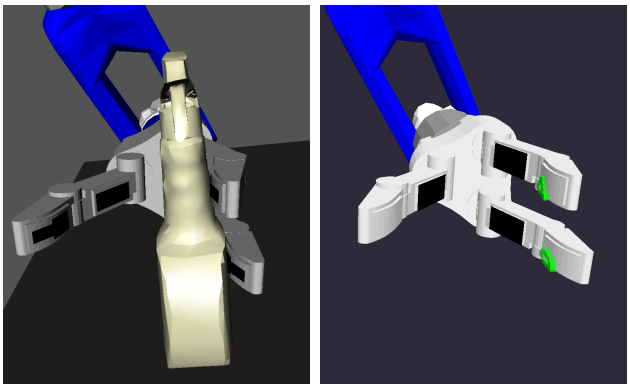


Figure 4.10: Simulator prediction support contact hypotheses generator. Left: Simulator, Right: Support contact hypotheses generated (green voxels).

This generator uses the available methods in OpenRAVE/OpenGRASP and generates *Support* hypotheses where contacts in simulation are detected (Fig. 4.10). There is no limitation on the number of contact points that this generator can detect, thus this is classified as a multi-contact generator. Moreover, the set of contacts provided by the simulation engine do not have any related confidence value that can be used as the likelihood. Hence, the likelihood used for each generated support contact hypotheses, depends on the weight that the simulator will have in the contact localization process. While performing our experimental validation, we found a good value on 0.5. This generator is only implemented for Tombatossals, it can be implemented on ARMAR-IIIb as well using its integrated simulation environment Simox [Vahrenkamp et al., 2012].

4.6 Contact hypotheses integrator

The contact detection and localization process is separated in two main steps. First, the hypothesis space HS is updated fusing the multiple inputs from the different contact hypotheses generators. Finally, the HS is traversed and a contact condensation method is applied to determine the estimated contacts that will be the output of the whole contact detection framework.

4.6.1 Hypotheses fusion

After all the hypotheses generators have provided their clouds of contact hypotheses, the fusion process is performed by the integrator. The integrator receives contact hypotheses from any number of generators, then fuses the incoming hypotheses and produces estimations of contact locations (Fig. 4.11).

As introduced in Sec. 4.2.1, the fusion is performed in the hypothesis space HS. At the beginning, the hypothesis space is empty. When a cloud of contact hypotheses is received by the integrator, the hypotheses are processed one by one and added to the hypothesis space.

When adding a new hypothesis, the integrator uses the hypothesis location to check whether that voxel is already occupied by a hypothesis. If so, both hypotheses are fused, otherwise the hypothesis is inserted into the voxel. When two hypotheses are fused, the resulting hypothesis keeps the location of the voxel and the force and direction are averaged using both hypotheses values weighted by their likelihood. The lists of likelihoods, timestamps and sources are combined with the incoming lists keeping the newest value if the same source is in both the incoming and current lists. A hypothesis is discarded when all its timestamps are older than a configurable timeout parameter. This parameter should be adjusted depending on the update rate of the hypotheses. The fusion process is detailed in Alg 4.1.

It is possible that the incoming hypotheses are generated from different sources at different rate, to perform the hypothesis fusion, the integrator waits until the hypotheses of all the active sources are received. From faster sources, the newest readings are used. The sources can be plugged and unplugged to the integrator dynamically.

After receiving and combining the contact hypotheses from all the active sources, the fused likelihood of each occupied voxel is computed using the DeMorgan's law, see Eq.(4.14). We assume that the measurements of the sensors are independent of each other.

By combining the likelihoods using Eq.(4.14) we expose the integrator to be saturated by inputs like $p(c|x, g_n) = 1$. On the other hand, the saturation will occur only on determined voxels and will not affect the entire HS. Moreover, the design of contact hypotheses generators should take into account this issue, and produce very high likelihoods only when necessary.

$$p(c|x, g_1, ..., g_n) = 1 - \prod_{i=1}^{n}(1 - p(c|g_i)) \tag{4.14}$$

4.6.2 Contact condensation

As a result of the hypotheses fusion, the hypotheses space contains a cloud of contact hypotheses with different likelihoods (see Fig. 4.11 center right). This contact hypotheses provide information about the possible contact locations, in order to provide the estimated location of the contacts, the cloud of contact likelihoods is post-processed. To obtain the estimated contacts and their location from the fused hypotheses cloud, different methods can be used:

Algorithm 4.1 Contact hypotheses fusion algorithm

```
function ProcessIncomingHypotheses(hypotheses)
    for all h in hypotheses do
        if IsVoxelOccuppied(h.location) then
            FuseHypotheses(h, GetVoxel(h.location))
        else
            SetVoxel(h)
        end if
    end for
end function
function FuseHypotheses(h1, h2)
    h3.location := h1.location
    h3.likelihood := h1.likelihood ∪ h2.likelihood
    h3.sources := h1.sources ∪ h2.sources
    h3.timestamp := h1.timestamp ∪ h2.timestamp
    h3.fMagnitude := weightedMean(h1.fMag, h2.fMag)
    h3.fDirection := weightedMean(h1.fDir, h2.fDir)
    SetVoxel(h3)
end function
```

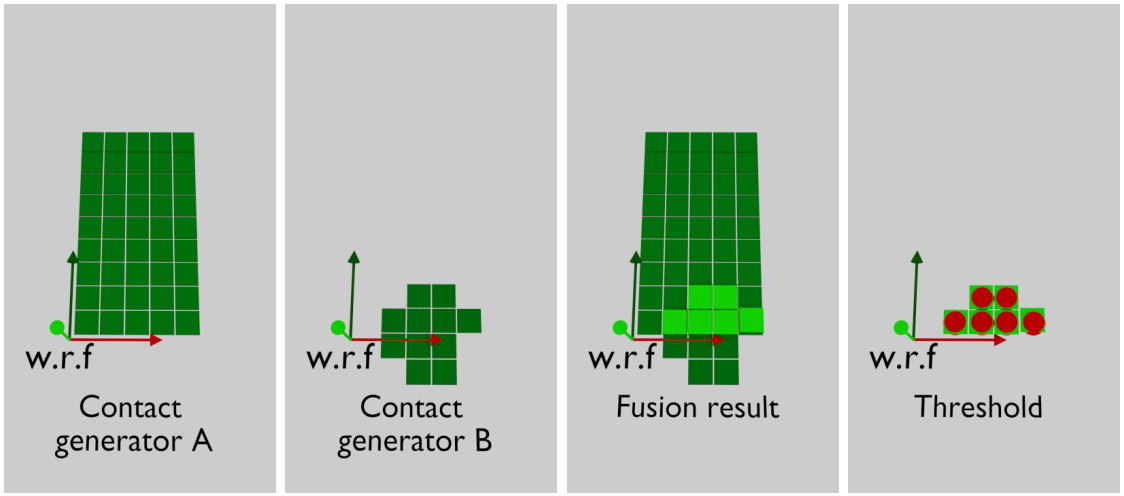


Figure 4.11: Contact hypotheses fusion and contact detection using the threshold method for contact condensation. The likelihood of each hypothesis is color encoded (Dark green: low probability, Light green: High probability). Red spheres show the result of the contact condensation step. Note that all the pictures use the same reference frame.

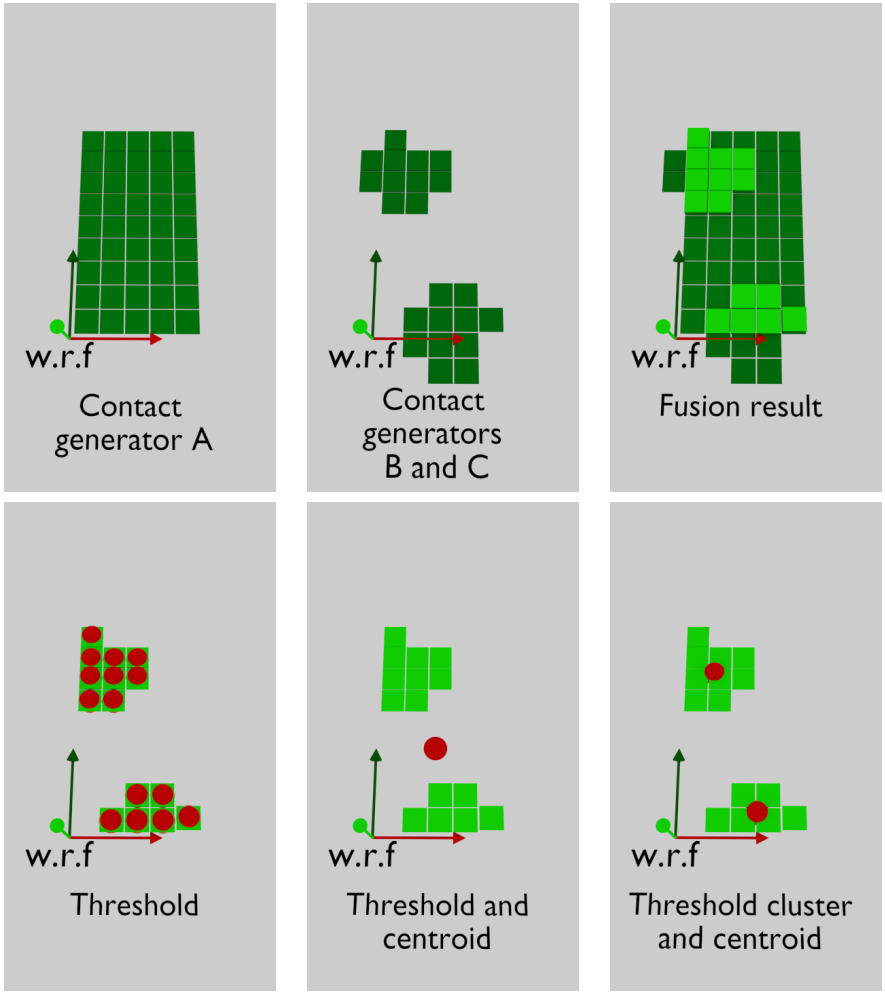


Figure 4.12: Comparison of the contact detection results using three different contact condensation methods: threshold, weighted centroid and threshold-cluster-centroid. Red spheres show the result of the contact condensation step. In this case the clustering based method is able to detect the two contacts accurately while the weighted centroid detects only one and the threshold detects too many.

- Threshold: threshold the hypotheses using their likelihood.
- Centroid: calculate the centroid weighted by the likelihood.
- Clustering: perform clustering to detect the connected hypotheses.

Those operations can be applied to the result of the fusion process. Depending on the order of application and the parameters used, different results will be obtained. We have explored the application of a simple threshold and a more sophisticated approach that performs a sequence of threshold, cluster and centroid operations. The following subsections give more details about the parameter settings and the results obtained using this approaches for contact condensation.

Threshold

A straightforward method is to set up a high threshold for the likelihood (e.g $H(x) \geq 0.6$), and filter the data to obtain the hypotheses that will be considered contacts. It is likely that the result is a cloud of contacts around the real location (see Fig. 4.11), to provide a single contact after applying the threshold, a centroid calculation weighted by the hypotheses likelihood can be calculated.

This strategy is eligible if all the generators produce very precise data, otherwise the likelihood will be distributed among many hypotheses and none of them will be over the threshold.

Threshold, cluster and centroid

On the other hand we can use a low threshold (e.g $H(x) \geq 0.1$) and calculate the global centroid weighted by likelihood. As in the previous method this will reduce the detected contacts to a single one, if there are separated contact regions the result will be the average of those regions. Thus, before performing the centroid calculation, the contact regions are detected using an euclidean clustering algorithm[1]. Then the centroid (likelihood weighted) is calculated for each cluster (See red spheres in Fig. 4.12).

With this condensation method, the system is able to detect multiple contacts at a time and localize them more precisely instead of providing a cloud of contacts. On the other hand the clustering method computation requires a nearest neighbour check for almost every hypothesis, and in high cluttered hypothesis spaces it can reduce the speed performance. A comparison of different combinations for the contact condensation is depicted in Fig. 4.11.

[1]Clustering algorithm taken from: `http://www.pointclouds.org/documentation/tutorials/cluster_extraction.php`

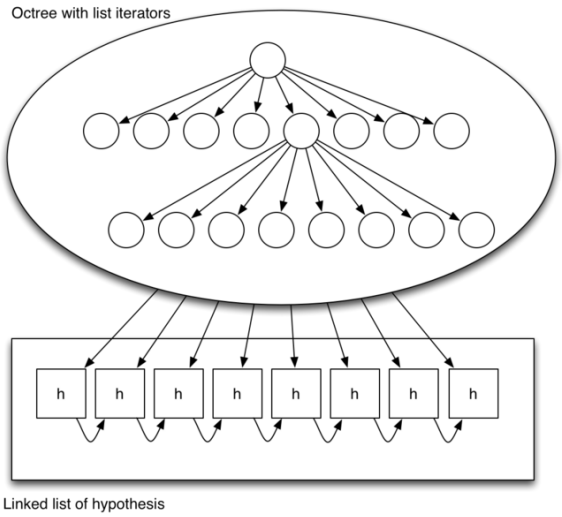


Figure 4.13: Suggested data structures for the implementation of the contact space and the sensor fusion framework. An octree indexes the linked list that contains all the contact hypotheses.

4.6.3 Implementation guidelines

The contact hypotheses space occupation is represented by an octree to efficiently retrieve whether a voxel is occupied by a hypothesis. The core implementation of the integrator is a linked list of contact hypotheses indexed by the octree (See Fig. 4.13). Each leaf from the octree contains an iterator that points to an element of the list that contains all the hypotheses. The octree is used for nearest neighbor searches in the hypothesis space, meanwhile the list is used to access all the contact hypotheses sequentially (i.e. to check the hypotheses timestamp or threshold the hypotheses by its likelihood).

4.7 Experiments

To demonstrate and validate the sensor fusion framework proposed and show its suitability for the contact detection and localization problem, we have conducted two experiments: a validation and a use case.

On Tombatossals we have performed an experimental validation of the implemented contact hypotheses generators and the fusion method. On ARMAR-IIIb we have applied the contact detection method on a real grasping situation; using the contact output information to drive a reactive grasp algorithm like [Felip and Morales, 2009] or [Hsiao et al., 2010]. We have used the same voxel size in the generators and in the integrator, 5mm side. Thus, the precision of the contact detection is limited to the 5mm resolution

of the hypothesis space. The selected contact condensation method is the threshold, cluster and centroid.

4.7.1 Experimental validation

The experimental validation is performed using the Tombatossals robot. This experiment consists on touching three different objects (a box, a cylinder and a sprayer bottle) each one from 15 different approach directions for a total of 45 touch experiments.

For the evaluation two performance metrics are used:

- Detection rate: Measures the number of contacts detected over all the contacts that really occurred. This metric provides the likelihood that the used sensor modalities detect a contact.
- Localization error: To measure the precision, we compute the error of the detected contact location as the distance to the ground truth contact location. This metric provides information about the accuracy of the contact localization.

In order to validate the benefits of using a sensor fusion approach, the contact hypotheses provided by each generator, are recorded separately. Then, the contact detection results are obtained for each modality, without fusing data from different generators. After that, we obtain also the contact detection results using all the modalities at the same time. The results are evaluated with the proposed metrics.

Experimental setup

Environment

The scenario consists of an object on a table in front of the robot, the object is in a known position inside the arm workspace, so the robot can touch it easily from different approach directions.

Test objects

Three different objects are used for the validation experiments. A box, a cylinder and a sprayer bottle (See Fig. 4.14).

Assumptions

The 3D models of the objects are known. The position of the objects on the table is also known. For the execution of the experiments, the position is kept static in the real scenario. On the other hand, to simulate the uncertainty of state of the art object pose estimation algorithms (e.g. [Aldoma et al., 2012]), the pose of the object is modified with Gaussian noise in the simulator.

Figure 4.14: Test objects used for the validation of the contact detection framework. To show the scale of objects a 1 Euro coin is placed next to each object.

Ground truth data

To obtain the ground truth data, the position of the object is calibrated using the robot left arm. The calibration is manually performed: touching several points of the real object and moving the simulated object to fit those positions. With the object position calibrated in the simulator, we have used the joint positions recorded from the experiment execution to get the exact hand-object contact points and use them as ground truth data. Two of the experiments did not really touch the object, they were removed leaving 43 touches. To keep the ground truth valid, the object is fixed and cannot be moved by the robot during the experiments.

The role of the simulator in this experiment is twofold, as a ground truth tool and as a prediction engine.

Hypotheses generators

The hypotheses generators used for the validation experiments are: force-torque sensor, tactile sensors, range sensor and simulator.

In this case, the simulator is used as a prediction engine to generate support contact hypotheses. In order to model the uncertainty introduced by state of the art 3D object recognition and pose estimation methods [Aldoma et al., 2012] we have added a Gaussian error, $\mathcal{N}(\mu = 0, \sigma = 2)$ in cm, to the objects calibrated position.

Results and discussion

The results, after the execution of 43 touches, are shown in Table 4.1. The distance between the ground truth contact and the result of the contact condensation (See Sec.4.6.2) is used as the error measure ϵ. The standard deviation of the centroid calculation performed by the contact condensation is used as the precision measure σ.

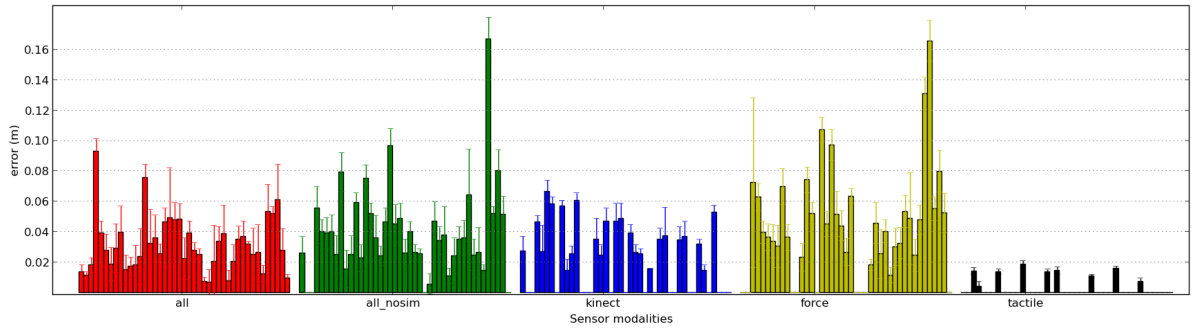


Figure 4.15: Results for the touch experiments considering different sensor modalities. Results are grouped by sensor modality, each one has 43 slots, one for each touch performed, if the contact was detected a bar shows the ϵ and σ for that contact.

Sensor Modality	Detected contacts	ϵ(cm)	σ(cm)
Tactile	9/43 (20.9%)	1.25	$\pm$0.20
Force	34/43 (79.1%)	5.37	$\pm$1.18
Range	25/43 (58.1%)	3.74	$\pm$0.73
All	39/43 (90.7%)	4.31	$\pm$1.10
All + Simulator	39/43 (90.7%)	3.33	$\pm$1.11

Table 4.1: Results for each sensor modality. Number and % of detected contacts. ϵ shows the distance between the ground truth and the detected contact. σ is the mean dispersion of the centroid calculation.

In Table 4.1 the mean accuracy and precision considering different sensor modalities is shown. Fig.4.15 depicts the individual results for each touch experiment considering different sensor modalities.

Although the tactile sensor modality has a small localization error (around 1cm ±0.2) the contact detection is quite low, 20.9%. This low detection rate is related to the reduced area that the tactile sensors can cover, thus many contacts happen outside the tactile sensor patches. Regarding the force-torque generator, although the detection rate is good (79.1%) the localization error is around (5cm ±1.18), the force-torque contact hypotheses generation method is very sensitive to noise and the effect of multiple contacts decreases the accuracy. The range modality shows average contact detection (58.1%) and good accuracy (3.7cm ±0.73), the main problem of this modality is that it requires the object to be moved in order to detect contacts.

Fusing the modalities, the detection raises to 90.7%. The accuracy depends on which sensors are detecting the contact. The fusion method automatically takes the most precise sensor (i.e. the hypotheses cloud with higher probability density). Note that the error (4.31cm ±1.1) is increased by those cases where only the force sensor generates hypotheses. This problem is solved adding the predicted contacts from the simulator, this reduces localization error by 23% leaving it at 3.33cm.

It is important to note that, beyond the sensors precision, the object position uncertainty (modelled by $\mathcal{N}(0, 2)$ cm), the robot model error and the joint encoders error also influence the final results.

4.7.2 Grasping application

To test the sensing framework on a real application, we have implemented a robust grasp primitive like the one presented in Sec.3.3.1 that uses the contact location output from the contact detection framework. The purpose is to test if the controller is able to grasp the bottle using the provided contact feedback.

Experimental setup

Environment

The scenario consists of a bottle on a table in front of the robot, the object is inside the arm workspace, so that the robot can grasp it without the need of moving the base.

Test objects

The test object consists on a transparent sparkling water bottle. The bottle is almost full as can be seen in Fig. 4.16.

Assumptions

The robot hand is placed approximately in front of the object, so it satisfies the initial condition for the grasp controller explained in Sec.3.3.1. There is no previous knowledge about the object shape, weight or other properties.

The object is not fixed and can be moved by the robot, only generators that depend on the object position (simulator predictions) would be influenced by this fact but for ARMAR-IIIb the simulator predictions are not implemented.

Hypotheses generators

The hypotheses generators used for the grasping experiments are: force-torque sensor, tactile sensors, finger pose feedback and motion estimation.

Results and discussion

The result of the experiment is shown in Fig. 4.16. The robot is able to detect the contact location fusing the information coming from the force sensor and from the arm motion. In this case, neither the tactile sensors nor the finger position did detect the contact. Using the detected contact location, the grasp controller can perform the corrective movements triggered by unexpected contacts and successfully grasp the target object even if the approach vector is not exactly in front of the object.

Under the experimental conditions, the sensor rate achieved was around 6Hz. It is important to note that all the system (sensor generators, integrator and visualization) were running on an average computer with two cores. As the hand was moving slowly for safety reasons, 6Hz were enough to react to the detected contact, correct the hand-object position and grasp the object successfully.

A more efficient implementation that decouples visualization from the contact detection framework, combined with a more efficient implementation of contact hypotheses generators resulted in a detection frame rate equal to the slowest contact hypotheses generator, faster than 30Hz.

4.8 Conclusion

In this chapter we have shown theory, validation and application of a sensor fusion method focused on contact detection. This framework provides multi-modal contact event detection to our system, a key element in the proposed approach for human inspired robotic manipulation.

One of the contributions is that the method allows input from other sources but sensors, such as context, predictions or environment. We have shown that the projection of predictions or beliefs into the sensor space improves the results.

(a) Motion support hypotheses are generated while the hand moves.

(b) Contact detected. Force-torque hypotheses are fused with motion support hypothesis.

(c) Contact information is used to correct hand pose.

(d) Finally a good grasp is achieved.

Figure 4.16: Execution on a real robot. Black voxels: Discarded hypotheses. Green voxels: Hypotheses after likelihood threshold, Red voxels: Contact detection output.

The theoretical approach has been implemented in two different robotic platforms. The experiments carried out using Tombatossals have shown that the method is suitable to be used in real environments providing a framework to fuse sensor, simulation and prediction data to improve contact detection and localization. The experiments also show that the fusion of different sources performs better than the sources separated.

However, it is important to highlight that the likelihood threshold used in the contact condensation to discard low probability hypotheses is a key parameter. Its selection may change the behaviour of the sensor fusion system. Moreover, the clustering approach used to determine how many contacts are detected, has a computational cost very sensitive to the number of hypotheses considered. If the hypotheses number grows too high, it may affect the performance of the contact detection system.

Using a common representation for all the sensory inputs enables contact based controllers, presented in Chapter 3, to be more hardware independent. This makes systems more portable (same inputs), scalable (easy to add new sensors) and robust (failure tolerant). Moreover, as we show in this work, this level of abstraction enables the addition of non sensor data like: context, control or predictions. In this way we provide the manipulation primitive framework with a platform independent perceptual system that can be used to implement hardware agnostic primitives allowing the robots to share plans more easily.

The implementation on multiple platforms shows that the approach can contribute to the sensor skills of any robot or even enable the interaction of different robots on sharing and combining their knowledge about existing contacts. This opens the door to multi-robot scenarios, where contact hypotheses generators from different robots can be used together to share physical interaction information and detect contact events. The framework is not only limited to robots but to any source of contact information, it is also suitable for ambient intelligence.

The precision of the contact detection is limited by the accuracy of the contact hypotheses generators. Hence, further research on the proposed contact hypotheses generators or the addition of more precise sensors will help to improve the overall performance of the system. Moreover, with a few modifications the framework can also be used to detect surprise (prediction and sensing mismatch) and enable low level reactive behaviours, internal model refinement or higher level reasoning.

The described framework, contact generators and experiments were published in [Felip and Morales, 2014]. The RGBD based method used to implement the range sensor contact hypothesis generator was published in [Bernabe et al., 2013].

Chapter 5

Contact event prediction

An important component of human manipulation is the ability to predict the sensor feedback produced by our actions. As explained in Chapter 2, humans use the predicted contact events to monitor the task execution and perform corrective actions when mismatches are detected. In this thesis, we have already presented the contact event detection framework in Chapter 4 and the action-phase controllers in Chapter 3. In this chapter, we discuss and present an implementation of a contact event prediction mechanism. It can be used in conjunction with the components presented in the other chapters in order to implement the contact based robot manipulation framework proposed by this thesis.

Different approaches could be used to implement the contact event prediction engine. One option is to use past experiences to determine where and when contact events shall be detected [Pastor et al., 2011]. However, it requires a learning process that provides past successful experiences for any new task that we want the robot to perform.

An alternative general implementation, is the use of a simulator to predict the outcome of actions, in this way the robot is able to foretell when and where contact events should arise. Unfortunately, real-time predictions are not easy to achieve and accuracy is usually compromised for the sake of real-time performance. The prediction engine shown in this chapter consists of the dynamic simulation of the robot.

5.1 Introduction

Simulators have accompanied robotics for a long time and have been an essential tool for their design and programming. In robotics research, simulators have an important role in the development and demonstration of algorithms and techniques in areas such as path planning, grasp planning and mobile robot navigation. In the context of grasping and dexterous grasping, simulation has been mostly limited to replicating the kinematics but not the dynamics of the robot manipulators. Dynamics simulation, which takes into account masses, forces, inertias, static and dynamic frictions, and even elasticities and deformations, is a very challenging problem, specially when it comes to considering

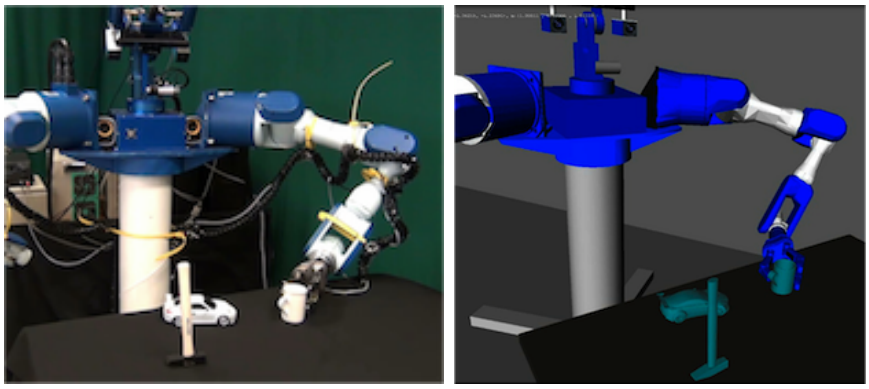


Figure 5.1: Left: Real robot. Right: Simulated robot (Tombatossals)

the interactions between contacting bodies. A large number of parameters, which are difficult to determine, affect the dynamics behaviour of the involved parts.

In the last years the use and development of physics engines has become increasingly popular in the robotics community for simulating the dynamics behaviour of a robot, specially in the case of mobile robotics. These packages usually include collision checkers, friction and contact models and a varied type of motion constraints to enable the simulation of articulated bodies. They also come with many limitations regarding the accuracy of the simulated behaviours, and computational time constraints.

A complete dynamic simulation of a robot can be of great benefit for robotics research. First, the simulation could replace the real hardware to the extent that it reproduces the actual physical behaviour, which is of special importance in the context of robot manipulation. Second, it can be used as an accurate prediction engine that can help us to understand the effects of actions and be the base for developmental learning. Additionally, if simulation accurately reproduces the real sensor and actuator feedback, robots could automatically learn from low-level sensor inputs without the erosion of real hardware.

In this chapter, we address the challenge of developing the full dynamics simulation of a complete robot (see Fig. 5.1), including the dynamics of the bodies, the actuation of the motors and the simulation of the sensor readings. The purpose of this work is to develop a complete framework, using existing tools, to assess the suitability of simulation as a surrogate of a real robotic grasping system. Furthermore, the simulator will be used to provide a contact prediction engine to our manipulation framework.

In order to validate the simulation engine, we propose and implement three of the most representative manipulation tasks and compare the real behaviour with the simulated one. There has been little work published about dynamics simulation of grasping, and what has been done, performed experiments with very tight constraints (e.g [Weit-

nauer et al., 2010]). For this reason, the proposed experiments are mainly focused on evaluating the simulation behaviour of performing realistic robot manipulation tasks. Finally, we compare the results obtained from the simulation with the ones obtained from real execution and report the degree of success in each of these experiments and the accuracy obtained.

5.2 Related work

Many robot simulators have been developed in the last few years, specially for mobile robotics [Carpin et al., 2007, Gerkey et al., 2003, Jackson, 2007, Kanehiro et al., 2002, Michel, 2004a, Echeverria et al., 2012]. However, the variety of simulation tools for robotic grasping is rather limited. The most renowned and prominent one is *GraspIt!* [Miller and Allen, 2004], which has several limitations including its lack of modularity. Another existing and publicly available software framework is OpenRAVE [Diankov, 2010]. It has been designed as an open architecture targeting a simple integration of simulation, visualization, planning, scripting and control of robot systems. It has a modular design, which enables its extension by other users. There are other more general simulators that can be also used for robotic grasping simulation like Gazebo [Koenig and Howard, 2004] or V-REP [E. Rohmer, 2013]. Another promising solution: MuJoCo proposed by [Todorov et al., 2012] is still not open to the public to test it. However, the most important component of a simulator, in order to achieve high fidelity simulations, is the physics engine. The available robot simulators use physics engines to calculate the dynamic interactions with the environment.

In the last decade, the interest of the video game industry in realistic effects, has pushed forward the development of physics simulation engines. The main problem of game oriented physics engines is the seek for real-time visual realism instead of physical realism, thus it is more important to look like the reality than actually exactly simulate it. Nevertheless, the results of the most prominent game physics engines are impressive and used in many simulators. There are many physics engines available, however not all of them are still under development. Some of the most active and advanced engines are: Bullet [Coumans, nd], Newton Game Dynamic [Jerez and Suero, nd], Havok[1] and PhysX[2]. The most popular rigid body dynamics library used by most of the state of the art robotics simulators is ODE (Open Dynamics Engine) [Smith, 2007]. Other free open source engine is dvc3D [Nguyen and Trinkle, 2010]. However, they have several limitations which jeopardize their ability to accurately reproduce dynamics simulation [Drumwright et al., 2010]. Another category of physics engines is populated by commercial simulation engines like AGX Dynamics [AgX Dynamics, 2015] and Vortex [CM Labs, 2015] which provide accurate dynamics simulation but are not freeware.

[1]Havok Physics: `http://www.havok.com/physics/`

[2]NVIDIA PhysX: `https://developer.nvidia.com/gameworks-physx-overview`

To ease the problem of selecting a physics engine and switching from one to another if necessary, there have been efforts on middleware implementation that can abstract the physics engine to the simulators programmers such as PAL [Boeing and Bräunl, 2007], OPAL[3] or FISICAS[4].

There have been some attempts to create a simulation of robot manipulation dynamics. [Laue and Hebbel, 2009] described a method to optimize a set of simulation parameters using evolutionary algorithms in the context of mobile robotics. [Weitnauer et al., 2010] studied how accurately the pushing of flat objects across a table can be predicted by a physics engine adapting some of its parameters to enhance the simulation. [Zhang et al., 2010] simulated planar grasping actions using experiments to calibrate the system and then evaluating the results using a hand with one degree of freedom. However, these simulations have been performed using very simple and controlled experiments in order to reduce the complexity of dynamics simulation.

5.3 Components selection

As shown in the previous section, there is a wide variety of simulators and physics engines available. Thus, it is likely that there will be a suitable simulator-physics pair for our target application: the simulation of grasping and contacts. Unfortunately, the physics engine choice is not independent of the simulator selected, the simulator must implement an interface to the selected physics engine or at least provide a plugin architecture that allows the users to implement their own physics engine interface. The architecture of the prediction engine is depicted in Figure 5.2.

5.3.1 Physics engine selection

To avoid the problem of selecting a specific engine, a reasonable solution would be to use a physics abstraction layer like PAL [Boeing and Bräunl, 2007], OPAL[3] or FISICAS[4]. However, they do not support all the physics engines available and their development is discontinued, thus they could not adapt to newer versions of the engines and possible API changes. This issues render the generic solution to be a dead end and a specific engine has to be chosen.

In order to decide which of the available physics engines is the best fit for our application, there are several criteria that can be taken into consideration:

- Performance: Most of the state of the art engines provide iterative solvers that allow the developer to set the accuracy-speed trade-off accordingly to the application requirements. Nevertheless, the accuracy obtained from the different physics engines for the same number of iterations might vary.

[3]OPAL Physics Abstraction Layer `http://opal.sourceforge.net/index.html`
[4]FISICAS is part of the OpenGRASP project `http://opengrasp.sourceforge.net/#fisicas`

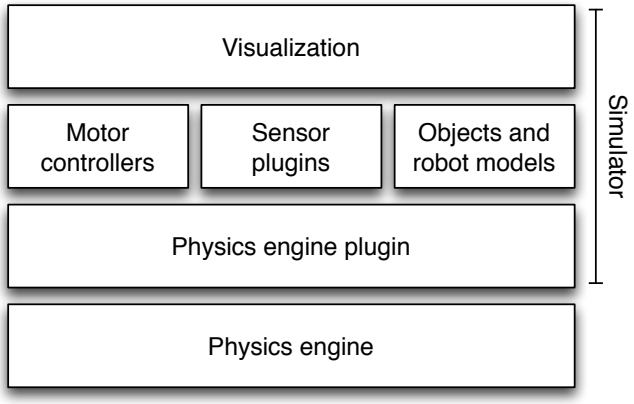


Figure 5.2: Components of the prediction engine composed mainly by a simulator and a physics engine.

- Standardization: A physics engine such that its Application Programming Interface (API) provides generic calls that are similar to other engines is easier to understand, use and migrate. Another important feature, is the support for the COLLAborative Design Activity (COLLADA)[5] Physics standard description, which will allow the engine to create the physics entities right away from a COLLADA file.
- Documentation: It is desirable that there is enough documentation and an active developers community that provide support for the possible implementation or performance issues.
- License: Open source, General Public License (GPL), Berkeley Software Distribution (BSD) or any other license that allows the integration of the library free of charge.

Despite the information that can be found on developers forums and blogs, there are scientific publications that try to guide the potential users towards the best solution [Boeing and Bräunl, 2007]. However, the comparison of the available engines does not focus on useful features for robotics but are gaming oriented, hence clear conclusions are not provided. Despite being good review papers the decision cannot be based only on those results or opinions.

The experts on the topic agree that there is no physics engine best at all the categories. Depending on the application and the type of bodies and interactions, one engine could be better than another. To address that problem, a tool for benchmarking physics

[5]COLLADA exchange file format: `https://www.khronos.org/collada/`

engines[6] was published by a member of the PhysX team, the source code is available and the plugin-based architecture allows other developers to implement their own interfaces to add extra support for other engines, out of the box it includes support for Havok, PhysX, Bullet, Newton, NovodeX and ICE. Although the results clearly point in the PhysX direction, there is still discussion among the engine developers about the fairness of the test. Thus, the conclusion is still not clear and the tests are still not focused on robotics simulation.

Although commercial engines like AGX Dynamics [AgX Dynamics, 2015] and Vortex [CM Labs, 2015] look very promising and are used for professional mechanical simulations, their commercial licensing makes it difficult to include them in comparisons with other engines and evaluate their suitability.

In a recent physics engine comparison, [Erez et al., 2015] provided some interesting results about robotics related experiments, however the results pointed out that the best fit would be the use of MuJoCo engine [Todorov et al., 2012] which is a licensed software and its documentation is still work in progress. Up to now it is only available to developers or testers that can work without proper documentation. Once the documentation is ready, it will be released for the public as a licensed software, not open source neither free for educational or academic research purposes.

After the preliminary study about physics engines presented in this subsection, we conclude that to be sure that the best physics engine is used, we have to implement interfaces to all of them, design and perform intensive testing, analyse the results and make a decision. As it is likely that engine performance will vary in different robotics scenarios, it will be good to be able to switch from one engine to another when necessary. In the evaluation of engines published by [Erez et al., 2015], the engines are evaluated using 4 different tests, but there is a specific test very relevant for the purpose of this thesis: a grasping test. An arm (modelled after the Shadow Hand a 35 DoF system) is grasping an object and shakes it. The aim of this test is to determine the stability of the engine under such constraints. The best performing engine is MuJoCo [Todorov et al., 2012] but the second best engine is ODE [Smith, 2007]. For this reason, ODE is the physics engine selected for our system. Nevertheless, the addition of other physics engines is desirable in the future.

5.3.2 Simulator selection

The physics simulation is the core of the prediction engine, it is possible to use just the physics engine for the implementation of the contact prediction component of the manipulation system, however it is very difficult to evaluate and to tune the parameters without the tools that a simulator provides. For example, a problem that could easily be detected visualizing the simulation result (e.g. an inverted joint) would be much

[6]PEEL: The Physics Engine Evaluation Lab `http://www.codercorner.com/blog/?p=1169`

more difficult to detect without visualization. Moreover, other graphical tools such as plots, force visualization, contact and distance queries visualization are very valuable tools for the validation of a simulated platform. In our approach, the simulator is used as a front-end of the underlying physics engine (see Fig. 5.2).

There are multiple robotics simulators that provide enough good features suitable for our purpose. Nevertheless, the simulator selection impact in the performance of the prediction engine is not as critical as the physics engine selection. We have considered some of the most relevant and used simulators in the robotics community: MORSE [Echeverria et al., 2012], Webots [Michel, 2004b], OpenRAVE [Diankov, 2010], Gazebo [Koenig and Howard, 2004] and V-REP [E. Rohmer, 2013] to name a few. From those that fulfil our requirements we will choose one. However, having different simulators available would be useful to compare them and use the best for each specific application.

The first requirement is that the simulator is integrated with Open Dynamics Engine (ODE), this discards MORSE from the candidates as it only supports Bullet, furthermore their authors claim that "we do not expect to accurately simulate robot arm dynamics or fine grasping". The second requisite is the licensing, it has to be free to use for academic purposes, Open Source if possible. Unfortunately, this removes Webots from the list, its modified version of ODE would have been an interesting feature to test. Other requirements are Robot Operating System (ROS) integration, support for COLLADA models, sensors (tactile, depth, cameras, force-torque) and modularity through a plugin based architecture.

In this chapter we have used OpenRAVE [Diankov, 2010] as the main simulation environment, which provides all the features required for our implementation and much more such as direct and inverse kinematics, path planning, ray tracing, distance queries and has a very modular and plugin based architecture. Nevertheless, Gazebo [Koenig and Howard, 2004] and V-REP [E. Rohmer, 2013] could also be used because they also provide the required features and are well known and stable simulation environments that have showcased outstanding results. Although we have devoted some effort on the implementation of our prediction engine to use those simulators, it is still work in progress.

5.4 Robot implementation

5.4.1 Real robot setup

The real robotic platform used to validate the dynamics simulation is the UJI humanoid torso Tombatossals. It has 25 DOF (see Fig. 5.1) and is composed of two 7 DOF Mitsubishi PA10 arms. The right arm has a 4 DOF Barrett Hand and the left arm has a 7 DOF Schunk SDH2 hand. Both hands are endowed with a Weiss Tactile Sensor system on the fingertips. Each arm has a JR3 Force-Torque sensor attached on the

wrist between the arm and the hand. The visual system is composed of a TO40 4 DOF pan-tilt-verge head with two Imaging Source DFK 31BF03-Z2 cameras. Attached to the centre of the pan-tilt unit there is a $Kinect^{TM}$ sensor from $MicrosoftCorp$. More details about this robot are given in Appendix A.1.

5.4.2 System architecture

The architecture that controls the system is separated in low and high level. The low level receives joint velocities, sends them to the motors and provides the sensor data to the high level. It has been implemented in C++ and makes use of ROS for inter-module communications, ROS is a middleware that provides a message passing framework and a set of robotics oriented open source software libraries and tools. The simulated robot implements its own low level system using exactly the same inputs and outputs as the real robot low level system. Thus the upper layer can transparently run any controller without knowing if the controlled robot is real or not. The details about the system architecture are given in Chapter 6.

5.4.3 Simulation and physics engine

The simulated representation of Tombatossals was made to work with OpenRAVE [Diankov, 2010]. Some plugins, developed by other authors, that simulate tactile and force-torque sensors are used. Those plugins are available in the robot manipulation toolkit OpenGRASP [León et al., 2010]. The ODE physics engine is integrated through the ODE plugin already available in OpenRAVE. However, the collision forces between the tactile sensors and the objects were calculated using the tactile sensor plugin from the OpenGRASP toolkit [Moisio et al., 2012]. Thus, when objects are in contact with the tactile sensors, the reaction forces are calculated by the plugin and then applied to the colliding bodies using the ODE interface to apply external forces on objects.

The geometrical representation of the robot, was obtained from different sources and modified to adjust it to the values of the real robot. The torso and the head CAD models were created according to the measurements of the real robot. The models of the Mitsubishi PA10 arms and the Barrett hand were taken from the OpenRAVE robot model database, assembled and modified to fit the real robot model. Finally, the Schunk hand CAD model used, was provided by the manufacturer.

The inertial properties of the arms were extracted from the robot's manual and CAD drawings. For the hands, the inertial properties, mass and center of mass were estimated from the 3D models provided by the manufacturers.

Sensors

The tactile sensors were simulated using a tactile sensor plugin developed by [Moisio et al., 2012] for OpenRAVE available in OpenGRASP [León et al., 2010]. The model

of the tactile sensors considers soft contacts and a full friction description including stickslip phenomena. The sensor model consists of a surface contact patch described by the mesh of the contact elements. Therefore, meshes with the appropriate geometry for each of the robot tactile sensors were created. The parameters needed to adjust the model of the tactile sensors were the static and dynamics friction, and the stiffness K.

The force sensor is also simulated as a plugin for OpenRAVE. The physics engine is queried for the force calculated on the sensor's body. However, the results obtained are very noisy and far from the real force sensor feedback. Therefore, the comparison with the real sensor is impossible. Further improvements in this model are needed, including testing other physics engines to see whether the problem is related with ODE.

The cameras were modelled using the camera sensors provided in OpenRAVE with the same instrinsic parameters and position as the real cameras. An ideal pin-hole camera model is used by OpenRAVE to simulate the cameras.

Actuators

The angular motors, available in the robot arm and hand joints, have been simulated with the ODE controller provided by OpenRAVE. Each simulated servo-motor is parameterized by the maximum speed, the maximum acceleration and the maximum torque that the motor can apply. To control each motor, an interface to the ODE velocity controller plugin has been developed. It enables the simulated robot to receive joint velocity commands from the controller through ROS, allowing the same control messages to be used by both real and simulated robots.

5.5 Experimental Setup

In order to evaluate the fidelity of the simulation during manipulation tasks, we conducted three manipulation experiments on both the simulated and the real robot using the same task definitions and controllers. Joint trajectories, sensor readings and object pose were recorded during both executions and compared to determine the similarity of the simulation to the reality.

The selected manipulation tasks were chosen to have different types of interaction between the robotic hand and the manipulated object: grasping, in-hand manipulation and sliding. For each manipulation task, a different object was used. The experimental environment consists of the Tombatossals robot in front of a table and next to a wall on its left (see Fig. 5.1). For these experiments only the left arm of the robot (see Fig. 5.3) was used.

Data from each task is gathered at a constant rate. Each sample from the real execution is compared with its corresponding from the simulated one. As explained in Sec.5.2, there is little work published about dynamics simulation of grasping, and what has

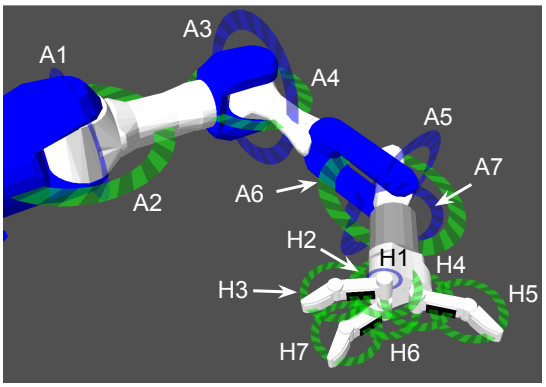


Figure 5.3: Arm (A) and hand (H) joints of the Tombatossals left arm.

been done, performed experiments with very tight constraints. For this reason, in this chapter the experiments are mainly focused on evaluating the simulation behaviour of performing the most common robot manipulation tasks. The grasp task is executed five times from different approach vectors, while the sliding and the in-hand manipulation are executed always with the same starting conditions. Each task is executed independently on the real and on the simulated robots.

The time step specified in the simulation needed to be small to get accurate results for the tactile sensors (1/1000 s). Increasing too much the simulation time step, would not only produce inaccurate results but also make the simulation unstable and produce unexpected and unrealistic motions such as joint explosions or objects flown off the scene.

5.5.1 Manipulation tasks

In order to validate the dynamics simulation of the humanoid torso, three different manipulation tasks have been executed both in real and in simulated environment.

Grasp

This task consists of grasping a box (116x103x218 mm, 143 g), lifting it 10 cm and placing it down again. The hand starting positions are depicted in Fig.5.4. The grasping task is defined using the manipulation primitives paradigm presented in Chapter 3 as a sequence of primitives: approach, grasp, lift, place, release and retreat. The movement and transport primitives are basically devoted to move the arm to the desired position. Meanwhile, the grasp controller closes the hand until a certain force is detected with the tactile sensors, the reactive grasping primitive is not used in this experiments. The grasp is executed from five different positions starting from a top grasp and rotating the hand around the object towards a lateral grasp (see Fig.5.4).

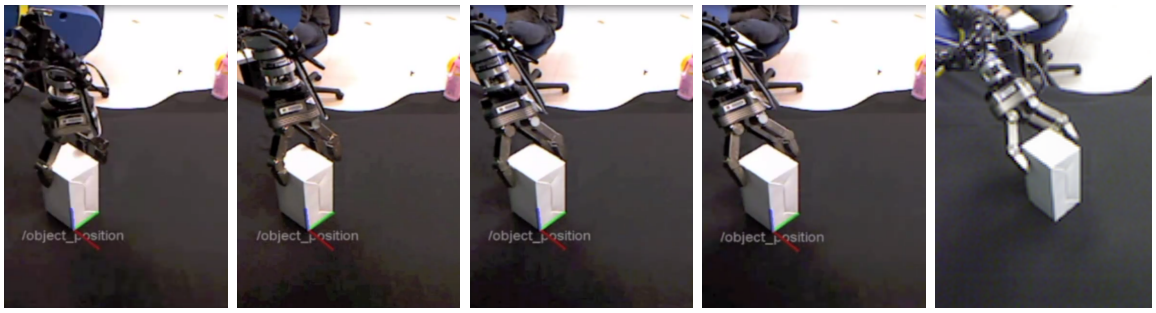


Figure 5.4: Grasping task starting positions. The object pose is acquired by a model based tracker and depicted as a reference frame.

Figure 5.5: Object sliding experiment. The robot slides the box from the starting position (left) to the target position (right). The object pose is acquired by a model based tracker and depicted as a reference frame.

Slide

This task consists of sliding an empty pizza box (320x370x55 mm, 263 g) from the initial position to the target position 35cm to the left as shown in Fig.5.5. The starting and target positions are defined in advance and the sliding task consists of the following manipulation primitives: approach, slide and lift. The slide manipulation primitive (described in Sec. 3.3.5) looks for a first contact with the tactile sensors. While the contact is detected the arm moves towards the target position. If there is too much force detected the arm moves up and if the detected contacts disappear the arm moves downwards until they are detected again. This behaviour continues until the hand reaches the target position.

In-hand manipulation

This task consists of swinging a wooden stick (60x530x16 mm, 370 g) that is already grasped by the robot. The starting setup is the robot holding the stick. In this case, instead of a task defined by a set of manipulation primitives, we have implemented a hand joint pose sequencer that moves the hand joints between three predefined positions: center, swing left and swing right. The controller starts moving the joint fingers to

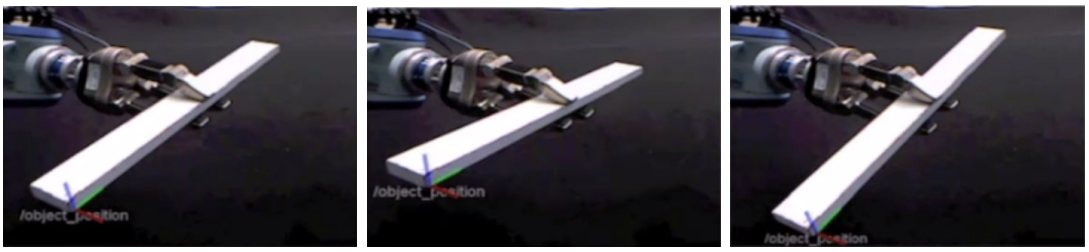


Figure 5.6: In-hand manipulation experiment. Center, swing left and swing right hand joint positions. The object pose is acquired by a model based tracker and depicted as a reference frame.

the first position (center). When the center position is reached, the hand starts moving to the next joint configuration (swing left). Then the hand moves its joints to the swing right configuration and starts over. The sequence of hand joint positions are depicted in Fig. 5.6. Force and tactile feedback are not used by this controller.

5.5.2 Metric definitions

A set of metrics to evaluate the similarity between the simulator and the reality has been defined. The compared values are: arm joint values, hand joint values, manipulated object position and tactile sensor readings.

Joint values

The joint values of the arm and the hand are logged to validate that the simulated actuators behave like the real ones. Although the similarity may not be 100% accurate, a similar behaviour during all the manipulation tasks would be enough to conclude that the simulator can be used to replace the reality in at least such conditions.

Tactile sensor readings

Comparing tactile sensor readings would not only provide the precision of the simulated sensors but the difference between the instants of detection of contacts and the difference between the detected values. The information obtained by the tactile sensors can be shown as an image. Fig. 5.7 shows an example of the tactile readings during the grasping task, the figure also depicts the centroid as red dots on each tactile patch. Regarding the pressure value detected on each taxel, real sensors are noisy. However it is very unlikely that false positives are produced. For this reason, the centroid of each tactile array is used for comparison.

Object position

To verify that dynamics simulation of manipulation tasks is achievable, the interaction between the robot and the simulated environment must be evaluated. Towards this end,

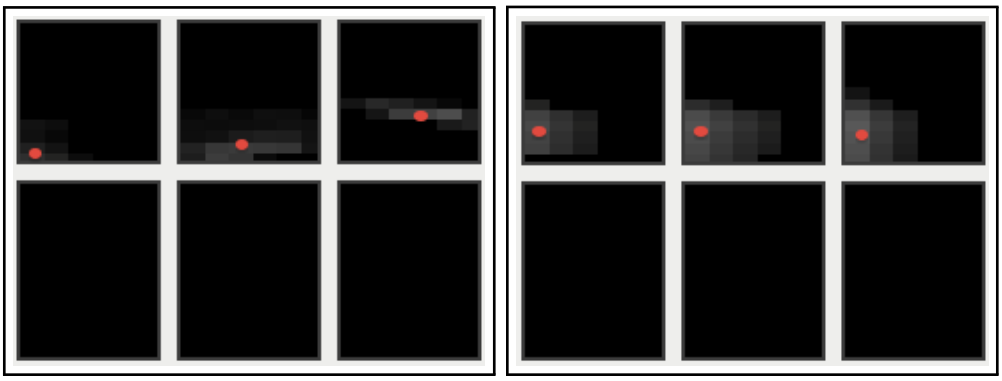


Figure 5.7: Tactile sensor data with centroids during the grasping task. Left: Real. Right: Simulated.

we have used the manipulated object pose as a metric. From the simulator, the object position is easily obtained given that the physics engine provides its center of mass at each time step. However, in order to get the object position from the real execution, an object tracker is needed. The objects used in the experiments have been modelled to use the model-based object tracker from VISP [E. Marchand, 2005]. This method is able to perform on-line object 6D tracking with enough accuracy to validate the simulator results. The object reference frame is located in one of its corners as shown in Figures 5.4, 5.5 and 5.6. As the reference systems for the real and simulated object are not the same, the variation relative to the object starting position is used as the comparison metric.

5.6 Results

The analysis was performed from three different aspects: 1) visual inspection of the videos recorded during both experiments, 2) detailed quantitative results for each task and 3) global quantitative results that summarize how close the simulation was to reality.

5.6.1 Visual inspection

The experiments were successfully carried out for the different manipulation tasks. At plain sight, the simulated behaviour matches the real one quite correctly. Although looking at the recorded video, the simulated execution is not showing perfect synchronization with the real one, it shows that the motion of the robot and the manipulated objects are very close to the real execution.

The experiments in the real scenario took between 37 and 93 seconds. The time step specified in the simulation needed to be small to get accurate results for the tactile

sensors (1/1000 s), for this reason the time taken to execute the simulation experiments was significantly larger. They took between 10 and 30 minutes.

5.6.2 Detailed quantitative results

The results obtained for the real and simulated environment were compared using the metrics explained in Sec. 5.5.2. As the duration of the experiments is different for the real and simulated environments, the results are presented over the progress of the experiment from 0% to 100%. To do so the starting and ending times were stored. To determine the starting time of a experiment, the first instant when the robot moved was considered. To determine the ending time the timestamp of the success event thrown by the last manipulation primitive was used.

Grasp

The five grasp experiments were executed successfully on real and simulated scenarios, visual inspection could not detect noticeable differences among both executions. Detailed results for each metric are shown in figures 5.8, 5.9, 5.10 and 5.11 for the first grasp position.

The controller used for the task execution on real and simulated scenarios is exactly the same, ideally the response of the motors is expected to be equal in both environments. Although not exactly the same, the arm joints show very similar behaviour (see Fig. 5.8) during the movement and all of them converge exactly to the same position at the end of the task. Regarding the hand joints, the motion is again similar but there are bigger differences (see Fig. 5.9). The controllers used are the same, but they are reactive controllers that adapt to the sensory input, in this case the tactile sensors do not detect the contacts at the same time which results in some joints stopping at different positions. Moreover, the deformation of the object when grasped is not modelled in the simulator, that explains why joints H2, H5 and H6 are not able to close as much as the real joints do, see Fig. 5.9.

Figure 5.10 shows the detected centroid position of each tactile sensor and its evolution during the grasping task. It can be seen that the sensors of the real robot start to provide feedback ahead of the simulated ones. This difference can be caused by the sensitivity of the simulated sensors, a parameter that can be tuned for further experiments. Despite the timing difference, when both sensors are detecting contact, the difference of the contact centroid is always less than 4 texels, which allows the controller the use of simulated and real sensor feedback in a similar way. The position of the object recorded by the tracker, has significantly more noise than the acquired by the simulator (see Fig. 5.11). However, the results are very similar in all axes, especially in z which is the direction of the object's movement. The rotation of the object in this experiments did not play a significant role which can be seen in the small variation of the plots.

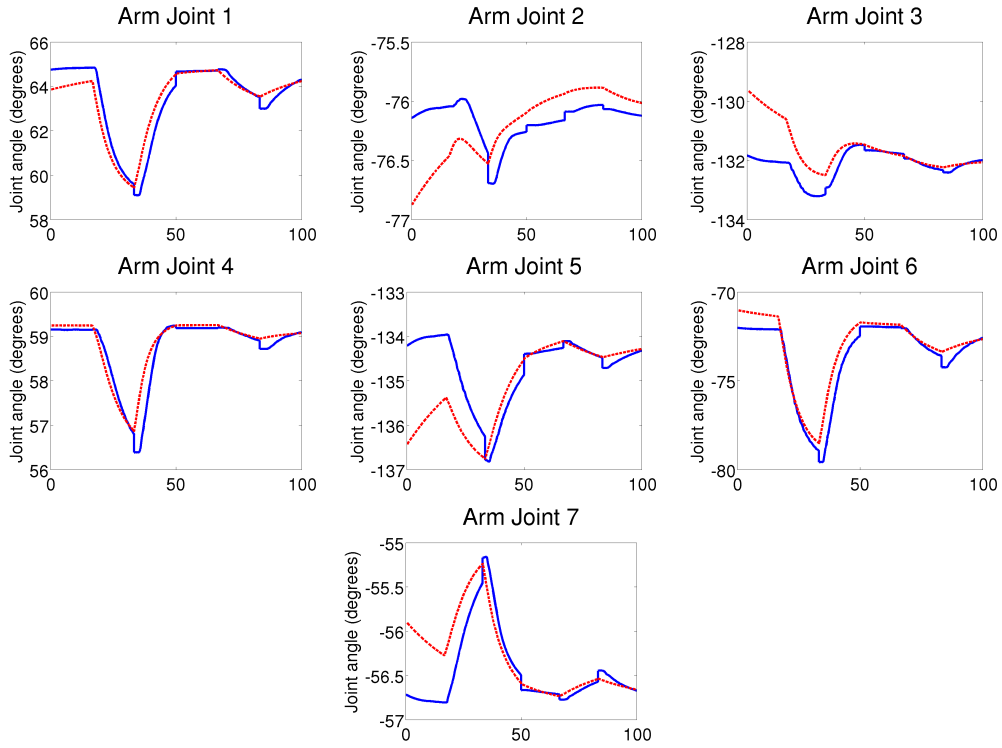


Figure 5.8: Arm joints positions for the first grasping experiment over time in percentage over the experiment duration. Blue lines indicate the readings from the robot and red dotted ones from the simulator.

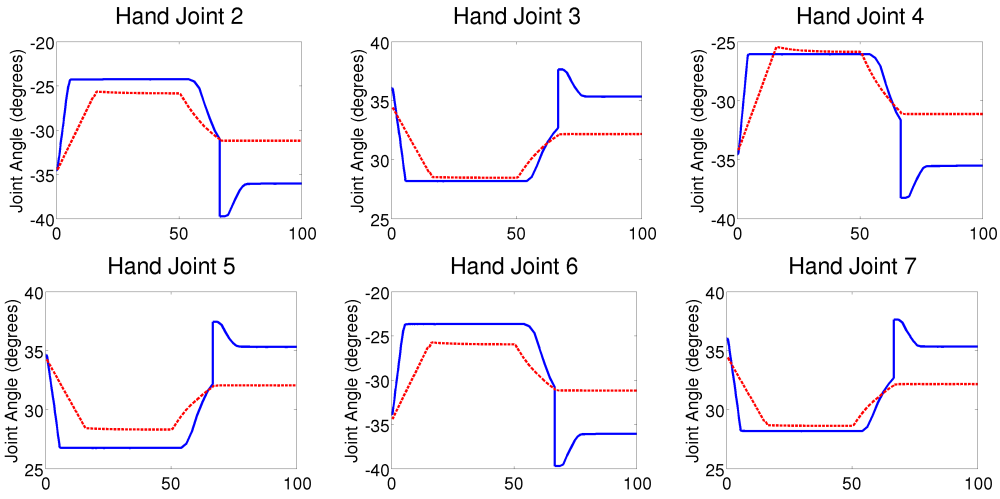


Figure 5.9: Hand joints positions for the first grasping experiment over time in percentage over the experiment duration. Blue lines indicate the readings from the robot and red dotted ones from the simulator. Hand joint 1 is omitted, it did not move and the error was constantly below 0.1 degrees.

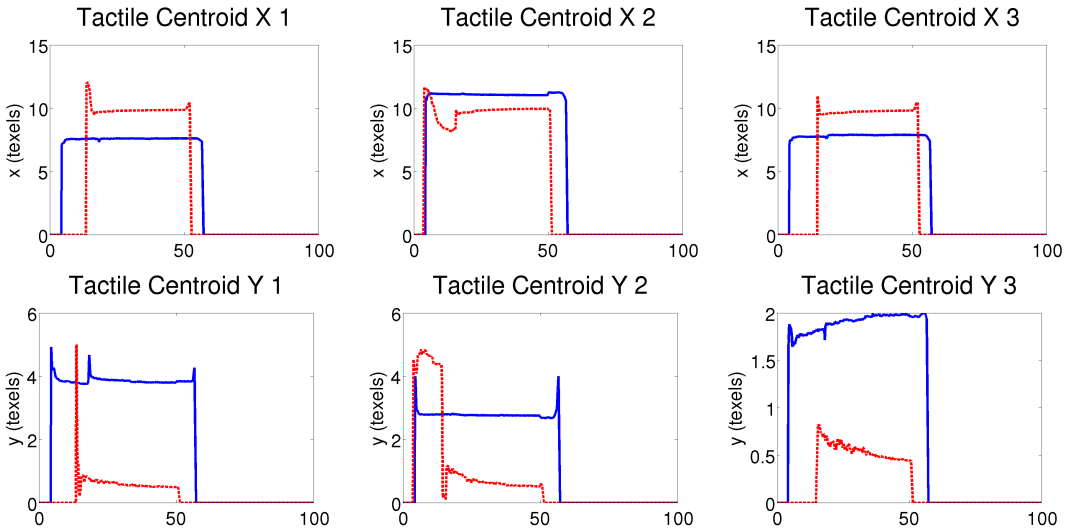


Figure 5.10: Tactile sensor results for the first grasping experiment over time in percentage over the experiment duration. Blue lines indicate the readings from the robot and red dotted ones from the simulator.

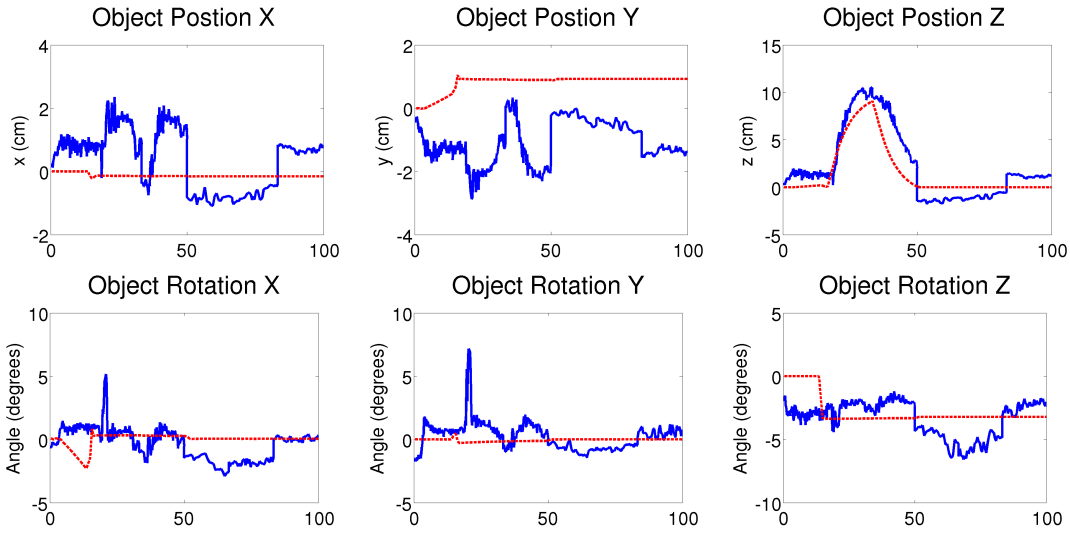


Figure 5.11: Object pose results for the first grasping experiment over time in percentage over the experiment duration. Blue lines indicate the readings from the robot and red dotted ones from the simulator.

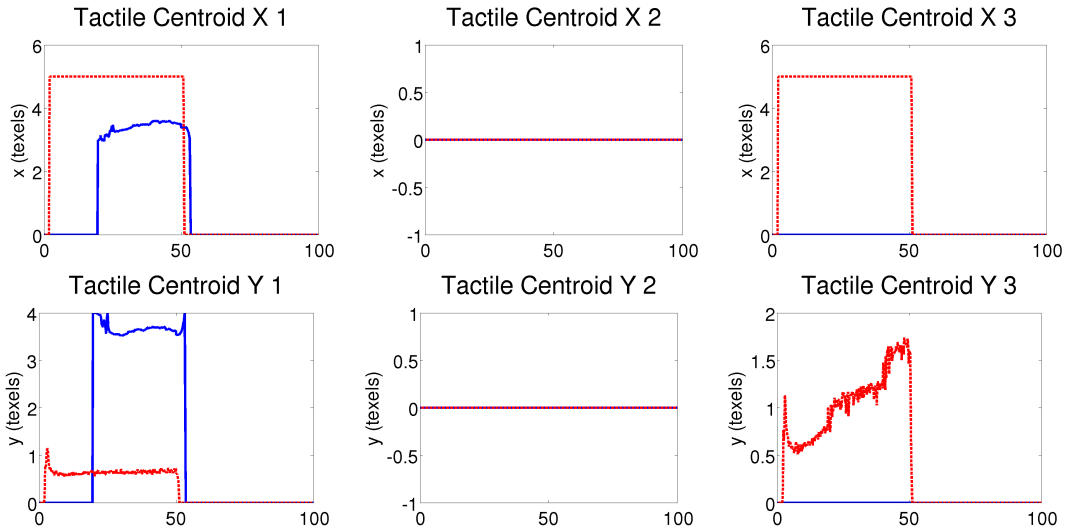


Figure 5.12: Tactile sensor results for the sliding experiment over time in percentage over the experiment duration. Blue lines indicate the readings from the robot and red dotted ones from the simulator.

Slide

The graphs showing the results obtained while sliding the pizza box can be seen in Fig. 5.12 and Fig. 5.13. The arm and hand joints gave similar results for both robots, as in the previous experiments, therefore the results are omitted. The tactile sensors, in the case of the real robot, only show readings for the first sensor given that the hand was slightly tilted towards that finger, then the pressure on the other tactile sensor was not enough to produce feedback. In addition, the real object is not rigid so it deformed when the fingers were in contact with the box surface, an effect that is not modelled with rigid objects in the simulator. In the simulated case both sensors gave similar readings. See Fig. 5.12.

The object position shows a slight variation, see first row from Fig. 5.13. The difference is explained by a lag which results from the simulated and real movements not occurring at the same time. The simulated robot starts ahead of the real robot. Nevertheless, the trajectory of both movements is almost the same. This lag can be explained with the results from the tactile sensors, where it can be seen that the simulated sensor detected the contact ahead of the real sensor which produced the sliding movement to start earlier in the simulation. Regarding the object rotation (see Fig. 5.13 second row), although the box slides, the movement of the object is different. In the simulation, it rotates about 20° around its z axis, while there is no rotation in the real execution. This could be caused because the contacts between the fingers and the box were not aligned

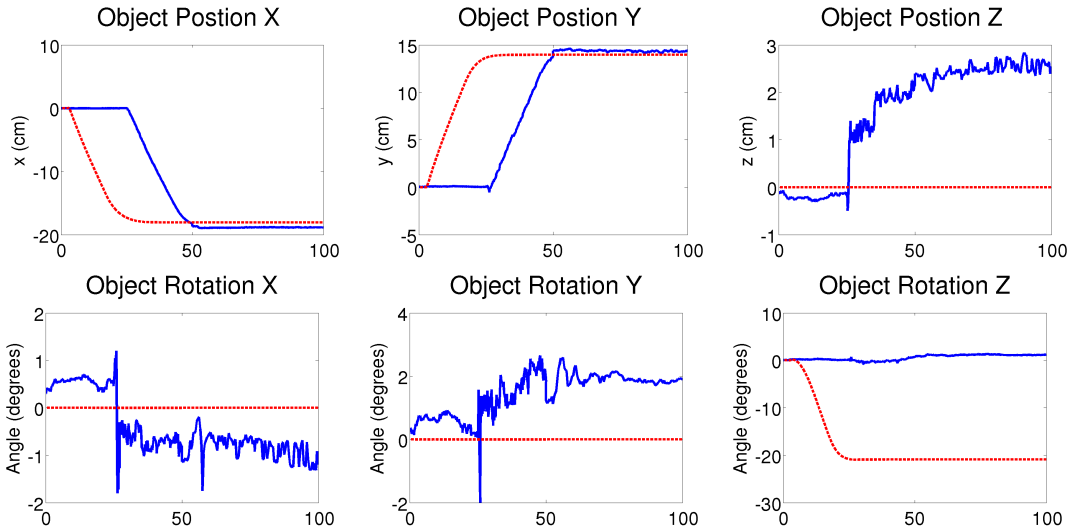


Figure 5.13: Object pose results for the sliding experiment over time in percentage over the experiment duration. Blue lines indicate the readings from the robot and red dotted ones from the simulator.

with the box center of mass, thus when applying the slide movement, also applied an undesired torque that rotates the object.

In-hand manipulation

The results obtained by the in-hand manipulation of the wooden stick are shown in Fig. 5.14 and Fig. 5.15. The arm joints remained static during the experiment and hand joints showed similar behaviour as in the previous experiments, therefore their results are omitted. The tactile readings show different results for the real and simulated case, see Fig. 5.14. The finger 2 tactile sensor hardly shows any reading for both environments, while the other two sensors show more activity in the simulated sensor than in the real one. This can be explained by the difference in the sensitivity of the sensors. This parameter can be modified, with further experiments, to better adjust the readings of the simulated sensor to replicate more accurately the behaviour of the real one. Nevertheless, for this experiment the tactile readings were not used by the controller and the behaviour was not influenced.

The object pose, specially the rotation in x, are the key measurements used to determine how close the simulation is to reality (see Fig. 5.15). The object position shows very different values. In the case of the simulation, it is near zero but for the real object it moves several centimetres. This is due to the difference in the object's reference system. In the real object, the tracker uses a reference frame on an object's corner, whereas in the simulation the object frame is in the center of mass. This is the reason

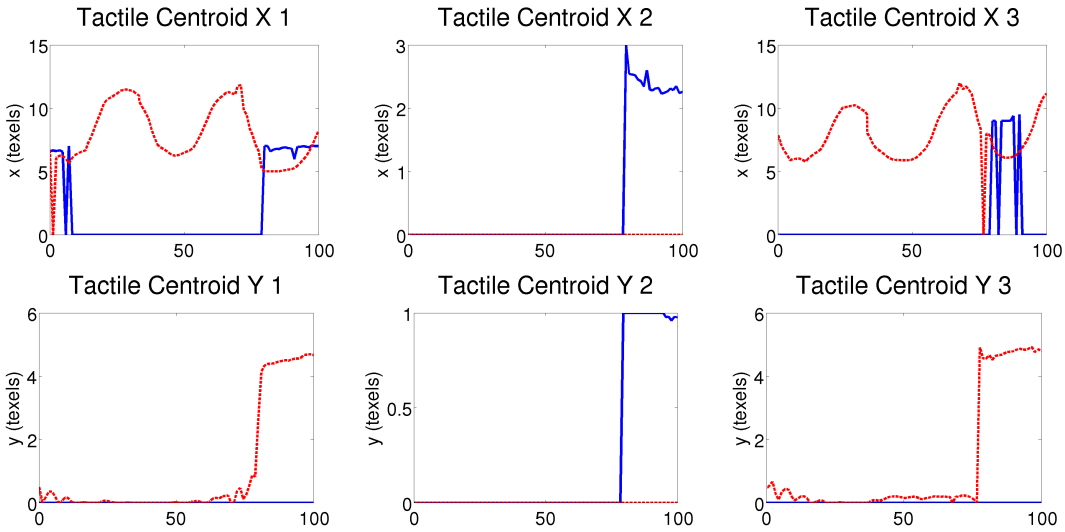


Figure 5.14: Tactile sensor results for the in-hand manipulation experiment over time in percentage over the experiment duration. Blue lines indicate the readings from the robot and red dotted ones from the simulator.

of the oscillation in the z axis in the real environment while it is almost constant in the simulator. Had the object reference frame been the same for the simulated and the real experiments, similar results in the object position would have been obtained.

However, the simulated object rotation in x, which was the most relevant in this experiment, shows a very similar behaviour to that of the real object. In the case of the simulation, the object was undergoing a rotation in y which was not the same as in reality. Nevertheless the overall movement of the object remained very similar to the real one.

5.6.3 Global quantitative results

The errors and standard deviations (for each time step) were calculated for each of the metrics over all the experiments (see Fig. 5.16(a)). The arm joints presented the greatest error in the slide experiment. This is mainly caused by the lag between the real and simulated executions. As the slide primitive only moves towards the target when there is enough force applied to the object (see the slide primitive description in Chapter 3), the different timings in the detection of that target force produced a delayed motion in the simulator that produced the difference in the joint readings.

The errors of the hand joints in Fig. 5.16(b), are in general higher than the arm joint errors but inside an acceptable range of 0.1 radians. The hand-closing controller relies on tactile sensing, thus the detection of contacts is critical for the results to be equal. The different timing that the contacts have shown (see Fig. 5.8 and Fig. 5.12) may

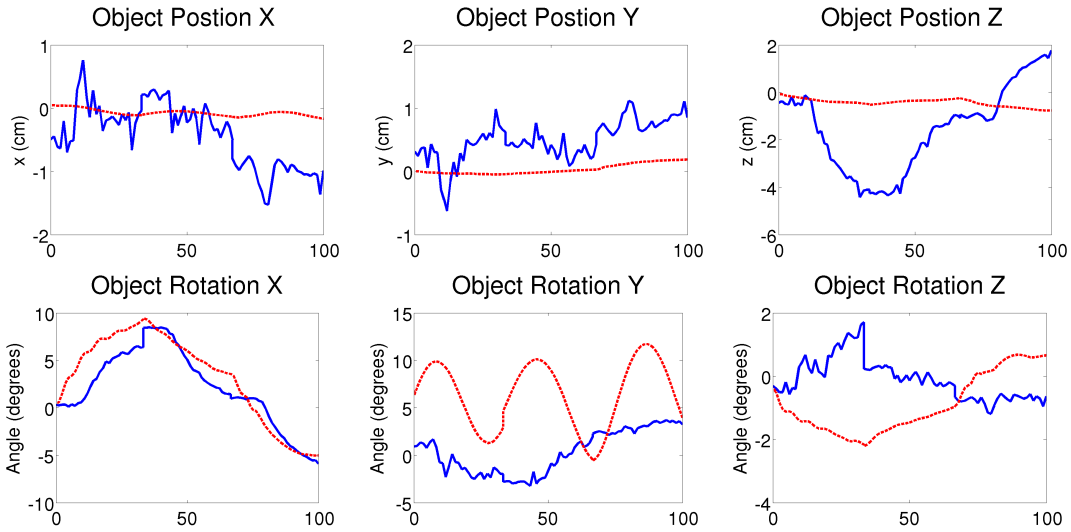


Figure 5.15: Object pose results for the in-hand manipulation experiment over time in percentage over the experiment duration. Blue lines indicate the readings from the robot and red dotted ones from the simulator.

be the main cause for the hand joint error. An special case showed up in the Grasp experiment #2 where the distal joint of finger 1 did not detect contact with the object and continued moving while it was detected in the simulator execution.

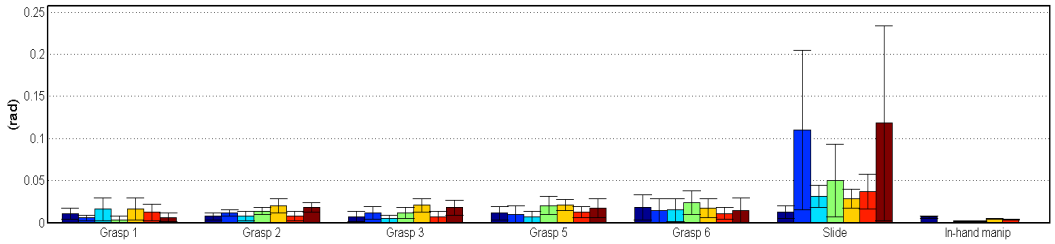
Regarding object position, the error (see dark blue column in Fig. 5.16(c)) is in general below 2.5 cm. While the rotation error, is below 0.05 radians which is very low. The special issues that rise up error, for the slide and in-hand manipulation task, are caused by the specific conditions already explained in Sec. 5.6.2.

Finally, the differences in the tactile sensor readings are depicted in Fig. 5.16(d). The mean error shows that although the tasks were completed successfully there were significant differences in the tactile sensor readings.

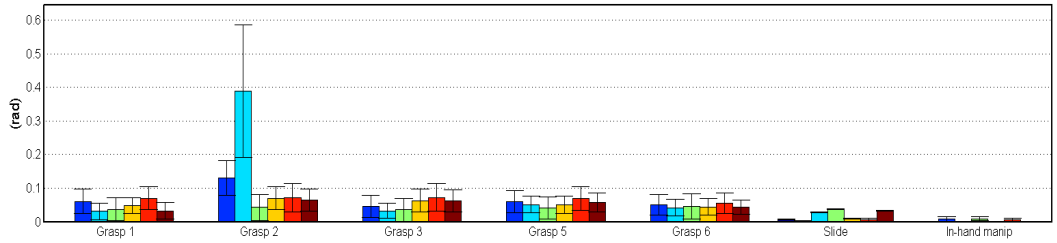
5.7 Simulation as a prediction engine

Once the simulation engine has been implemented and tested, it is possible to use it as a prediction engine. To do so, we propose to integrate the simulator into our manipulation framework. As depicted in Fig. 5.17, the system will be composed of the real robot and the simulated robot executing independently the same task. Using the same scheme used by humans detailed in Chapter 2, the contact events detected by the sensors and the prediction will be compared to detect mismatches.

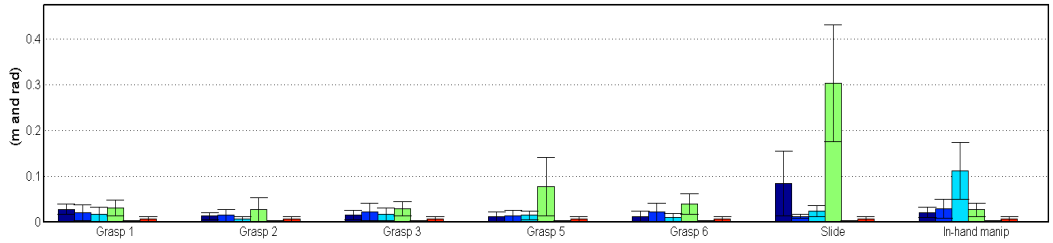
The simulated robot will run an exact copy of the controllers and tasks that are executed on the real robot. The controllers running on both robots will be independent. Thus,



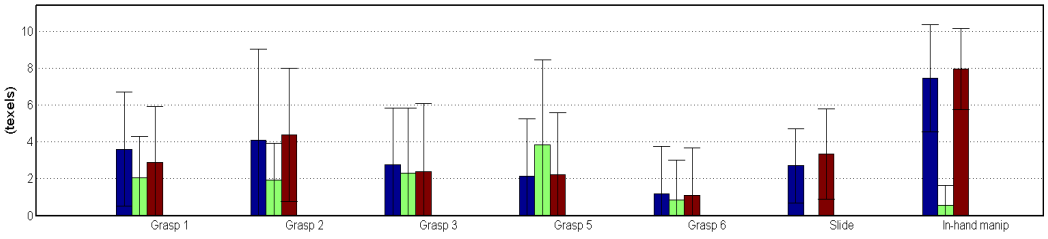
(a) Arm joint errors: mean and stdev for each joint of the arm grouped by experiment. For each group, from left to right: joint A1(dark blue) to joint A7(brown).



(b) Hand joint errors: mean and stdev for each joint of the arm grouped by experiment. For each group, from left to right: joint H2(dark blue) to joint H7(brown). Joint H1 is not represented because it is not used in the experiments.



(c) Object position errors grouped by experiment. For each group, from left to right: Translation error in meters (first column) Rotation error x,y, and z (other three columns) in radians.



(d) Tactile sensor centroid position errors grouped by experiment, position error (texels) of the detected centroid. For each group, from left to right: finger 1, finger 2 and finger 3.

Figure 5.16: Aggregated error for all the experiments performed. Each group of columns represents the results for one of the experiments performed.

on the real robot, the sensor feedback will be provided by the robot sensors and the feedback to the simulated robot controllers will be provided by the simulator.

Both robots will be also independently running the contact perception framework presented in Chapter 4 and generating contact events. The real contact events generated by the real execution will be compared with the predicted contact events obtained from the simulation. Mismatches will trigger corrective actions on the real controllers.

After the corrections are performed, the task continues running and the current perceived state of the robot and the environment is sent to the simulator so it can synchronize, update the internal models and continue providing predictions.

5.8 Conclusions

It is unknown how humans predict contact events and the result of their actions. The human internal representations of objects, environment, action and interactions are still undiscovered. However, it is accepted that the physics laws are not directly encoded in the brain and the representation used by humans has to be based on past experiences. In an effort to use a different approach for prediction, instead of using a physics engine, [Kopicki et al., 2011] used learning techniques to encode the effects of physical interaction on objects. However, this approach does not generalize well, produces implausible results and requires manual tuning of parameters that are object specific. This method was improved by [Belter et al., 2014], they combine physics engines with the learning approach to force physically plausible predictions. As the authors claim, this method provides fairly good results useful for prediction. However, it still requires to train one predictor for each object. A possible next step for this methods is to train category-based predictors and switch the predictor used depending on the object category detected.

A different approach for prediction, instead of predicting the arm and object motions is to predict the sensorimotor events that should be detected during the task execution. As proposed by [Pastor et al., 2011], sensorimotor memories acquired from past experiences can be used for this purpose. In this case, the generalization of the experiences to different objects and tasks is an open issue. Moreover, it requires previous successful experiences for training.

For the manipulation system presented in this thesis it would be enough to predict the contact events, which can be obtained from the sensorimotor memories approach. However, humans do also predict trajectories and the results of their actions on objects. Hence, it is also important to have a prediction engine that can provide that kind of data.

Regarding contact event prediction, physics simulation and sensormotor memories could be combined. First, physics simulation parameters can be tuned by the execution of ex-

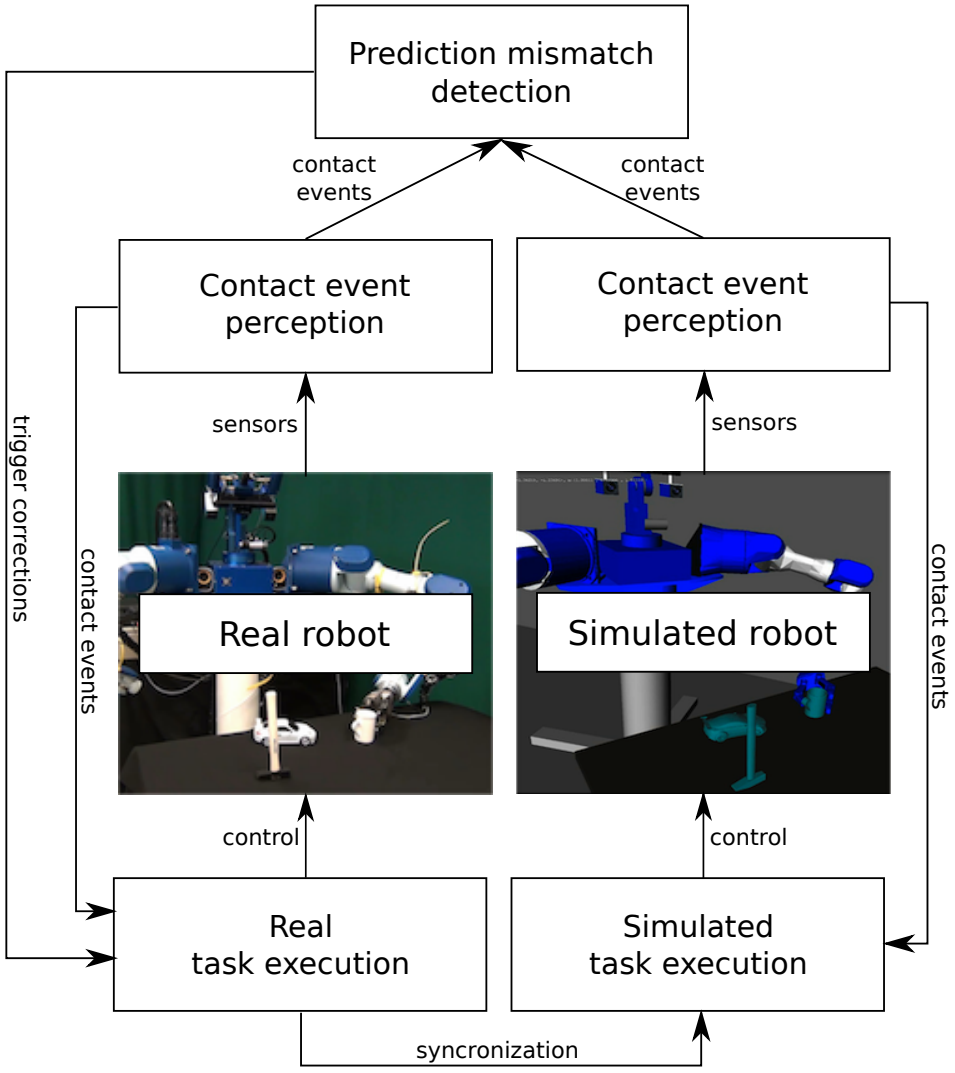


Figure 5.17: Integration of the simulator as a prediction engine into the presented framework. The real task and the simulated are running at the same time, the predicted contact events (obtained from the simulation) are compared with the contact events perceived from the real robot. The mismatches trigger corrective actions in the real controllers, when the correction is finished, the simulator is synchronized with the reality and the prediction process continues.

ploratory interactions with the object. Second, the simulation can be used to provide training data. Finally, the learned sensorimotor memory can be used to drive the execution of tasks and trigger corrections when mismatches are detected.

Considering object motion prediction, the same exploratory interactions can be used to train the system provided by [Belter et al., 2014] which provides more accurate and faster results for this scenarios than standalone physics simulation.

When dealing with novel objects, the robot needs to build an internal representation of the object, that can be bootstrapped from a category-based generic representation or using a similar known object representation. The detected prediction mismatches have to trigger model updates. The physics engine approach can be easily adapted, on the other hand, the learning approaches have to learn again the interactions with the object.

In this chapter, we have presented a complete dynamics simulation of a humanoid torso robot. Moreover we have evaluated the similarity of the simulation to the real behaviour of manipulation tasks, by using the same controller on the real and the simulated platforms. Then, we have discussed and analysed the differences. The results have shown that it is possible to simulate manipulation tasks with the current state of the art of simulation tools. Although the precision is not perfect, the simulator is able to perform manipulation tasks using the same controller used in the real world with very similar results. Therefore, an important result is that with the proposed prediction system, it is possible to use simulated sensor data in manipulation task controllers that use sensor feedback. On the other hand, perfect simulation is still far from being a reality, and the physics engines still need to evolve more to provide near-perfect real-time simulation results.

Finally we have proposed the architecture for the addition of the prediction engine into the contact based manipulation framework. However, the implementation of the contact event comparison, the corrections triggered by contact event mismatches and the integration of the simulator into the framework are still not developed and part of the future work.

The implementation efforts and the experimental results shown in this chapter are already published in [Leon et al., 2012].

Chapter 6

System integration

Endowing a humanoid torso with autonomous manipulation abilities involves different research topics and requires the integration of many components. All the different building blocks of the system identified in Chapter 2 and implemented in Chapters 3, 4 and 5 need to work together in a coordinated fashion. Hence, it is necessary to provide an integration architecture that allows the modules to communicate to each other. It is also important that the proposed solution supports the paradigm of manipulation primitives and task descriptions introduced in Chapter 3. Finally, it has to be general, such that the integration and implementation of other research projects is possible.

A general approach to integrate different components into a system is the use of middlewares (e.g. ROS, Yet Another Robot Platform (YARP)). However, using a well known middleware and implementing the system components as modules, does not satisfy all the requirements. In big projects with many modules like the one presented in this thesis, structureless implementation and heterogeneous inputs and outputs results in chaos. Therefore, to ease the code maintenance and the integration of new modules, an integration architecture that defines a common structure, concepts, coding style and documentation is required.

In this chapter we describe the software architecture developed to integrate all the components of the contact-based manipulation system, it consists of a modular four layered architecture with three different data flows. The different layers and data flows are designed to support natively the manipulation primitives paradigm and the definition of tasks as introduced in Chapter 3.

Moreover, there are several built-in abilities (e.g. inverse kinematics, object detection) that are supposed to be available by the manipulation primitives, contact perception or contact prediction. Those abilities are also presented in this chapter. Finally, the portability of the architecture is addressed and its implementation on another robotic platform is shown.

6.1 Software architecture

To control all the modules within an integrated platform, we have implemented a layered system that enables us to interact with the robot at different abstraction levels (see Fig. 6.1). The core of the architecture is formed by three layers: service, primitive and task. These core layers communicate with the simulator and the robot hardware through the interface layer. There are other secondary components such as GUI, robot models and databases. Each layer is composed by modules that run in parallel and communicate with each other using three main types of messages. The *data messages* are the input/output of the modules. They contain any type of information, raw or processed (e.g. joint status, camera image, object positions). The *control messages* change the parameters (e.g. threshold, loop rate) and the status (e.g. run, stop) of the modules, all the modules accept by default the commands run, stop or reset. Finally, the *event messages* contain information about a detected situation (e.g. an object is localized or grasped successfully, motion detected).

ROS is used to handle the messages between the different system parts. ROS is a middleware that provides a message passing framework among other features. Typically, a ROS-based system is composed of multiple ROS nodes. A ROS node is a process that can send and receive data using two different methods: *ROS topics* for asynchronous n-to-n data streams and *ROS services* that provide a request/response 1-to-1 mechanism.

Each module belongs to one of the four layers depending on its purpose and its implementation. Hardware interfaces and drivers belong to the *interface* layer. Above them, modules that perform basic operations such as sensor processing or motor control, belong to the *service* layer. Mid-level modules like manipulation primitives (grasp, push, etc.) or perceptual primitives (locate object, wait for event, etc.) are classified into the *primitive* layer. Primitives can rely on services to accomplish their goals. More complex modules that require several primitives, services or other tasks to work, are classified into the *task* layer (e.g. pick and place). Depending on the layer where a module belongs to, it must follow specific implementation rules.

6.1.1 Robot and simulator interfaces

The robot interface layer converts the messages back and forth from hardware drivers to the ROS messages used by our system. Exactly as for the real hardware interface, we have implemented interfaces for the OpenRAVE/OpenGRASP [León et al., 2010] simulator and Gazebo [Koenig and Howard, 2004]. The simulator interfaces provide the same inputs and outputs than the hardware interfaces. This allows the upper levels to work without knowing whether they are controlling the real or the simulated robot. As detailed in Chapter 5, the simulator is also used as a prediction engine.

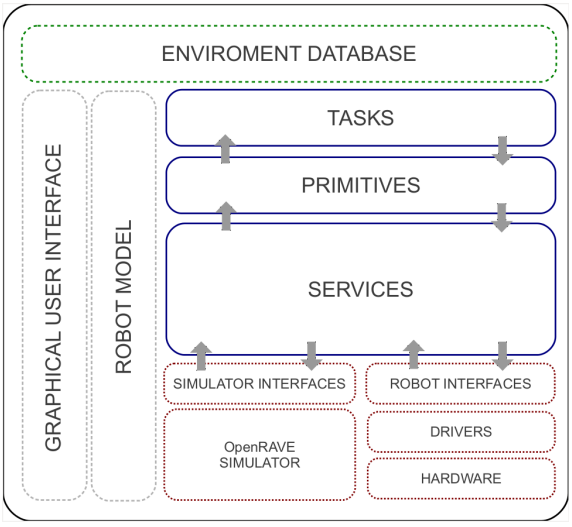


Figure 6.1: Integration software diagram.

6.1.2 Services

The *services* are basic processes that receive an input and provide an output. Thus, services are mostly used to obtain high level information from raw sensor data and to send control data to the interface layer. The purpose of the service layer is to provide the building blocks for higher layers. Services are characterized by the following properties: i) they are continuous non blocking loops that never stop by themselves, ii) they generate at least one output and require zero or more inputs, but they do not generate control messages.

6.1.3 Primitives

The *primitives* define higher level actions or processes. A primitive is a control loop that gets processed data from services, generates events and sends commands to the underlying services. They are defined by the following properties: i) They are continuous non blocking loops that never stop by themselves. ii) They generate at least one output and can have but do not require inputs. They also generate events towards the task layer. The role of the primitives is to use and manage services to control the robot (grasp, move, transport or look at), perceive (recognize or localize objects) and detect special situations (motion detected, object lost).

6.1.4 Tasks

Tasks represent the highest level of our architecture and use primitives as building blocks to generate desired behaviours. A task (as defined in Chapter 3) can be described as a cyclic, directed, connected and labelled multi-graph, where the nodes are primitives

and the arcs are events. An example of a task description that performs active object tracking with head movements is depicted in Fig. 6.2. It is used in Fig. 6.3 together with a manipulation task to compose a task that grabs an object while looking at it. Tasks have the following properties: i) They can end, do not need to be continuous. ii) They can have multiple inputs and outputs. iii) They generate events.

The role of a task is to coordinate the execution of the primitives, by generating control messages and changing the data flow between primitives and services. Multiple tasks can run in parallel and can also be coordinated by other tasks.

Command message Messages sent to the modules to change the configuration parameters or change their execution status. The default commands for each module are: run, stop and reset.

Data message Generic message used to send data from one module to another module.

Event message Message that is generated by the detection of a specific condition or event (e.g contact detected, object lost, object detected).

Module generic component of the system. Is composed by a ROS node with defined inputs and outputs (ROS topics and services). Its constraints and default functionalities depend on the layer it belongs to.

Service basic processes that receive an input and provide an output. Are continuous non blocking processes that never stop by themselves. Generate at least one output and require zero or more inputs. Do not generate control messages. (e.g. filters or joint controllers)

Primitive define higher level actions or processes. A primitive is a control loop that gets processed data from services, generates events and sends commands to the service layer (e.g. grasp, transport).

Task represent the highest level modules. They use primitives as building blocks to generate the desired behaviours. A task (as defined in Chapter 3) can be described as a cyclic, directed, connected and labelled multi-graph, where the nodes are primitives and the arcs are events (e.g. clear the table).

6.2 Implementation and task setup

6.2.1 Module implementation tool

The software architecture does not only settle the concepts but also provides a tool to automatically generate code. It follows the coding style and guides the developer through the process of creating a new service, primitive or task. A module can be described by its inputs, outputs, parameters and events. For the creation of a new

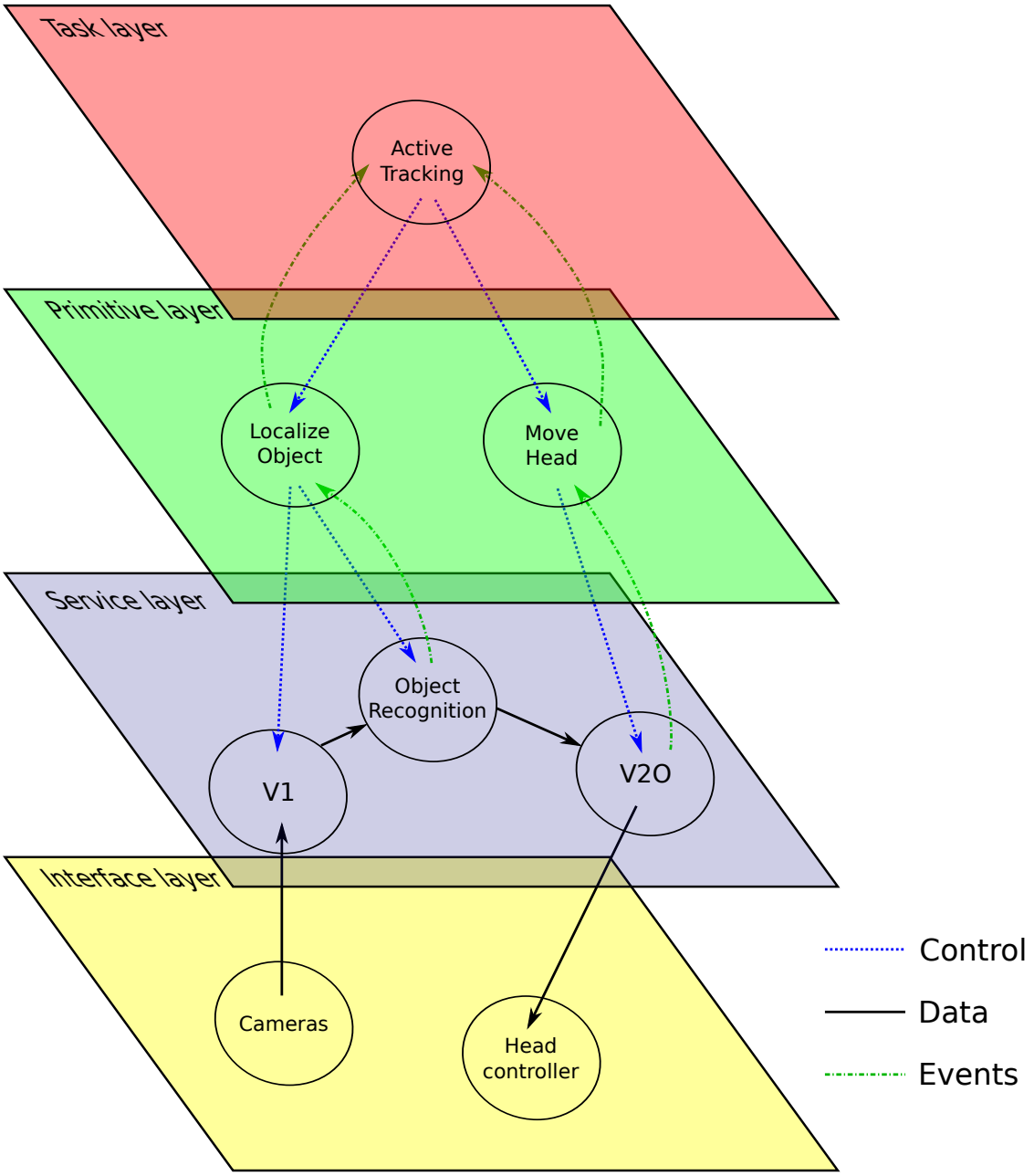


Figure 6.2: Active tracking task. It consists of two primitives, *Localize Object* and *Move Head*. The former employs two services to extract visual features and localize the target object. The latter contains a service that calculates the eye movement required to gaze at the target. Both primitives trigger an event when their state changes.

system module (service, primitive or task), we provide an interpreter that uses the XML description of the module and creates doxygen friendly code stubs with all the communication and parameter handling. It also takes care of configuration files, cmake files and launchfiles. Advantages of the interpreter are: 1) the developers do not have to waste time creating the code stubs, configuration files and cmake files, which is a mechanical task that can be automated. 2) All the modules have a similar structure making the code standard and easily understandable by other developers, the XML description is human readable and provides detailed module information at a glance. 3) Automatic code generation eliminates the possible errors introduced in those parts of the code.

6.2.2 Configuration and parameter setting

Each module XML description contains the parameters, inputs and outputs of the module. The configuration (inputs, outputs and parameters) of each module can be statically set in a YAML Ain't Markup Language (YAML) configuration file. Moreover, the module configuration can also be modified online as the task is being executed using command messages. To configure a module to use it in an experiment, it is mandatory to define the connections of its inputs and outputs. Although the modules provide default values for their parameters, it is important to set up them accordingly to the task.

6.2.3 Task example

A simple task devoted to recognize and actively track an object is depicted in Fig. 6.2. Assuming that all the modules required are implemented, the configuration files for each module should be provided. The configuration files contain, for each module, the parameter values and the input and output connections. For example, the YAML file for the *Object Recognition* service contains the input connected to the *V1* service, the output connected to the *V2O* service and the parameter with the object id. The primitives and tasks are configured using the same process.

More complex experiments using this architecture have been presented in Chapter 3 where the robot is able to empty a box full of objects and grasp a bottle with one hand and unscrew its cap with the other hand. Another experiment using a different robotic platform is detailed in Sec. 6.5 where the task implemented to solve the Amazon Picking Challenge is presented.

6.3 Implemented modules

A set of services can be connected in series, connecting the output to the next service input. This kind of configuration, where the raw input is processed by several services, is called pipeline. Pipelines are used to implement complex processes as a series of

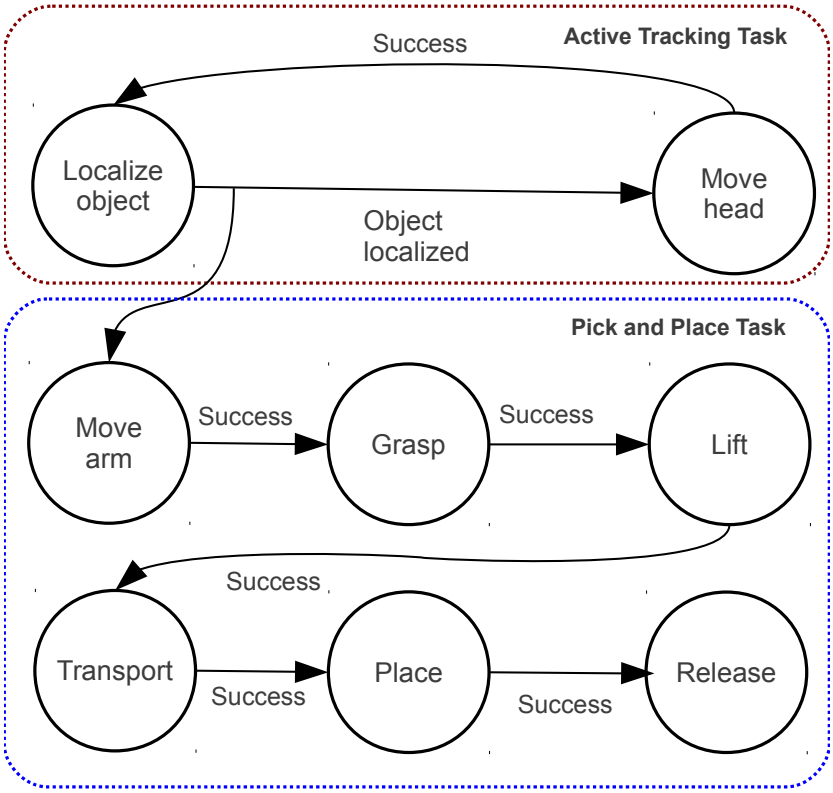


Figure 6.3: Cooperation among tasks. The robot executes a task which consists of grasping and moving the target object (*pick and place*), that requires six primitives. Meanwhile, the robot actively tracks the moved object.

simple operations. An example of a pipeline is detailed in this section where the visual processing pipeline and the grasp planning pipeline are outlined.

This section describes the basic modules and pipelines implemented on the robot either as services or primitives. They are used together with other primitives and services, to create new tasks.

6.3.1 Joint controller services

The low level control is performed by three services that interface the architecture with the robot or simulator interfaces. Since each joint can receive different control signals concurrently from different controllers, the *joint state subscriber continuous* combines the control inputs, using a configurable weight, and sends the result to the interfaces at a fixed frequency of 200Hz. *Joint state publisher continuous* performs the inverse task: it gathers the information from different hardware interfaces, combines them and provides the information of all the joints of the robot in a single message at 200Hz. Using the robot model and the joint values, the ROS tool *robot state publisher* provides the transformation tree for the entire robot using the TF library[1]. TF is a ROS library that provides tools for geometric transformations of different data types among different reference frames. The TF tree is the representation of the reference frames available on the system, the tree for Tombatossals is depicted in Fig. 6.4.

6.3.2 Arm controllers and planning

The arm control is composed of several services grouped into two pipelines. The inverse kinematics pipeline converts the end effector target pose, velocity or wrench specified in the Cartesian space (w.r.t. any known frame), to robot joint velocities. The planning pipeline converts the target pose into a plan in joint space and executes it.

The *inverse kinematics pipeline* is composed of two services. First, the *Cartesian controller* is in charge of converting any target input (position, velocity or wrench) into a Cartesian velocity command w.r.t. robot_base using the TF library (see Fig. 6.4). Second, the *ik solver* transforms a Cartesian velocity command w.r.t. robot_base into joint velocities. It is an inverse kinematics iterative solver based on the pseudo inverse of the Jacobian matrix. The solver is provided by the KDL library [Smits, 2015] and requires the robot model, the kinematic chain and the controlled frame.

The *planning pipeline* is a set of services that interface with the MoveIt! [Sucan and Chitta, 2015] framework. The first service of the pipeline is the *ros moveit interface* that converts the input target pose into a MoveIt! message and forwards it to the planning pipeline. The planning request is then received by the *move group* service, the core service of the pipeline. This service uses the robot description, the environment description and the depth sensor input to obtain a collision free plan from the current

[1]See `http://wiki.ros.org/robot_state_publisher`

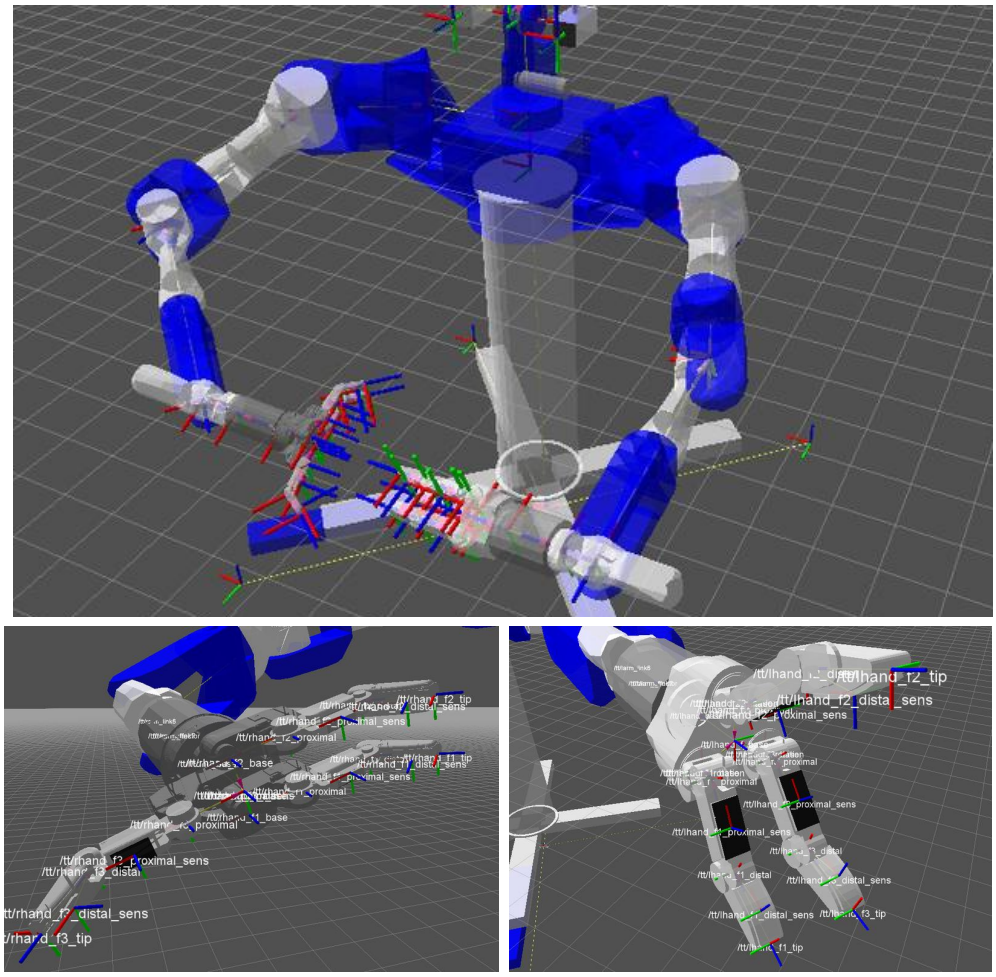


Figure 6.4: Visualization of the TF tree for Tombatossals. Top: full robot. Bottom left: right hand. Bottom right: left hand.

arm pose to the target pose. To obtain the plan, the *move group* service uses one of the algorithms available in Open Motion Planning Library (OMPL). The algorithm is selected and configured through the *move group* service commands. The result of the planning process is a trajectory composed of a waypoint list. Each waypoint contains a target joint configuration of the robot. Finally, the planned trajectory is received and executed by the *robot trajectory* service that moves the robot joints from waypoint to waypoint. While moving, the force and RGBD sensors are monitored to watch out for unexpected collisions. If a collision is detected the execution is aborted.

The target inputs can be specified on any known reference frame connected to the robot TF tree. This allows the upper layers to set up a task frame and specify their commands in the task frame without the need of performing conversions to the robot frame. Using a task frame is very convenient to specify positions, velocities and forces. It is easier to specify the rotation of a door knob in the door knob frame than specify the same rotation on an arbitrary frame not aligned with the door knob rotation axis.

6.3.3 Visual perception pipelines

The visual system of the robot consists of a Kinect sensor and two cameras (more details about the visual sensors are provided in Appendix A.1). Several services that exploit common libraries such as PCL [Rusu and Cousins, 2011] or Opencv [Bradski and Kaehler, 2008] have been implemented to process the visual information.

3D pipeline

The 3D visual pipeline (Fig. 6.5), processes the cue from the RGBD camera and provides a segmented point cloud for each object in the Region Of Interest (ROI). The first service of this pipeline is the *plane detector*. This service detects the principal plane of the scene using RANSAC. Based on that plane, the ROI is calculated extruding the bounding box of the points that belong to the principal plane. As the principal plane is not expected to change fast, this service runs at 1Hz to save computational power. The *ROI filter service* receives two inputs: a ROI (defined by 8 vertices) and a point cloud. Then it filters out the points outside the ROI. Then the *self filter service* is applied, this module uses a spherical model of the robot in conjunction with the current robot joint positions to remove the points in the point cloud that are part of the robot (the spherical model of the robot is detailed in Appendix B). After this process, the point cloud only has points that belong to objects in the workspace. Finally, the remaining points are processed by the *cluster extractor service*, which performs an euclidean clustering on the input point cloud and publishes serially each cluster in a separated point cloud. On our current setup, the pipeline is able to provide the extracted clusters at 30Hz. However, the computational complexity of the clustering algorithm depends on the number of points to be processed, we consistently obtained a 30Hz rate but that performance might decay for larger objects with more points.

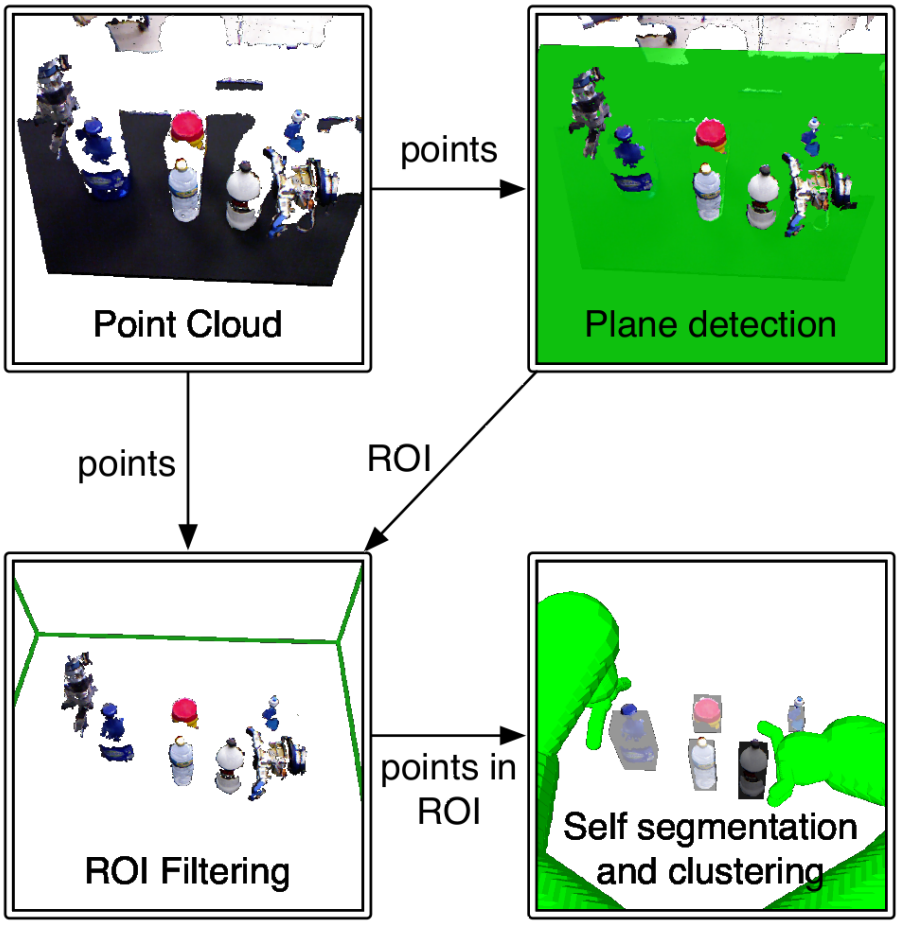


Figure 6.5: The 3D vision pipeline for object segmentation. The point cloud is processed by the plane detector and the principal plane is obtained. The region of interest (ROI) is calculated extruding the bounding box of the points that belong to the principal plane and sent to the ROI filter service. The ROI filter service processes the input point cloud and applies the ROI obtained from the plane detector. From the resulting points the self-filter service removes the points that belong to the robot, then a clustering algorithm detects the objects.

All the features in this pipeline are implemented using the algorithms and filters available in the PCL library [Rusu and Cousins, 2011]. The pose of the RGBD camera is known in advance. To calibrate it, an AR marker was attached to a known frame of the robot and calculated the camera-robot transform. This calibration allows us to add the RGBD camera frame to the TF tree and the visual system can output the data w.r.t. robot base if necessary.

6.3.4 Contact perception pipeline

There are several sensors able to detect physical interaction with the environment. Although each sensor can be accessed independently from either services or primitives, the contact perception services implement a sensor fusion approach combining all the available sensors to produce contact information which can be used by other services or primitives. The *contact_hypothesis_fuser* service provides the output of the combination of all the contact hypotheses generated by the other services of the contact perception pipeline. The details of the fusion process and contact hypothesis generation from tactile, force, vision, simulation and context are already provided in Chapter 4.

6.3.5 Grasp synthesis pipeline

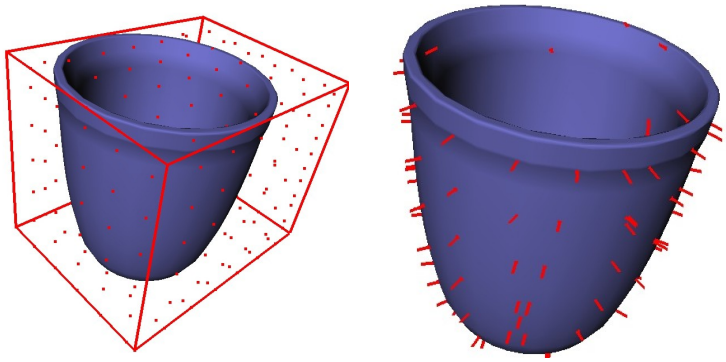
In this thesis we use sensor-based reactive approaches for the implementation of manipulation skills, thus the grasp planning algorithms used do not provide exact contact points but Approach Vectors (AVs). AVs determine the starting position for object manipulation strategies. There are two implemented methods to generate AVs: the *simple grasp generator* based on the perceived object geometry and the *openrave grasp generator* based on the object model.

The *simple grasp generator* implements a method inspired by [Rombokas et al., 2012], that performs PCA on the object's segmented point cloud to generate approach vectors that are parallel to its principal axes. The position is determined by the intersection of each principal axis and the bounding box of the perceived object geometry.

The *openrave grasp generator* uses the object model (6D registered or reconstructed) and the robot model in the OpenRAVE grasp generator to produce the AVs. First, the surface of the object's bounding box is uniformly sampled. Second, the intersection of the object and a ray originating from each sampled point going inward is taken. Finally, the normal of the object's surface from each of these intersection points is taken to be the approaching direction of the end-effector, see Fig. 6.6. More details about the OpenRAVE AV generator can be found in the OpenRAVE online documentation[2].

To select a single approach vector or reduce the search space for the final AVs selection, the generated AVs can be filtered using different services. The *reachability filter service* filters out the AVs that are not reachable by the selected robot arm. The *Collision-free*

[2] `http://openrave.org/docs/0.8.2/openravepy/databases.grasping/`



(a) The surface of the bounding box of the object is uniformly sampled.

(b) The intersection of the object and a ray originating from each sampled point going inward is taken.

Figure 6.6: Example of the approach vectors generated by the OpenRAVE grasp planning plugin. Images obtained from the OpenRAVE online documentation.

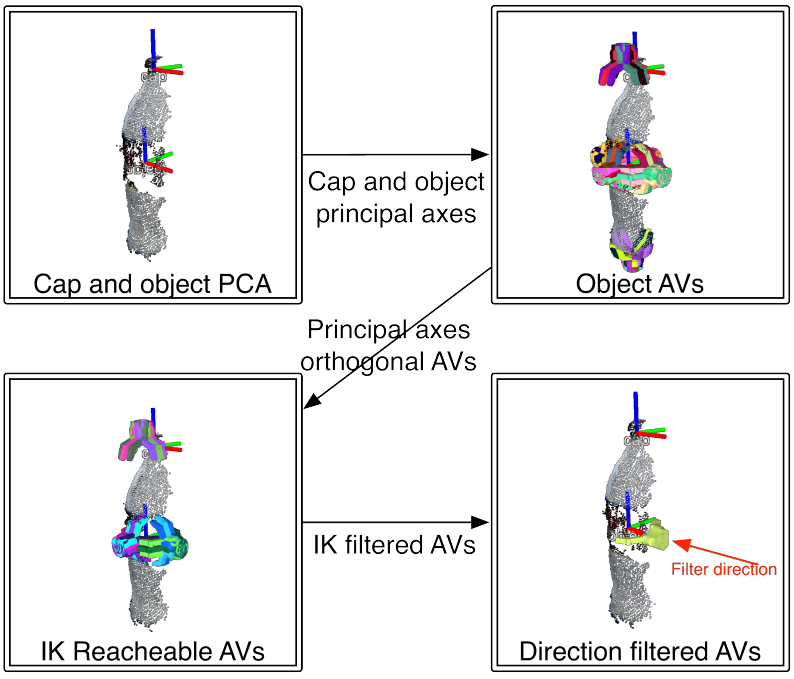


Figure 6.7: Example of the approach vector pipeline using the simple grasp generator on a bottle finally constrained to a side grasp. Each generated approach vector is represented by a CAD model of the hand.

filter service removes the AVs that are in collision with the environment, the object or the robot itself. The *orientation filter service* deletes the AVs that are not pointing to a determined direction ± a threshold. The *distance filter service* filters out the AVs that are further than a threshold from a configurable frame. An example of the AVs filtering process is depicted in Fig. 6.7.

6.3.6 Manipulation primitives

As detailed in Chapter 3, manipulation primitives are single reactive controllers designed to perform a specific primitive action on a particular embodiment. The primitives are configurable to adapt their behaviour to a specific task or environment. The primitives available on the system are briefly described below, a more detailed description with all the available parameters can be found in Chapter 3.

- Move arm to: can either use the planning or inverse kinematics services to move the selected arm to the desired position. Monitors the motion and stops if a collision is detected.
- Robust grasp: performs a reactive sensor-based grasp strategy with the selected arm and fingers.
- Hold: controls the fingers of the hand to keep the desired force.
- Lift: moves the arm upwards (w.r.t. world) until the desired lift distance is reached.
- Place: moves the arm towards the target supporting surface until contact is detected.
- Release: opens the hand to the desired opening position. Meanwhile zero-force control is used for a compliant release movement.
- Slide: slides the object over the surface towards the target position controlling the applied force.
- Push and pull: pushes or pulls an object compliantly for the desired distance or until the force limit is detected.
- Unscrew: grabs a threaded cap and performs the required twist movements pulling the cap until it is free for removal. While the cap is being unscrewed, the object needs to be fixed (e.g. by the other hand).
- Touch: moves the arm forward until a contact is detected.

Beyond manipulation primitives, there are other primitives needed to compose a task. Those auxiliary primitives can activate or deactivate perception pipelines, change parameters, introduce delays or wait for key strokes.

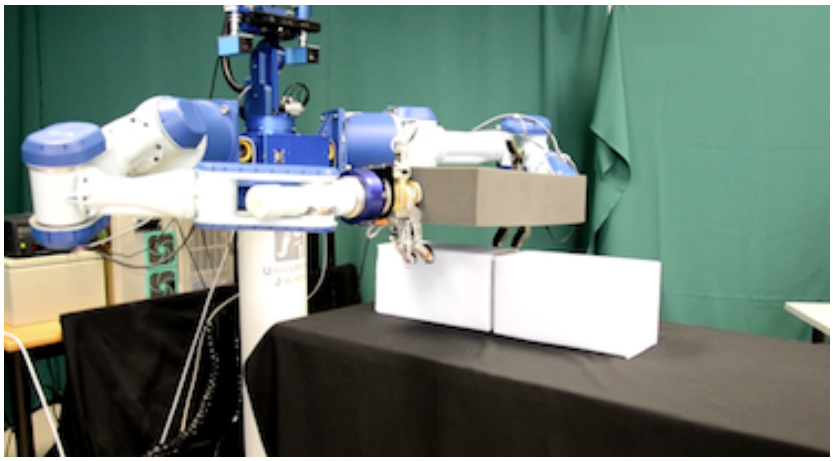


Figure 6.8: Dual arm manipulation of big objects.

6.4 Other research experiments

In addition to the experiments detailed in previous chapters, regarding object manipulation, contact perception, contact prediction and object detection and recognition, the architecture presented in this chapter was also used to perform other manipulation experiments. Dual arm control is another of the ongoing research lines that have been conducted with our robot. As depicted in Fig.6.8, some preliminary results have shown dual arm coordination abilities for manipulating big objects with both arms.

Besides grasping and manipulation experiments, the head of the robot has also been used for research experiments about biologically inspired learning. Work about implicit mapping of the peripersonal space can be found in [Antonelli et al., 2011] and more results of saccadic adaptation and learning saccade control are available in [Chinellato et al., 2012, Antonelli et al., 2013, Antonelli et al., 2015].

6.5 Implementation on other embodiments

Theoretically, the implementation of the layered architecture allows the software components to be used on different hardware platforms. The only modules that need to be implemented are the drivers and hardware interfaces. Thus, services, primitives and tasks can be used independently of the underlying hardware. However, this is not frequently the case, and services, primitives and even tasks are usually platform dependent and require modifications in order to adapt to other platforms. Although it is possible to implement services and primitives in a hardware agnostic manner, each platform has its specific features and not exploiting them would cripple the capabilities of the robot.

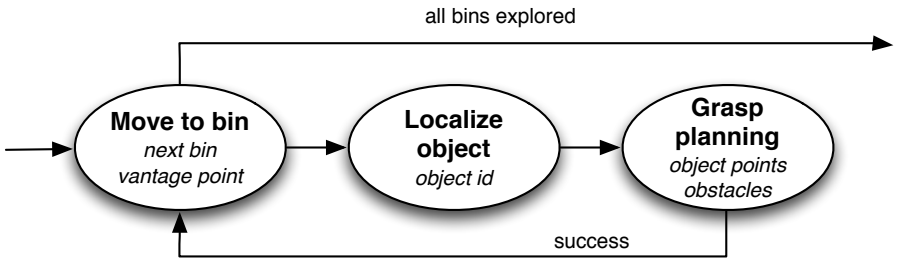


Figure 6.9: Exploration task definition for the APC. Parameters required by each primitive are written in italics. The target bin is switched by the task each time the Grasp Planning primitive finishes and before the Move to bin primitive is activated again.

All the modules presented in this chapter were implemented for Tomabtossals. Moreover, the four building blocks presented through this thesis to build a complete contact-based manipulation system were mainly implemented for the same platform. In order to participate in the APC (Amazon Picking Challenge), some of the modules developed for Tombatossals had to be ported to Baxter (see a detailed description of the robot in Appendix A.4), which was the robotic platform to be used during the competition. The aim of the challenge is to pick all the objects in an order list from a shelf and place them into a bin next to the robot. The robot is given a file with the target objects and it has to autonomously search, pick and place them into the bin.

To solve the APC task, we divided the problem into two sub-tasks: exploration and manipulation. The former uses the Kinect attached to the robot forearm to explore the shelf, look for the target objects and generate an approach vector to grasp each target object (see Fig. 6.11). The latter uses the list of plans generated by the exploration subtask to sequentially grasp the target objects and place them in the bin. Both subtasks are described using the manipulation primitives paradigm and depicted in Fig 6.9 and Fig 6.10.

The modularity and structure introduced by the software architecture made the adaptation of the services and primitives easier. On the other hand, the task, some primitives and services were implemented ad-hoc for the challenge. The visual and grasp planning process is depicted in Fig. 6.11, where the segmentation, clustering, recognition and grasp planning steps are depicted. Finally, Fig. 6.12 shows a picture of the robot grasping an object from the APC shelf and the result of the visual and grasp planning pipelines.

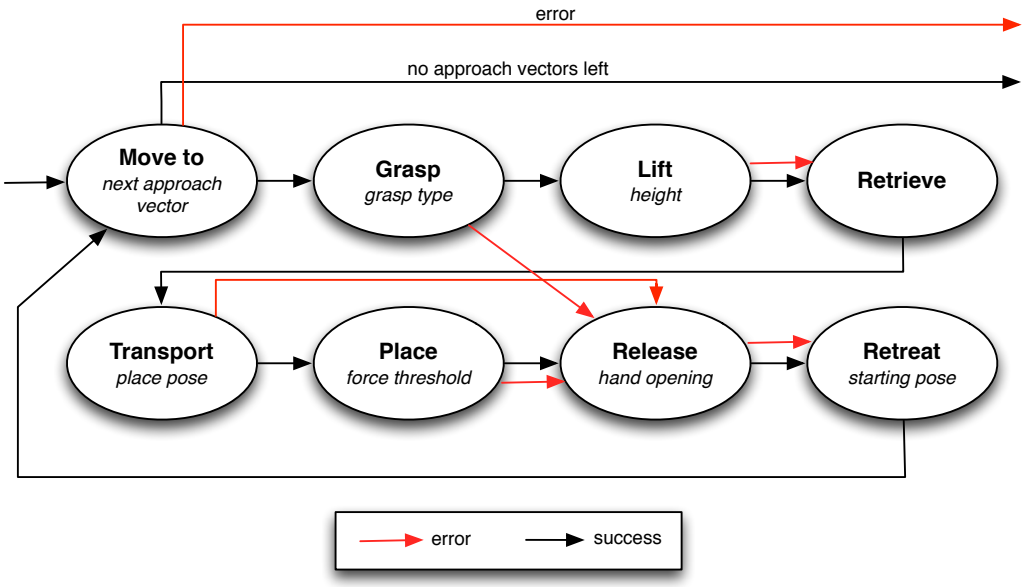


Figure 6.10: Manipulation task definition for the APC. Parameters required by each primitive are written in italics. Each primitive is configured before activating it. The *approach vector* and *grasp type* are obtained from the list of plans, the other parameters are defined in a configuration file and do not depend on the plan or the bin.

Figure 6.11: Grasp planning process for the APC challenge. Left: Initial image with bin reference frame. Middle: object segmentation, clustering and recognition. Right: grasp planning.

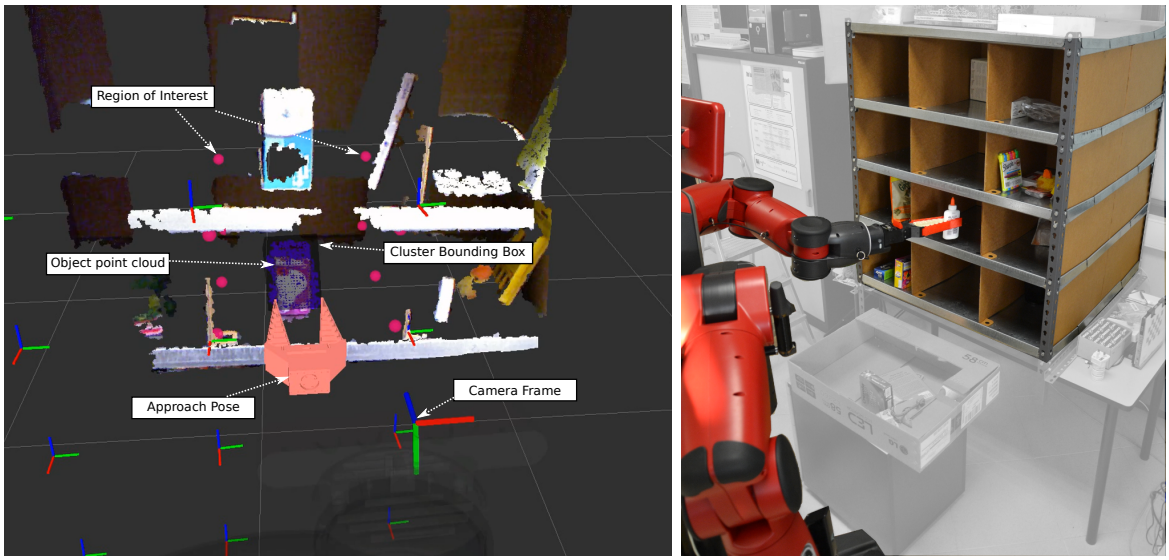


Figure 6.12: Left: Result of the visual and planning pipelines after the exploration of a bin of the APC shelf, the approach vector is represented by a CAD model of the gripper. Right: Execution of a manipulation task on a target object inside a bin of the APC shelf.

6.6 Conclusion

In this chapter, we have presented the layered software architecture used to integrate the proposed building blocks of the complete contact-based manipulation system.

Aside of the human inspired components identified in Chapter 2, there are other underlying abilities that are important and necessary for autonomous robotic manipulation, such as path planning, motor control, object perception or kinematic chain solvers. In this chapter we have also shown the basic skills that enable us to build more complex abilities on top of them. The work presented in this chapter, does not only include the architecture concepts and the implementation of some basic abilities. It also provides a tool to automatically generate code stubs for new modules, easing the creation of modules, increasing the readability of the generated code and saving a lot of programming and configuration time.

However, there are several drawbacks that need to be addressed in future versions of the architecture. Regarding message synchronization, when using a pipeline, it is not possible to determine which input generated which output. This is very important for example for learning algorithms. The design of pipelines, uses streaming data from a source and processes each message. Synchronization tools that allow the user to tag messages or to know exactly which message originated which output are an idea to

solve this issue. Other solutions such as a list of timestamps or synchronous pipelines are possible ideas that could solve this issue as well.

The process to set up a new experiment is complex and time consuming. First, all the involved modules have to be identified. Second, the configuration files for each of the modules has to be completed and the module to module connections have to be detailed. A graphical tool would be very useful in order to automatically generate the configuration files that properly connect modules to each other. Such a tool could rely on task specifications similar to the ones used in this thesis to generate the required task description and configuration files. That would complete the layered architecture allowing us to program at any abstraction layer, from low level drivers to high level graphical programming like Aldebaran's choregraphe[3] or Scratch[4].

Shared resources management is another important issue that has to be addressed. It is possible that different primitives use the same services or that two services use the same hardware interface. In the current version of the architecture, the developer has to be aware of this issue when setting up a new task. However, tasks will eventually become more and more complex and it will be very difficult to handle the shared resources manually. Thus, a mechanism to handle this issue has to be provided. A possible solution can be a warning system that automatically checks the modules used by the top level task.

The software architecture presented in this chapter was published together with the Tombatossals description in [Felip et al., 2015]. Moreover, the architecture and some of the abilities presented, were used by the RobInLab team during their participation in the 2015 Amazon Picking Challenge. Without the structure and organization of the architecture it would have been impossible to migrate all the important skills from Tombatossals to Baxter in only 4 months and be able to obtain good results at the lab and participate in the competition.

[3]Choregraphe: `www.aldebaran.com/en/robotics-solutions/robot-software/development`
[4]Scratch programming language: `scratch.mit.edu`

Chapter 7

Embodiment abstraction

The architecture presented in Chapter 6 allows us to port services, primitives and tasks to other platforms with a reasonable effort, however it is not possible to execute the same task description on two different embodiments and plans cannot be shared without manually modifying or tuning them.

The aim of the work presented in this chapter is to provide a tool that allows to use the same task descriptions regardless of the robot used. In this way, it is possible that a task learned by a robot could be transferred to another robot or that a manually designed task could be executed by different robots with different embodiments.

In this chapter we present a hardware abstraction mechanism built on top of the manipulation primitives paradigm presented in Chapter 3 that complements the software architecture presented in Chapter 6 and allows the same task description to be used across different hardware platforms. We demonstrate the abstraction mechanism by executing the same task description on two different robots with different arms and grippers.

7.1 Introduction

The approach extends the work presented in Chapter 3 by introducing an embodiment independent abstraction mechanism on top of the manipulation primitives paradigm. The abstraction offers several advantages. Firstly, complex actions can be described in terms of simple abstract primitives. Secondly, plans can be shared over different embodiments because the vocabulary of primitives is shared. Thirdly, manipulation primitives offer to high-level planners a vocabulary of reliable actions onto which build manipulation tasks and plans, thus simplifying and robustifying planning. Finally, these abstract models can be translated to embodiment specific models, constituting of reactive sensor-based controllers, such that the full capabilities of each platform can be utilised.

Task A cyclic, directed, connected and labelled multi-graph where the nodes correspond to *manipulation primitives* or other *tasks* and the edges to *events*.

Plan An instance of a *task* with the parameters set for a specific execution in a determined scenario and context.

Manipulation primitive A reactive controller, designed to perform a specific primitive action on a particular embodiment.

Abstract primitive An embodiment independent primitive action that can be translated to a manipulation primitive. It has required and optional parameters used to adapt the primitive behaviour to the specific action and will be used during the translation process.

Events Represent the detection of a specific perceptual or internal condition.

To flesh out the abstract primitives, in Chapter 3 we presented a complete set of reactive manipulation primitives for an arm manipulator equipped with a wrist force sensor and a three-fingered hand equipped with tactile sensors. We look into the power of the embodiment independent abstract primitives in the scenario where two different platforms with different hardware capabilities are used to complete a manipulation task using the same abstract description. A primitive based vocabulary is an effective way of transferring knowledge and plans between embodiments.

7.1.1 Related work

Few studies have addressed the issue of abstracting hardware from action. [Petersson et al., 1999] presented a somewhat similar framework but to our knowledge that framework has never been demonstrated in practice with multiple embodiments. An earlier version of the framework presented here appeared in [Laaksonen et al., 2010]. [Ellenberg et al., 2010] studied how algorithms for humanoid robot walking can be transferred between embodiments. The RoboEarth project has proposed a web platform for sharing environment models as well as action "recipes" between multiple robots using Ontology Web Language (OWL) [Tenorth et al., 2012].

From another perspective, Programming by demonstration (PbD) instead of focusing on transferring plans between robots, focuses on transferring skills from a human to a robot. However, the problems that need to be solved by PbD (a.k.a. Imitation Learning) include the task generalization problem, where the demonstrated task has to be described using a general representation that can be grounded on a robot. The task representations used by PbD include symbolic representations, sensorimotor representations or machine learning tools such as Artificial Neural Networks (ANNs), Radial Basis Functions (RBFs) or Hidden Markov Models (HMMs) [Billard et al., 2008].

Human action detection and understanding provides tools for abstract action representation. Regarding the symbolic and semantic representations used in this context, [Yang

Abstract primitive	*Parameters*	*Meaning*
move	target, *trajectory*, move type	Move without object.
transport	target, *trajectory*, move type	Move with object.
place	target, *trajectory*, move type	Place down object.
push	target, *trajectory*, move type	Push object.
grasp	*preshape, object size*	Grasp object.
release	*hand opening*	Release object.

Table 7.1: Abstract primitives and parameters (optional parameters in *italic).*

et al., 2015] propose a framework for learning the semantics of manipulation actions where using a Combinatory Categorical Grammar (CCG) based learning models, the manipulation plans can be obtained from a video sequence. However, after obtaining the task definitions, it is still future work to parametrize each atomic action to address a specific scenario and execute the task on a real robot.

7.2 Embodiment independence through abstraction

For the definition of abstract tasks, the same mechanisms proposed in Chapter 3 can be used. Tasks are composed of manipulation primitives connected by events. However, manipulation primitives and events are embodiment specific. For the implementation of the abstraction mechanism we have defined the concept of abstract primitive: a semantically meaningful primitive action that can be translated to an embodiment specific manipulation primitive. Like the manipulation primitives, abstract primitives are also configurable using parameters. There are *required* parameters that need to be specified for the primitive to work and *optional* parameters that are used to provide additional information to the underlying controllers.

The set of abstract primitives proposed in this work is shown in Table 7.1. This set of abstract actions allows moving objects by grasping or pushing them and has been found to support many common manipulation actions.

When the primitives are translated to an embodiment specific manipulation primitive, the required parameters which describe constraints need to be fulfilled but the optional parameters can be ignored if necessary as their purpose is to serve as hints how to perform the task.

All primitives except *grasp* and *release* are related to arm motions. The required parameters for these primitives define the target pose to move the arm to and the type of the motion. The motion target can be a single waypoint or a trajectory represented as a set of waypoints. However, defining a strict trajectory to be followed should be avoided when not made necessary by the task to allow each embodiment to use its own capabilities in the best possible way. Defining only the end pose is usually sufficient from the task perspective and leaves the freedom for the embodiment to choose a collision free path for that particular embodiment.

Abstract event	*Meaning*
success	Primitive successfully completed.
grasp_stable	Stable grasp detected.
grasp_lost	Grasp loss detected.
timeout	Timeout for specified time.
hardware_failure	Hardware failure detected.
error	Generic error.

Table 7.2: Abstract events.

In addition to the motion target, the type of motion is specified. Supported motion types include free, guarded, and constrained motions. In free motion, the embodiment is free to use any path to reach the target. In a guarded motion, the embodiment is required to use a Cartesian straight line path. In a constrained motion, rotational degrees of freedom can be constrained to remain the same for the duration of the motion. This is useful, for example, to transport containers with liquid. The underlying idea in the required parameters is thus to constrain the effects of a primitive rather than the ways to achieve them.

The grasp primitive allows to use an optional parameter to choose the grasp preshape and the object size. Because the parameters are optional, they can be ignored by platforms which do not support a particular grasp type. In that case, the primitive is likely to be translated to the closest possible grasp available.

To allow the embodiment independent description of tasks, the state transitions need also to be described in an abstract fashion. This is done using abstract events shown in Table 7.2. The events are related to completing a primitive successfully (*success*), grasp stability (*grasp_stable, grasp_lost*), and failure conditions (*timeout, hardware_failure, error*). Each platform is again free to use the available sensor set in any possible way to detect these events.

It should be noted that the primitives and events at the abstract level are not coupled to any particular embodiment. An important note here is that the sets of abstract primitives and events need to be rich enough in order to allow wide use of sensors in the embodiment specific controllers, while at the same time it is important to keep the semantic meanings of the abstract entities clear to allow the mapping between abstract and platform specific sensor events and manipulation primitives.

7.2.1 Abstract Task Description

The abstract task description (ATD) is a hardware independent description of a manipulation task. As in Chapter 3, tasks are defined as cyclic, directed, connected and labelled multi-graphs where the nodes correspond to abstract primitives or abstract tasks and the edges to abstract events. The definitions of task, plan, event and manipulation primitives are detailed in Sec. 3.2. XML (eXtensible Markup Language) is used to describe the relevant information, such as the abstract primitives and the tran-

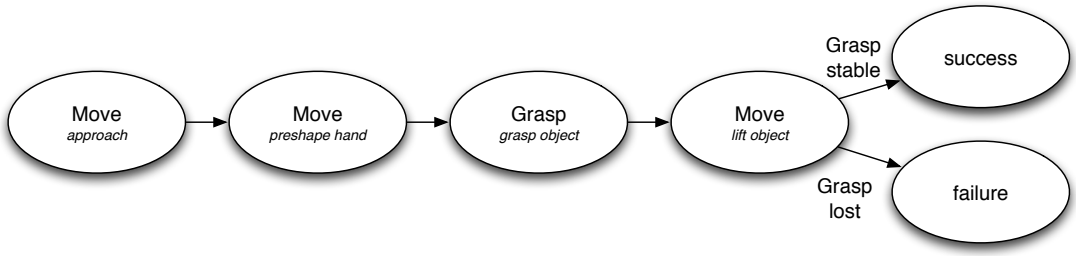


Figure 7.1: An example abstract task definition, describing a simple grasp and lift manipulation task. Some of the elements have been left out for brevity, e.g. properties of the object and some of the common edges, e.g. timeout to the failure state.

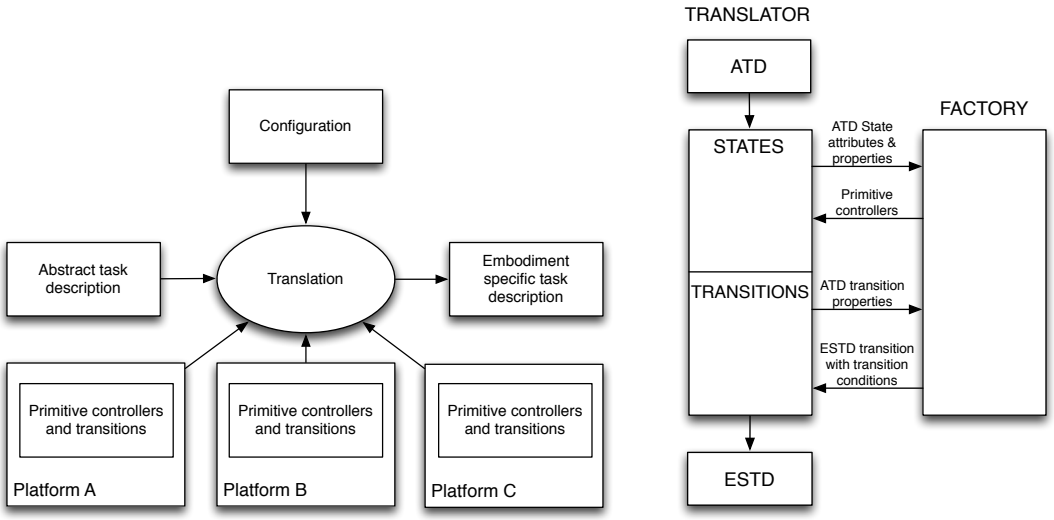
```
<statemachine>
  <state name="approach" type="move">
    <movement>free</movement>
    <hand_shape>open</hand_shape>
   </state>
  <state name="preshape_hand" type="move">
    <movement>guarded</movement>
    <hand_shape>pinch_grasp_preshape</hand_shape>
  </state>
  <state name="grasp_object" type="grasp">
    <movement>guarded</movement>
    <hand_shape>pinch_grasp</hand_shape>
  </state>
  <state name="lift_object" type="transport">
    <movement>guarded</movement>
    <hand_shape>pinch_grasp</hand_shape>
    <path>
      <position>0.2 0.6 0.25</position>
    </path>
  </state>
  <state name="success_end" type="success">
  </state>
  <state name="fail_end" type="failure">
  </state>
```

```
  <transition origin="approach"
   destination="preshape_hand">
    <success/>
  </transition>
  <transition origin="preshape_hand"
   destination="grasp_object">
    <success/>
  </transition>
  <transition origin="grasp_object"
   destination="lift_object">
    <success/>
    <grasp_stable/>
  </transition>
  <transition origin="lift_object"
   destination="fail_end">
  <grasp_lost/>
  </transition>
  <transition origin="lift_object"
   destination="success_end">
    <success/>
    <grasp_stable/>
  </transition>
</statemachine>
```

Figure 7.2: XML definition of the Abstract Task Description shown in Fig. 7.1.

sitions triggered by abstract events. In addition to nodes and edges, information about the environment such as obstacles and the location, mass and approach direction to the target object are included in the abstract task description. All the properties and definitions in XML are hardware independent.

The abstract task is described through definition of nodes and edges. Both nodes and edges have properties that can be used to further inform of the intended action. The most important node property is its *type*, corresponding to one of the primitives introduced above or the special states *"success"*, or *"failure"*. The two latter types indicate end states of a task with either success or failure reported to the higher level controller. In addition, the parameters of the primitives are specified as node properties. For example, the hand preshape for grasping or the target position of the end-effector can be set through node properties. The edge properties describe the set of abstract events which



(a) Translation process

(b) Relationship and communication of the translator and factory

Figure 7.3: Translation process and components.

trigger the node transition. For example, the loss of a grasp can trigger a transition to another node.

The attributes are the key factor in selecting the manipulation primitives during the translation process explained in Sec. 7.2.2. An example abstract task and its XML definition, describing a simple grasp and lift manipulation, are shown in Figs. 7.1 and 7.2. Some of the elements have been left out for brevity, e.g. properties of the object and some of the common edges, e.g. timeout to the failure state. It should be noted that the task description does not need to be a sequence or a tree but it can be any directed graph, however, the simple form in the example is used to limit the size of the associated XML code shown.

7.2.2 Translation from ATD to ESTD

The translation process connects the Abstract Task Definition (ATD) and the Embodiment Specific Task Definition (ESTD). The translation takes the abstract task definition as an input, and translates it into an embodiment specific task definition. The high-level translation process is depicted in Fig. 7.3(a).

As can be seen in Fig. 7.3(a), the translation component needs input defining the configuration of the translation process, i.e., the target platform and the platform specific events and manipulation primitives used directly in the embodiment specific task description. The benefit of this arrangement is that the only hardware dependent blocks

shown in the figure are the manipulation primitives and events that are platform specific. The critical requirement of real-time operation for sensor-based control is also fulfilled as the embodiment specific task can be run as is, without any additional overhead from maintaining hardware independence.

The translation process requires a mapping component which produces the embodiment specific task description from the abstract description. In our case the mapping component is constructed from two sub-components, shown in Fig. 7.3(b). The first part, *translator*, consists of necessary book-keeping and the internal logic that is independent from the embodiment. The translator also constructs the final ESTD which is used to execute the desired abstract task. The second part, *factory*, handles the embodiment specific construction of the nodes and edges of the ESTD, i.e., the sensor based manipulation primitives and transition conditions. This division was made to reduce the implementation time of the factory, which needs to be implemented for each different embodiment. The relation and the communication between these two sub-components are shown in Fig. 7.3(b). The factory also receives object and environment information of the ATD in addition to a particular node or event and its properties. This gives the factory the complete information needed for the manipulation primitives and events. The translation process proceeds as shown in Fig. 7.3. First, each of the abstract nodes is mapped independently by the factory to a suitable embodiment specific manipulation primitive. Then, the abstract events (transitions) are processed in a similar fashion.

The embodiment specific factory, uses abstract primitive parameters and environment information to choose a suitable embodiment specific controller and its parameters. Typically, each type of abstract primitive is mapped to a certain corresponding embodiment specific primitive, although it is possible that this relation is not one-to-one or even static. For example, it is possible to map different abstract arm movement primitives to a single embodiment specific primitive if that primitive can be parametrized in a suitable fashion, as we show in Table. 7.3. In addition to choosing the type of the controller, the factory can deliver embodiment specific parameters to the controller. These can be used, for example, to communicate a collision free path for that particular embodiment. Thus, in this case, the factory will also act as an embodiment specific path planner. A similar process is in place for the transition events, that is, the factory produces computation nodes for sensor processing which use the available sensors of each embodiment to detect the events.

For free motions in a collision free space and guarded motions, common primitive controllers can be used over several embodiments. This is possible by having common control and sensor interfaces for the arm, which in our case perform either Cartesian or joint space velocity control. Thus, we can use manipulation primitives that use the arm velocity control for all hardware platforms without modifications just by setting appropriate parameters through the embodiment specific factory. The same applies to

Abstract	*Tombatossals*	*Extra parameters*	*Control and sensors used*
Grasp	Robust grasp	Pregrasp size	Arm control, FT and tactile sensors
Move	Transport	-	Arm control
Push	Transport	-	Arm control
Transport	Transport	Obstacles, constraints	Arm control
Place	Place	Contact threshold	Arm control, Force-torque sensor
Slide	Slide	Slide force threshold	Arm control, Force-torque sensor
Release	Release	Hand position	Arm control

Table 7.3: Mapping of abstract primitives to Tombatossals implementation. Extra parameters show the parameters that are not in the abstract definition given in Table 7.1 but can be specified for the embodiment specific primitive.

the transition conditions, for example a timeout transition condition can be used across all platforms as the condition relies on measurement of time which should be available in every platform.

The rules that are observed in the translation process are simple:

- Each node in the ATD must correspond to one node in the ESTD.
- Each edge in the ATD must correspond to one edge in the ESTD.
- Each edge label in the ATD can be represented by one or more edge labels in the ESTD.

These rules ensure that the execution of the ESTD can be traced back to the original abstract task description. This allows the system to report back failures to higher level so that the higher level system operating on the abstract task description is able to reason using the same concepts. The possibility to represent an abstract transition condition by more than one embodiment specific ones allows, for example, to check the success of multiple manipulation primitives, such as separate arm and hand controllers, with a single success transition condition in the ATD.

While the translator component is universal across all embodiments, the factory component needs to be built specifically for each platform. The complexity of the factory affects the flexibility of the final system. A simple factory with fixed mappings between abstract and embodiment specific primitives and events is sufficient for many relatively simple tasks. Complex factories considering for example path planning for redundant manipulators or the choice of a grasping primitive among several are possible and discussed more in Sec. 7.4. If the factory is unable to find a suitable mapping for any reason, the mapping fails which is reported back to the task level. However, it should be noted that the factory is often fairly simple to implement because there are only a limited number of abstract primitives, event types and parameters, and the factory needs to consider only one primitive or event of an abstract task at a time.

Abstract	*Melfa*	*Extra parameters*	*Control and sensors*
Grasp	Grasp	grasp force	Arm control, tactile sensors
Move	Move	-	Arm control
Push	Move	-	Arm control
Transport	Transport	Motion constraints	Arm control, tactile
Place	Transport	Motion constraints	Arm control, tactile
Slide	-	-	-
Release	Release	Hand position	Arm control

Table 7.4: Primitives for the Melfa platform. Extra parameters show the parameters that are not in the abstract definition given in Table 7.1 but can be specified for the embodiment specific primitive.

7.3 Experimental validation

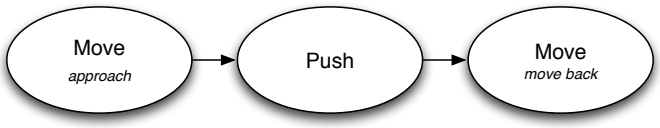
We demonstrate the mapping of the abstract state machine by showing a pick and place task that needs first clearing the path to the object to be grasped. This is achieved by developing two simple abstract task descriptions, the first to push an object away (see Fig. 7.4(a)) to clear the path to perform the second action: a simple pick and place (see Fig. 7.4(b)).

To enable mapping of the ATD, we implemented the translation component described in Section 7.2.2 for two different platforms, *Tombatossals* and a 6-DOF Melfa RV-3SB arm with a 1-DOF WRT-102 gripper from Weiss Robotics. The WRT-102 gripper is based on the PG-70 parallel jaw griper from Schunk. The implementation included the required platform specific controllers for the different nodes in the ATD and the platform specific transitions, as well as the required configuration information.

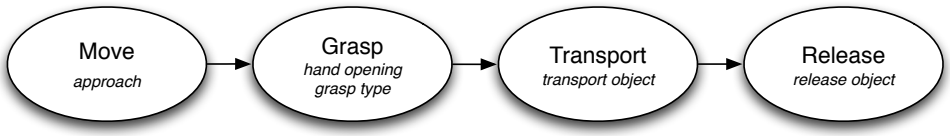
While the translation and the different requirements for the primitives are shown for Tombatossals in Table 7.3, the same information is available for the Melfa platform in Table 7.4. The Melfa platform does not utilize as much sensor feedback in the primitives due to the difference in hardware. The primitives for the Melfa platform are, in general, different from the primitives presented in Chapter 3 for Tombatossals as the SDH hand integrated into Tombatossals is much more capable in terms of DOF for example. The effects of the use of different embodiments can be seen, for example, in the *grasp* primitive. The Melfa robot is not able to do any of the corrections that *Tombatossals* does, it is only possible to perform the force adaptation using the tactile sensors.

For the implementation of the manipulation primitives for the Melfa platform, we used simple strategies. The basic guidelines used for the implementation are provided in the following list:

- Grasp: Closes the gripper until the desired *grasp force* is detected by the tactile sensors.



(a) Push object abstract task definition.



(b) Pick and place abstract task definition.

Figure 7.4: Abstract task definitions tested on Tombatossals and Melfa platforms.

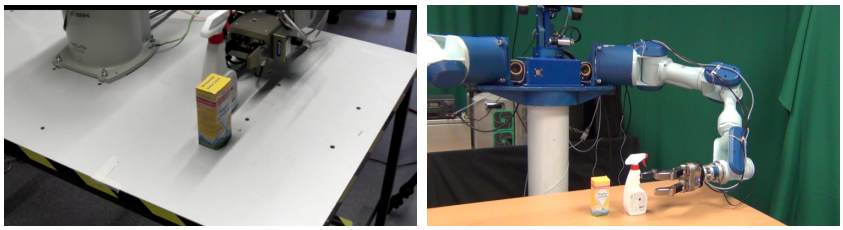
- Move: This is a very similar implementation of the transport primitive described in Chapter 3 for Tombatossals.
- Transport: Behaves exactly as the move primitive but keeps applying force with the gripper. Otherwise the transported item would be loosened by the gripper.
- Release: Open the gripper until the fingers reach the *hand position* parameter.

As a result, shown in Fig. 7.5, we were able to push away one object and grasp the second one based only on the sensor data from the hand and the arm, when given estimates of the pose of the objects. Using the same abstract task definitions for both platforms shows clearly that we are able to use abstraction and then turn this abstract information to platform specific primitives and transitions used in the sensor-based control.

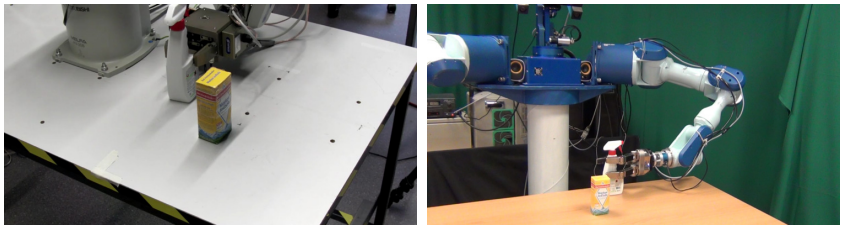
In the context of the demonstration we used the same Cartesian controllers for both arms. On the other hand, the hands are too different in terms of kinematics and sensors so that each hand had its own implementation of control. Also the transitions for grasp stability or instability were customized for each of the platforms in order to effectively use the different sensor capabilities available on the platforms. It should be noted that the task was nevertheless described using only the abstract description, without any embodiment specific information.

7.4 Discussion

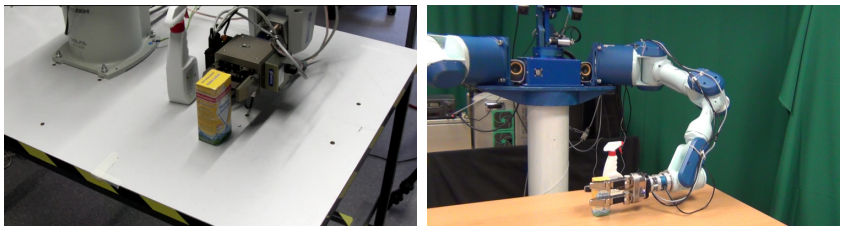
In the approach presented in this chapter, some of the manipulation primitives implemented in Chapter 3 have been implemented for the Melfa platform. Though all of them are intended to have the same behaviour and effects, their implementation can



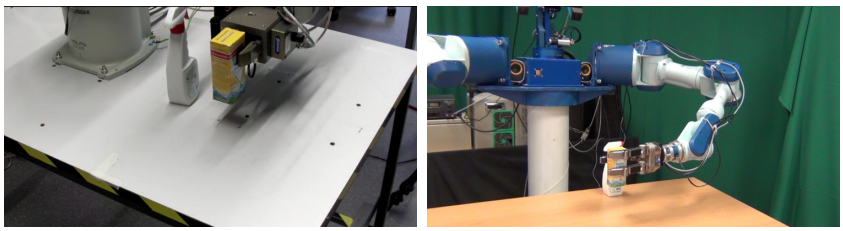
(a) Approach



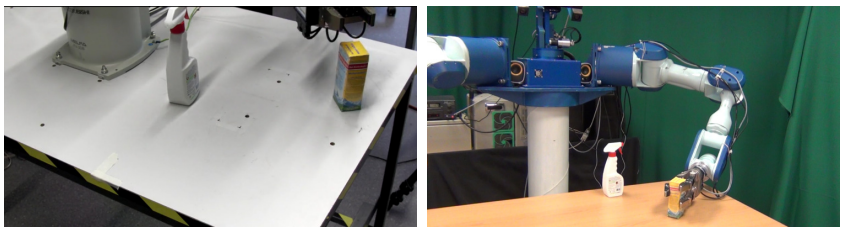
(b) Push



(c) Grasp



(d) Lift



(e) Release

Figure 7.5: Action execution on different platforms. Left column: Melfa RV-3SB with PG70 gripper. Right column: *Tombatossals*

vary significantly due to the mechanic, kinematic, and perceptual differences between embodiments. Especially in the case of the grasping primitives, the differences in hand construction (number of fingers, number of joints, number of actuators) and available sensors are on a level which makes automatic construction of primitives a grand challenge. In order to better exploit the advantages of each embodiment, it would be possible to use a more detailed abstract description of the grasping primitive, for example, differentiating between grasp types such as enveloping grasps, power grasps, precision grasps, or hook grasps. Nevertheless, the state-of-the art in grasping does not currently allow this level of abstract information to be used automatically and therefore each of the grasp flavours would need to be adapted manually, depending on the hand characteristics. This is the case especially if the full reactive capabilities of the embodiment are to be used.

On the other hand, primitives related primarily to control of arm motions can be general so that the same manipulation primitive implementation can be used on multiple embodiments, as we demonstrated experimentally. However, in order to enable the full capabilities of an embodiment to be used, the path planning of the arm motions needs to consider the particular embodiment. This means that the factory component of the translation process is embodiment dependent, at least to some extent. Nevertheless, the planning of collision free paths between end-effector poses can be performed using openly available software libraries and therefore the implementation of the factory is possible with reasonable effort.

The position of the factory component is central in the approach. Differing capabilities of different embodiments, for example the size of the workspace, have the effect that there is no way to guarantee that an abstract plan would be translatable to any embodiment. Without requiring certain capabilities, it cannot be known with a certainty that a specific abstract plan can be executed on a specific embodiment. In the longer term, general principles on how embodiments could automatically instantiate sensor-based primitives would offer great benefits. However, a complete solution would need to 1) analyse and abstract a skill performed by an existing system and 2) be able to map the abstract skill to the present embodiment.

7.5 Conclusion

This chapter presented an abstraction framework allowing multiple embodiments. The main contribution is the abstraction framework and translation mechanism. We showed experimentally that the transfer of action plans is possible between different system setups while retaining the specific reactive capabilities of each embodiment.

These results complement the results from the RoboEarth project, where similar results have been shown for the higher level planning without the viewpoint of reactive primitives presented in this thesis. The results encourage us to believe that manipulation

problems can be solved in complex, unstructured scenarios while retaining hardware independence on a higher level. However, immediate feedback capabilities seem essential in coping with the complexity of the world.

The embodiment specific manipulation primitives currently require careful design for each embodiment. Procedures which could automatically at least bootstrap the building of the controllers, or even construct the controllers, would be very valuable. It seems that the use of machine learning techniques would be an interesting and possibly profitable avenue of research in this direction. This approach would most likely require high quality simulations of the embodiment in order to provide training data for the learning approaches, a possible application of the simulation engines presented in Chapter 5.

The abstraction mechanisms presented, implemented and validated in this chapter were published in [Laaksonen et al., 2010] and extended in [Felip et al., 2013].

Chapter 8

Object perception and recognition

In chapter 2, based on neuroscience experiments that studied how humans physically interact with objects at sensorimotor level, the building blocks for a complete autonomous sensor-based manipulation system were settled. In chapters 3, 4 and 5 we have presented the implementation and validation of each of the building blocks. However, the identified blocks focus on physical interaction and do not include any visual perception or recognition of objects, which is necessary for object manipulation.

That gap is addressed in this chapter. We present an object detection and recognition component, that can be integrated into the presented framework in order to provide information about the objects. First, we have taken inspiration from human and primate studies to propose a theoretical scheme for hierarchical object recognition based on a three step process: classification, recognition and recall. Finally, the proposal is implemented and tested on a real robot. The integration on the system, can be done using the concept of perceptual primitives already introduced in Chapter 3. This primitives allow us to include specific perceptual actions in the task definition.

8.1 Introduction

The visual cortex of humans and other primates is composed of two main information pathways, called ventral stream and dorsal stream in relation to their location in the brain, depicted in Fig. 8.1 [Goodale and Milner, 1992]. The dorsal, "where/how", stream is concerned with providing the subject the ability of interacting with its environment in a fast, effective and reliable way, such as in limb movements. The dorsal stream includes areas especially dedicated to extract and encode 3D features of objects in a format suitable to be used for planning and executing reaching and grasping actions toward them. The ventral, "what", stream is instead devoted to perceptual analysis of the visual input, such as in recognition, categorization and assessment tasks.

The streams dissociation has been supported, but also criticized, by the neuroscientific community, and the original theory is constantly being revised and updated. The trend is towards a more integrated view, according to which, the two streams have complementary tasks and often interact with each other [Goodale, 2004].

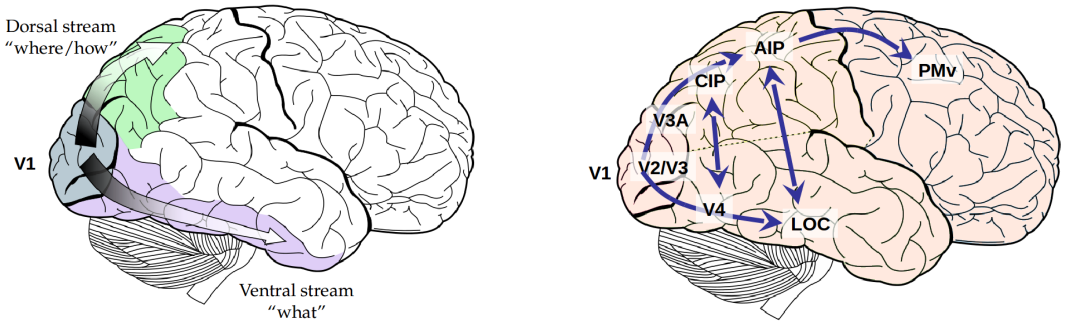


Figure 8.1: Left: Dorsal and ventral streams of the human brain. Right: Human brain areas of the dorsal and ventral streams.

In previous, related works, Chinellato et. al. modeled the visuomotor processing performed by dorsal stream areas in reaching and grasping actions [Chinellato and Del Pobil, 2008], [Chinellato and del Pobil, 2016]. That work devoted special attention to the area of the primate brain dedicated to grasping, Anterior Intraparietal Sulcus (AIP), but taking also into account possible interactions between the ventral and the dorsal streams [Chinellato and Del Pobil, 2009]. Their modeling efforts were validated by the implementation of the fundamental concepts on a real robotic setup, resulting in a skilled vision-based grasping behavior [Chinellato et al., 2008], [Grzyb et al., 2009]. The whole model framework is represented in Fig. 8.2, for a full detailed description of the model see [Chinellato and del Pobil, 2016].

In this chapter, we extend the work from [Chinellato and Del Pobil, 2009] implementing the ventral stream part of the model (i.e. object recognition) as presumably executed by the primate visual brain (light blue modules from Fig. 8.2). We offer a hierarchical interpretation of the incremental identification capabilities of a subject presented with geometrical 3D objects.

According to the proposed framework, ventral stream processing consists of 1) identifying the object class; 2) recognizing a single object within a class; 3) identifying a previously encountered object even among completely similar candidates. The first two steps of our object identification model have been implemented on a robot setup. The system is able to classify target objects in one of a given number of classes, and subsequently recognize a certain object among objects of the same class, taking advantage also from the estimation of object weight.

8.2 Neuroscience background

In humans, the visual information is processed through the dorsal and ventral streams in a sequential manner. Each of the streams goes through different brain areas that are

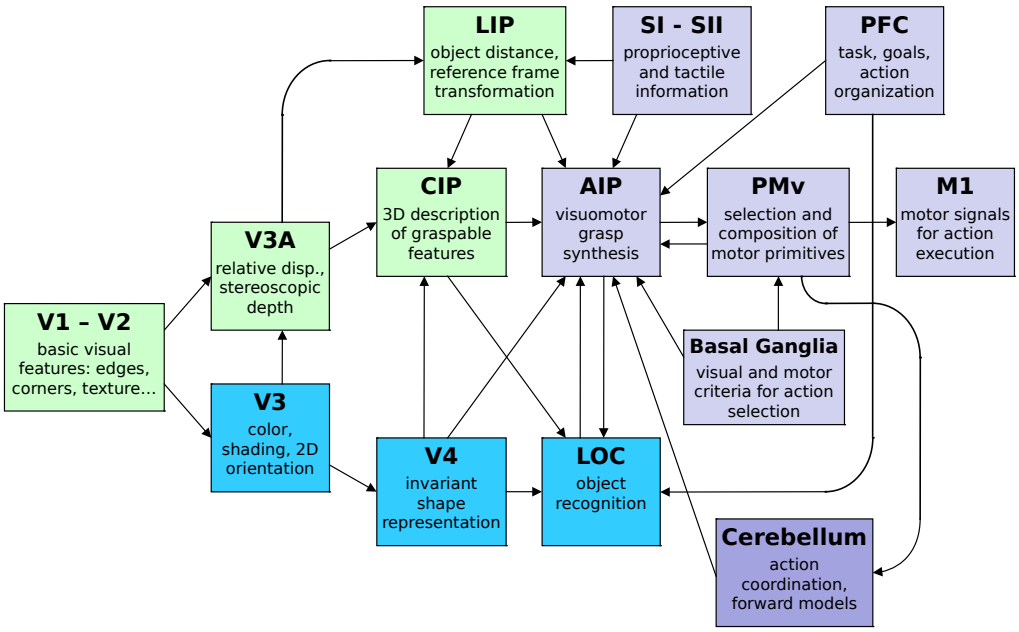


Figure 8.2: Model framework depicting all the principal areas involved in the planning and execution of vision-based grasping actions. Green areas correspond to the dorsal stream. Light blue areas are considered to be ventral stream areas. Violet areas correspond to pre-motor cortex, motor cortex and the end of the dorsal stream.

separated from each other depending on their function (see Fig. 8.1). In the ventral stream, starting in the Primary Visual Cortex (V1), visual information is processed in a pipeline-like sequence until the objects in the scene are recognized and their identity is recalled in the Lateral-Occipital Complex (LOC). Although the information flows from V1 to LOC there are many feedback connections that connect the different areas to each other. In this chapter have used the functional model proposed by [Chinellato and del Pobil, 2016] and extended the already implemented dorsal stream with the implementation of the ventral pathway (see Fig. 8.2).

At the beginning of the visual processing, ventral and dorsal streams are not separated. Starting at V1 area, neurons are mainly sensitive to edges but also to the more global organisation of the scene. As information is further relayed to subsequent visual areas,

it is coded as increasingly non-local frequency/phase signals [Hubel and Wiesel, 1977]. The mathematical modelling of this function has been compared to Gabor transforms. Neurons in Secondary Visual Cortex (V2) are tuned to simple properties such as orientation, spatial frequency, and color [Hegdé and Van Essen, 2000]. Third Visual Complex (V3) area is where the division of the visual pathways begins, dorsal and ventral V3 have distinct connections with other parts of the brain, and contain neurons that respond to different combinations of visual stimulus. Colour-selective neurons are more common in the ventral V3 also known as VP. Visual Area V4 (V4) exhibits long-term plasticity, encodes stimulus salience in an invariant shape representation and is sensitive to attention [Sereno et al., 1995].

The last area is the LOC, in this area is where the results from the other ventral areas are integrated and the final steps of object recognition are performed. Object representation in LOC is highly invariant with respect to the stimulus type, showing equally good performances with either 3D or silhouette images, different color maps, lighting and so on. This suggests a higher level, conceptual representation of objects, independent of the actual stimulus that allowed recognition [Kourtzi and Kanwisher, 2001]. Object recognition in the ventral stream is very likely a "faded" process rather than a binary one. In fact, activation in the anterior part of the LOC is modulated by the actual level of recognition, and not by the nature of the stimulus [Bar et al., 2001]. In any case, geometric data are integrated with additional information, regarding for example color and texture of objects, to speed up and make object recognition more reliable [Grill-Spector et al., 1999].

Object recognition is performed gradually and hierarchically [Grill-Spector et al., 1998], [Bar et al., 2001]. Other findings indicate that the identification process is composed of at least two sequential stages, categorization and identification [Grill-Spector and Kanwisher, 2005]. In the first stage, an object is classified as belonging to a given class or family of objects, and such process is strikingly fast, requiring just few milliseconds. The classification delay is so short that there is probably time to feed category information to the dorsal stream, for improving the online estimation of action-related features. The second stage of object recognition is proper identification, performed by LOC, in which object identity is recognized within its category.

Regarding possible connections of ventral stream areas with the dorsal stream, a direct link has been found in the macaque brain between the most 3D responsive ventral inferior temporal area (the lower bank of the superior temporal sulcus) with the Caudal Intraparietal Sulcus (CIP) [Janssen et al., 2000]. This link could indicate both a ventral contribution to pose estimation, which we have previously modeled [Chinellato and Del Pobil, 2009] and a dorsal effect in object recognition.

8.3 Computational model of V4 and LOC areas

This section presents the description of the ventral stream modules from the brain functional model shown in Fig. 8.2 and described in [Chinellato and del Pobil, 2016].

Visual processing in the ventral stream is based on the production of increasingly invariant representations aimed at object recognition. In the functional model of the brain we follow, the ventral stream starts at V3 area (Fig. 8.2), region V4 codes at the same time shape, color and texture of features, which are then composed in the LOC to form more complex representations recognizable as objects. Output from area V3 is thus used by V4 to build a viewpoint invariant simple coding of the object, that can be used to classify it as belonging to one of a number of known object classes.

Downstream from V4, the LOC compares spatial and color data with stored information about previously observed objects, to finally recognize the target as a single, already encountered object. Object identification is thus performed in a hierarchical fashion, where the target is first classified into a given class and, only later, exactly identified as a concrete object. In each of these steps, recognition is not a true/false decision, but rather a probabilistic process, in which an object is classified or identified only up to a given confidence level. Thus, confidence values should be provided by the classification and identification procedures, so that ventral information can be given more or less credit. If recognition confidence is high, visual analysis can be simplified, as most required information regarding the target object is already available in memory. If recognition is instead considered unreliable, more importance is given to the on-line visual analysis performed by the dorsal stream. An aspect relevant for modeling purposes, is the method employed by the ventral stream for performing object recognition [Ullman, 1996]. At least for the first classification stage, visual input is very likely compared to memorized 2D representations [Bülthoff et al., 1991]. A classification based on 3D representations would require mental rotation, and this can hardly be performed with the quickness observed in the experiments of [Grill-Spector and Kanwisher, 2005]. Moreover, the consistent preference of some "canonical" views during free and classification-oriented object exploration indirectly supports the existence (if not the dominance) of 2D object representations [Blanz et al., 1999], [James et al., 2001]. For this work, a viewpoint invariant classification procedure was implemented, based on basic 2D global object representations.

Considering the output of the V3 area as a segmented 2D contour of the object. Possible computational representations of 2D object contours are, for example, chain codes (e.g. Freeman Chain Code of Eight Directions [Freeman, 1961]) or 2D shape indexes (e.g. curvedness index).

Regarding possible dorsal contributions to ventral stream processing, various researchers pointed out that action-related information maintained in the dorsal stream is likely to

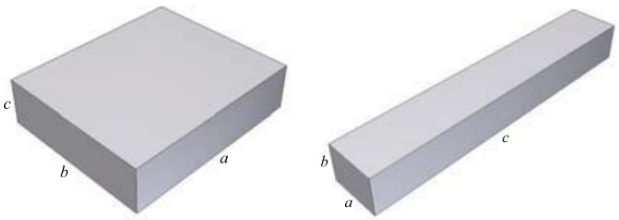


Figure 8.3: Examples of SOS (left) and AOS (right) dominant objects.

play an important role in the object recognition process. A set of possible affordances constitutes an additional way of identifying an object [Sugio et al., 1999], [Shmuelof and Zohary, 2005]. In this chapter we have modelled the dorsal-ventral interaction as a connection from the dorsal area CIP to the ventral area LOC as found in the macaque by [Janssen et al., 2000].

Two main neuronal populations have been distinguished in CIP. They both code for object orientation in space, but are selective for different object types. Surface Orientation Selective (SOS) neurons [Shikata et al., 1996] preferentially respond to flat stimuli of the kind shown on the left in Fig. 8.3. The second class of CIP neurons, Axis Orientation Selective (AOS) neurons [Sakata et al., 1998], represent the 3D orientation of the longitudinal axes of elongated objects (Fig. 8.3, right). The activation of SOS and AOS neurons according to different stimuli was previously modelled with the purpose of providing area AIP with information useful for grasp planning [Chinellato and Del Pobil, 2008]. Here, we employ this same information to aid LOC in object classification.

The SOS and AOS responsiveness found for the target object could be one possible format used by the dorsal stream to help the ventral areas in the recognition task. It is in fact very unlikely that two objects share the same SOS and AOS activations. CIP projections would thus provide the ventral stream with additional information for improving the reliability and speed of object recognition. For what concerns the representation of known objects, in their first years of development, human beings accumulate experience on properties such as color, texture, material, object identity, learning the likelihood of different relations among them. A working model of this recognition and generalization capacity should rely on a knowledge base founded on these properties (see e.g. the proposal of [Metzinger and Gallese, 2003]). In this chapter we use both SOS and AOS activations to aid object recognition and build a very simple knowledge base of geometrical shapes to use it for object identification purposes, reduced to basic features such as dimensions, color and weight.

In summary, emulating the mechanisms suggested by neuroscience studies, the approach to object classification proposed in the model is composed of a three stage process.

1) *Shape classification*. In this stage the target object is classified into one of a number of known classes. For example, a bottle would be classified in the class of cylinders. Simple visual information such as shape silhouette or a basic topographic relation between object features is enough for this task. No actual data regarding the size and the proportion of the object are considered. Nothing is inferred at this point about object composition, utility or meaning. The information recovered at this stage is used by early areas of the dorsal stream in order to estimate the size and pose of the object. This process is performed in the V4 area of the brain.

2) *Object Recognition*. Actual object recognition is the goal of this stage. The target object is identified as if the task was to name it. What was a general cylindrical shape in the previous stage is now identified as a bottle. Additional conceptual knowledge is thus added to the previous basic information. Composition, roughness, weight of the object can be inferred if not known for sure. The object proper use in different tasks is also recalled at this point. Object recognition directly affects the process of grip selection, providing a bias toward grasp configurations better suited to the object weight distribution, possible friction and common use. This process occurs in the LOC area of the human brain.

3) *Object Recall*. In this final stage, that also happens in the LOC area, a subject recalls a single well-known object which was encountered, and possibly grasped, before. Going back to the cylinder example, here it can be recognized as a wine bottle recently bought, and thus previously known and dealt with by the subject. Compared to the previous one, this stage adds confidence to the estimation of the object characteristics. To recognize an object as a bottle helps in estimating its weight, whilst to identify a previously encountered bottle provides an exact value of that weight.

In all stages, the classification process has to be viewpoint invariant. A very important issue is that object classification and recognition is always a gradual process, not a binary one, and each classification is accompanied by a confidence value, necessary to clarify its reliability. Any classification having a low confidence should be used prudentially, and if no class or object are clearly identified the system should rather provide a failed classification answer, to clarify that the situation is uncertain and needs further exploration. Feedback from execution outcome can later be used to complete and improve the world knowledge in these situations. The last stage of the process, Object recall, requires a higher level memory of the agent interactions with nearby objects, involving some sort of awareness regarding the nature of his behavior and his relation with the environment, and is thus beyond our goals and current modeling skills. The robotic implementation of the first two stages is described in the next section.

8.4 Implementation

The recognition system follows a hierarchical scheme, starting from categorization, then recognition and finally object recall. In this section the two first steps of the object recognition system described above are implemented on the robot setup of the Robotic Intelligence Lab. The implementation presented in this section takes into consideration the robotic setup used and the reduced universe of possible objects. It is a platform dependent implementation that intends to replicate the functional brain model for a further validation under certain known conditions.

8.4.1 Robotic setup

The robotic setup consisted of one PA10-7C 7DOF manipulator with a force-torque sensor and a barrett hand. A stereo camera Videre Design was mounted on the arm of the robot with an eye-in-hand configuration, see Fig. 8.5. This implementation was performed before the Tombatossals torso, described in Appendix A.1, was available. In fact, the system used for this implementation was later used to compose Tombatossals left side.

8.4.2 V4 area: Shape classification

The shape classification module has to categorize objects seen from different poses and distances. With this purpose, it has to consider object images globally, rather than focusing on local features. In the reduced world of the robot, the goal is to classify an object as pertaining to one of three known object classes: parallelepipeds (boxes), cylinders and spheres. This has to be done using only a couple of stereo images, without changing the viewpoint. Moreover, it is important to retrieve a value measuring the confidence in the classification, represented by the percentage of likeliness assigned to each class. As explained in the previous section, it is possible that the V4 area of the human brain encodes the objects using an invariant shape representation. Given that the input of the V4 area is the object contour, two different approaches were tested: a chain code representation and a curvedness index.

Chain code representation

The first tested object representation consisted in computing a chain code of the contour, which constitutes a representation that is invariant with respect to size and distance, while maintaining the feature topology necessary to identify the object. However, after the preliminary experiments, this solution did not provide the required behaviour. In fact, results on training objects from different viewpoints gave recognition success very close to 100%, but test objects were often misclassified. Moreover, even in the wrong cases, confidence was always very high, often above 98-99%. The conclusion is that the method is very good at recognizing known objects, but not at generalizing.

The sequential order of different object features, like straight and curved segments, or corners, would be enough for classification. Instead, the chain code representation takes into account and hence classify objects also according to the feature length, distinguishing for example a short cylinder from a long one. Moreover, classification should be much more shaded, with confidence percentages not always close to 100%.

Curvedness index object representation

This representation is based on only one index for each object, the curved fraction of its contour. The curvedness of a contour is calculated as the ratio between the length of its curved features and the total contour length. For the shapes in use, experimental data showed that parallelepipeds, cylinders and spheres normally possess linearly separable curvedness values.

8.4.3 LOC area: Object Recognition

After object class has been identified, the second step in the recognition process is to distinguish among objects of the same class. A number of fundamental features can be defined for object recognition purposes in order to perform this second step. For the experiments we have only used box-like objects, we exploit this assumption to select the features that will form our object representation in the LOC area. We considered the estimated size of the three sides (D_1, D_2, D_3 ordered from larger to smaller such that $D_1 > D_2 > D_3$), color (C), weight (W) and the activation of SOS and AOS neurons (SOS, AOS).

As discussed in Sec. 8.2, dorsal information is likely forwarded to the ventral stream, SOS and AOS activation is a sort of information that is very likely forwarded to the ventral stream by dorsal areas. The implementation of SOS and AOS activation are non-linear combinations of the estimated principal object dimensions, defined according to neurophysiological data and potential use in vision-based grasping actions, the transfer functions of both AOS and SOS were modelled by [Chinellato and Del Pobil, 2008]. SOS activation is defined by equation 8.1 and AOS activation is defined by 8.2

$$R_{SOS} = 1 - \left(\frac{D_1 - D_2}{D_1 + D_2}\right)^2 - 0.03\frac{D_3}{D_1 + D_2} - 0.5\frac{1}{1 + e^{-0.04(D_3 - H)}} \tag{8.1}$$

$$R_{AOS} = 1 - \frac{D_1 - D_2}{D_1 + D_2} - 0.37\frac{D_1}{D_3} - 0.5\frac{1}{1 + e^{-0.04(D_1 - H)}} \tag{8.2}$$

Where D_1, D_2 and D_3 are the dimensions in millimetres of the object's bounding box and $D_1 > D_2 > D_3$. H corresponds to the comfortable hand opening parameter which generally is 150mm. The constant values were tuned to match the real response obtained from real experiments. The details about the computational modelling of SOS and AOS neurons is provided in [Chinellato and Del Pobil, 2008].

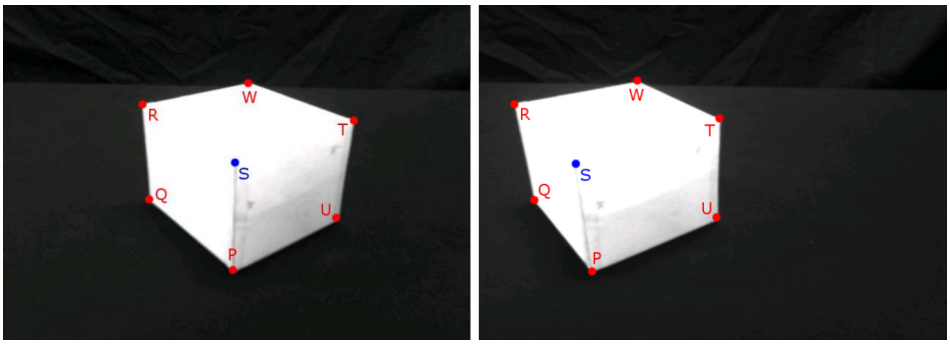


Figure 8.4: The corners of the box shaped object are used to determine its dimensions (D_1, D_2, D_3) ordered from larger to smaller. The dimensions are used for object recognition and approach vector computation in order to perform grasping actions on it.

The principal dimensions (D_1, D_2, D_3) are calculated exploiting the assumption of box-like objects. The principal corners of the object are detected in both images of the stereo pair (see Fig 8.4) and the main dimensions are calculated using the 3D position of the detected object points.

Given that the position and orientation of the object can be detected, a grasping action is performed on it, and using the force-torque sensor on the wrist, the weight of the grasped object can be estimated and used for recognition purposes.

8.5 Experiments

In order to validate the proposed implementation of the ventral stream and the functional model of the brain; two experiments have been carried out. The first one to test the implementation of the object classification module (V4 area), the second one to validate the implementation of the object recognition module (LOC area).

8.5.1 Scenario and assumptions

To ease the segmentation process in both experiments, objects are presented with light colors over a black background. A less restrictive object segmentation system that could be used for further experiments was detailed in Section 6.3.3. Objects are placed on top of a table in front of the robot inside its workspace.

8.5.2 Shape classification

For the implementation of the shape classification module (V4 area), two possible representations were proposed: chain code and curvedness index. However, during the preliminary experiments, the chain code was found to be not suitable and was dis-

Figure 8.5: Left: Objects used for the experiments on black background. Right: Robotic setup consisting of a PA10-7C manipulator with a Barrett Hand and a Videre Stereo vision system.

carded. In this subsection the curvedness index representation is experimentally tested and validated.

Test objects

For the categorization, we divided the objects into three main classes: box-like, cylinders and spheres, see Fig. 8.5 Left. The objects used for training the system are pure basic shapes while some of the objects used for testing are regular objects.

Curvedness index object representation

To validate the use of the curvedness index as a shape category descriptor, we have performed an experiment that consisted of two phases, training and testing.

The classification process begins with a training phase during which the system is presented with five different boxes (B), three cylinders (C) and two spheres (S). Images are taken again from 19 viewpoints distributed along a 90° range in azimuth, with elevation kept at about 40° to grant a clear 3D view of objects. Average curvedness values μ_K and corresponding standard deviations σ_K are calculated for the three classes, $K \in (B, C, S)$.

Given a test point c_i (i.e. the curvedness coefficient of object i), its degree of membership m_{iK} to class K is computed as the reciprocal of the relative distance to the class center:

$$m_{iK} = \frac{\sigma_K}{|c_i - \mu_K|} \tag{8.3}$$

At this point, classification percentages for the three classes K = B,C,S are given by:

$$p_{iK} = \frac{m_{iK}}{m_{iB} + m_{iC} + m_{iS}} \tag{8.4}$$

As explained above, a missing recognition response is better than a misclassification. To favor the former over the latter, a high confidence value of 70% is required to assign the object to any class. If no class reaches this value, the object is not classified, and an exploratory movement aimed at providing the robot with images taken from a different viewpoint is required. An exception is the case of uncertainty between boxes and cylinders. If $p_{iB} + p_{iC} \geq 70\%$, then the object is classified in the less restrictive class, i.e., as a cylinder. This is because in our biologically-inspired pose estimation system [Chinellato and Del Pobil, 2009] boxes provide more useful information for orientation estimation than cylinders. Thus, a misclassification of a box as a cylinder would just imply that some available information is not used, whilst a misclassification of a cylinder as a box would very likely cause a wrong interpretation of the available data.

Results and discussion

After performing experiments using the two proposed object representations, the results obtained are presented and discussed in the next subsections.

Curvedness index object representation

Classification results for objects in the training set are provided in Table 8.1 Left. Cases of misclassification are highlighted in **bold red** whilst uncertain cases are underlined. For the training set, only two problematic cases are identified, both for cylinders seen from a 0° angle (objects 5 and 6). It is not surprising that this is a difficult condition for the recognition system, as the contour provides limited if any information on curvature, and more elaborate methods which take into account shading would be required for proper classification.

Classification results for test objects are given in Table 8.1 Right. Most cases of missing classification regard the same problem observed for the training set. Cylinders seem to be difficult to recognize, especially for extreme viewing angles, in which their silhouette appears as a rectangle or as a circle. Nevertheless, the prudential decision of assigning the object to class C in case of uncertainty between box and cylinder, works in nearly all conditions: only objects 14 and 16 from the 0° viewpoint are finally misclassified, the first as a sphere and the second as a cylinder. Object 18 cannot be clearly put in any of the three classes, but it has one face that can be used for slant estimation, as cylinders, hence its classification as a cylinder is the most appropriate from a practical point of view.

8.5.3 Object Recognition

In the training phase, the system is provided with a number of labelled objects, and uses visual perception to associate detected features to object identity. For the recognition engine we have used a probabilistic linear estimator. A feature matches a given object

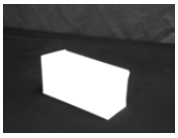
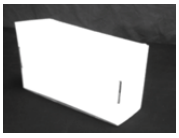
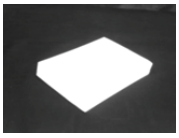
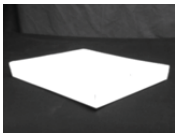
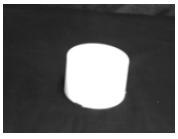
#	Object	0°	30°	60°	90°	#	Object	0°	30°	60°	90°
1		98.2 1.6 0.2	86.6 11.6 1.8	84.8 13.1 2.1	94.9 4.4 0.7	10		98.8 1.1 0.1	85.8 12.5 1.7	85.1 13.1 1.8	85.6 12.7 1.7
2		93.0 1.6 0.9	85.9 11.6 1.9	84.8 13.1 2.1	91.2 4.4 1.2	11		94.6 4.8 0.6	90.0 8.9 1.1	85.1 13.1 1.8	91.1 7.9 1.0
3		93.9 5.3 0.8	84.8 13.1 2.1	84.8 13.1 2.1	87.3 11.0 1.7	12		80.5 17.7 1.8	96.2 3.4 0.4	86.0 12.3 1.7	91.7 7.6 0.7
4		99.9 0.1 0.0	86.9 11.4 1.7	84.8 13.1 0.1	99.2 0.7 1.7	13		94.5 5.0 0.5	95.6 3.9 0.5	90.6 8.3 1.1	99.4 0.5 0.1
5		**86.2** 12.4 1.4	0.6 98.6 0.8	0.3 97.9 1.8	0.8 92.9 6.3	14		0.6 30.8 68.6	5.9 91.4 2.7	0.3 96.6 3.1	0.5 51.9 47.6
6		58.1 38.7 3.2	1.7 96.8 1.5	20.8 75.0 4.2	0.4 97.9 1.7	15		60.3 36.2 3.5	61.7 35.0 3.3	35.9 59.4 4.7	0.3 99.2 0.5
7		2.7 95.2 2.1	2.4 95.7 1.9	0.6 94.6 4.8	9.0 88.5 2.5	16		57.9 38.6 3.5	84.9 13.0 2.1	93.3 5.8 0.9	98.1 1.7 0.2
8		0.5 25.4 74.1				17		0.8 98.3 0.9	1.0 87.4 11.6	0.4 95.1 4.5	0.7 90.3 9.0
9		0.4 24.2 75.4				18		17.9 77.3 4.8	0.2 97.4 2.4	3.7 94.3 2.0	3.8 93.8 2.4

Table 8.1: Object classification percentages for different slants. Left: Training shapes (objects 1 to 9). Right: Test shapes (objects 10 to 18). Percentages of each class shown row-wise (B,C,S) for each object.

identity i if the set of features of the sample x is in the variability range of that object identity, expressed by the multidimensional mean μ_i and variance σ_i of its feature space:

$$\mu_i - n\sigma \leq x \leq \mu_i + n\sigma \tag{8.5}$$

where parameter n defines the tolerance of the classifier. We tested our classifier with a "risky" setting, $n = 3$, which should grant higher recognition rates but also more errors, and a more prudential $n = 2$, that should increase the number of unclassified samples. Statistically, $n = 3$ corresponds to about a 99% confidence interval, and $n = 2$ to approximately 95%. The mean and standard deviation vectors identifying the feature space of an object class are computed on a training set including samples of all available objects.

During the first phases of our experimental tests, we realized that the color feature is dominant, and no other features would be required if objects had different colors. Recognition tests in which shapes were distinguishable by color gave us more than 99% correct identification rate, showing that the problem was indeed too easy. For making our classifier more robust and test the importance of other features, in particular the SOS and AOS representations, we employed objects of the same material and the same color, and omitted color information in the computation. Thus our representation of an object is formed by 6 features: the three main object dimensions D_1, D_2, D_3, SOS and AOS activation, and weight W.

Test objects

Object recognition tests were performed on nine objects of the Box class, i.e. objects 1, 2, 3, 4, 10, 11, 12 and 13 of Table 8.1, plus one additional object. The weight of the target object was estimated upon grasping and lifting it, performed according to a multimodal visual/tactile procedure [Grzyb et al., 2009], [Felip and Morales, 2009].

Results and discussion

We checked the behavior of our probabilistic linear classifier including different subsets of the features, for $n = 3$ and $n = 2$, as shown in Table 8.2. We are especially interested in two aspects: the significance and usefulness of the SOS and AOS features and the advantages of multimodal integration offered by the use of object weight. Comparing the first three lines of Table 8.2 we notice that the pair SOS, AOS is nearly as informative as the entire set of dimensions D_1, D_2, D_3, being their performances nearly equal (only about 1% difference in correct answers). This indicates that the way we modeled neural activation of CIP neurons is not only suitable to represent object features for action planning, but captures also the global shape of the objects employed in recognition. Nevertheless, not all visual information regarding target objects is contained in the SOS, AOS pair, as can be seen by the increased performance obtained adding one of

	n=3			n=2		
Feature set	**C**	**W**	**U**	**C**	**W**	**U**
SOS, AOS	72.3	26.8	0.9	65.9	25.7	8.4
D_1, D_2, D_3	73.3	23.0	3.7	66.7	17.6	15.7
SOS, AOS, D_1	77.1	21.9	1.0	68.7	15.7	15.6
SOS, AOS, W	70.1	16.4	13.5	59.2	11.5	29.3
SOS, AOS, D_1, D_2, D_3, W	78.7	0.8	20.5	57.6	0.0	42.4

Table 8.2: Classification results of probabilistic linear classifier. Percentages correct (**C**) and wrong (**W**), and of uncertain cases (**U**)

Feature set	**MLS**	**MN**	**ND**
SOS, AOS	42.0	72.7	78.2
D_1, D_2, D_3	42.0	77.1	80.7
SOS, AOS, D_1	42.8	50.7	77.9
SOS, AOS, W	58.0	75.7	97.9
SOS, AOS, D_1, D_2, D_3, W	70.2	73.1	98.0

Table 8.3: Classification results of Minimum Least Square (MLS), Nearest Mean (NM) and Normal Density (ND) classifiers. Percentages of correct classifications.

the dimensions, e.g. D_1, as in line 3 of Table 8.2. It is significant though that the triplet SOS, AOS, D_1 performs better than the simple set of dimensions, again reinforcing the idea that our modeled expressions do capture significant visual characteristics of objects. On the other hand, object recognition in the ventral stream is substantially size-invariant, so it is reasonable that information on absolute size offers only little additional advantage.

The above considerations can be confirmed looking at Fig. 8.6 , in which all the set of samples for the nine target objects is depicted on an SOS/AOS space. Again, while for some objects the two features are very informative and nearly enough to recognition, it is apparent that other objects require additional information to be resolved from each other. The graph shows also that there is a large variability in the representation of some objects, due to visual imprecisions. It is worth reminding on this regard that the samples were taken observing all objects from different canonical viewpoints, and this constitutes an important additional complexity in the recognition process.

Regarding multimodal recognition aided by object weight estimation, lines 4 and 5 of Table 8.2 show that the performance in term of correct (C) answers does not really improve. On the other hand, the number of wrong (W) answers is now much smaller (less than 1% for the whole feature set, line 5), and many more samples are classified as uncertain (U). The introduction of the W feature seems to provide the system with the

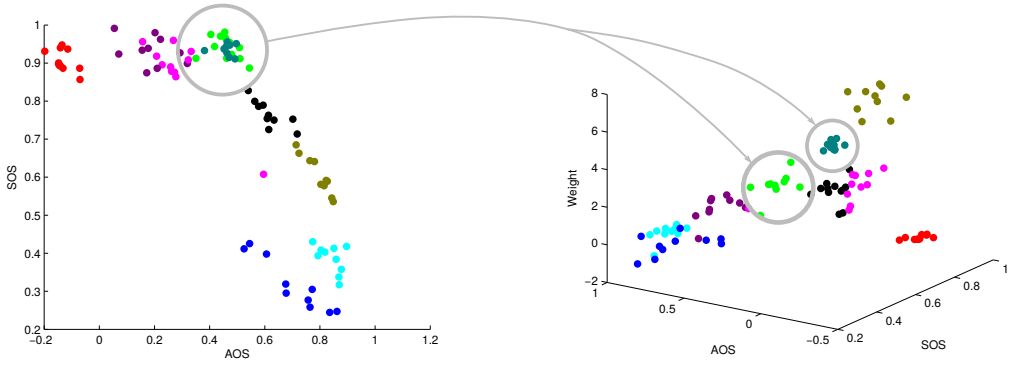


Figure 8.6: Distribution of object samples. Each color corresponds to samples of a different object. Right: Distribution of object samples plotted on a SOS/AOS feature space. Left: Distribution of object samples plotted on a SOS/AOS/W feature space. The grey circles and arrows show how adding the weight to the feature space it is possible to separate dark green and light green classes.

ability of detecting potentially problematic situations, in which the wiser decision is to avoid inserting the sample in any given class. The comparison in performance between the different values of n confirms the hypothesized behaviour, showing higher correct recognition values, but also more wrong answers, for $n = 3$, and more unclassified samples for $n = 2$. These results suggested us a new experimental scheme, in which n changes dynamically with the number of classified samples, starting from lower, more conservative values and growing ideally up to a value that grants no uncertain cases, once the set of objects has been fully learnt.

As the overall performance of the recognition system is not extremely good, we wanted to check whether this was due to the limits of our simple classifier or to the properties of the feature set. The graph in Fig. 8.6 also indicates that the triplet of dimensions SOS, AOS and W provide a high separability of the classes. In order to solve this issue, we applied other three classifiers to our set of features: Minimum Least Square (MLS), Nearest Mean (NM) and Normal Density (ND), from the Matlab PRTools4 Toolbox for pattern recognition [van der Heijden et al., 2004].

We did not consider classifiers that require to maintain a full memory of all encountered samples, such as k-nearest neighbours, for their lack of biological plausibility. The results of Table 8.3 show that at least one of the classifiers, ND, grant very high recognition rates, for all feature subsets. The performance of NM are comparable with our linear classifier, whilst MLS is definitely worse. Comparing again the different feature subsets, the pair SOS, AOS and the triplets D_1, D_2, D_3 and SOS, AOS, D_1 are approximately equivalent. The inclusion of W provides much better results (apart for the MN

classifier), and it is interesting to observe that for ND the subset SOS, AOS, W gives practically the same, extremely good performance that the whole set of features (97.9% against 98.0%). On the one hand, this confirms the appropriateness of the SOS-AOS representation to tackle the recognition problem, and the edge offered by multimodal processing and the use of object weight. On the other hand, the difference between our linear classifier and ND is not very large for the purely visual subset, but rises up to almost 28% in multimodal classification, suggesting that more complex tools are required to take full advantage of its potentialities.

We have also implemented an incremental version of the learning algorithm, in which, if a sample is classified, it is directly added to the classifier memory to be used in subsequent tests, and mean and variance are immediately recalculated. Unclassified instances are ignored, unless a human supervisor is available. In this case, he/she is asked to label unidentified samples so they can be added to the memory. Thus, in this implementation the module keeps learning while recognizing objects, and the more samples the system can include in its "knowledge" of the world, the more robust its classification becomes, and the approximation of the average to the real value of the feature set improves.

8.6 Conclusion

In this chapter we proposed and implemented a theoretical scheme for hierarchical object recognition inspired by primate brain mechanisms. The scheme proposed is based on a three step process. We implemented the two first steps, shape classification and object recognition on a real robot setup, achieving good results in both tasks.

Distinguishing features of our approach are: 1) the use of typical dorsal processing information, such as SOS and AOS activations, in a ventral visual task, implementing a possible link between the cortical visual streams; 2) multimodal integration by including object weight in the recognition process.

However, we have considered a reduced universe of objects and the representation used to encode them (i.e. contour curvedness) should be tested on a broader set of objects to validate its suitability for a real world scenario.

Current and future research include: 1) a dynamical learning framework for the object recognition step, in which the agent gradually increases its confidence in classifying new samples, and thus increasingly improve its knowledge of the world; 2) a return projection from the ventral stream to dorsal areas, in which remember object identity contributes in properly configuring the hand during grasping actions; 3) enhancement of initial visual processing to get rid of the color and background assumptions; 4) integration into the contact based manipulation framework.

The experiments and implementation presented in this chapter were performed before the development of the architecture presented through this thesis. However, the integration of the work presented in this chapter into the system was taken into consideration and it can be done through the use of perceptual primitives.

The research leading to the results presented in this chapter was published in [Chinellato et al., 2011]. The work presented in this chapter is the result of the collaboration with Eris Chinellato. The computational model of the brain shown in Figure 8.2, the implementation of the AOS and SOS activation and the object categorization using the shape curvedness descriptors are part of his PhD. Thesis [Chinellato, 2008].

Chapter 9

Conclusion

Autonomous robotic manipulation in unstructured environments is still one of the grand challenges in robotics. In order to manipulate an object, the robot has to perform several steps, each of them a research area itself. First, the robot has to look for the object in the environment and determine its position. Second, it has to plan how to manipulate it considering the task constraints and the environment. Finally, it has to execute the planned action adapting to prediction mismatches and possible external interferences. In this thesis, we have made use of state of the art algorithms for the two first steps and developed sensor-based approaches for the last step.

Nowadays, there are a lot of manipulator robots that have proven good manipulation skills. However, the solutions available usually deal with small subsets of the problem. Robots are still far from being close to the scene understanding and dexterity of humans in such scenarios. In this thesis we have taken inspiration from human grasping experiments to implement a system capable to perform manipulation tasks in unstructured environments. In the implemented reactive contact based manipulation system: sensory feedback, adaptive control, contact detection, contact prediction, object detection and object recognition are key. Although there are also assumptions and constraints on the algorithms and approaches presented, we believe that they will be slowly but steady removed in the future.

Regarding the limitations and possible extensions of the work presented, an important feature that a robot shall have is the ability to adapt to the environment and learn from its interaction with the real world. However, in the system detailed in this thesis, learning is not present and should be incorporated in the future. Learning can be incorporated at the different levels of abstraction, the guidelines to add learning capabilities to the robot are discussed in Chapters 3 and 5 and summarized in Section 9.2.

Even with the organization and structure that the presented architecture provides, it is difficult to prepare new experiments and execute new tasks. Although task definitions are easily created, the connections of the services, primitives, and the configuration files, force the user to have a deep knowledge of the existing modules. Learning approaches could be used to mitigate this problem enabling to program tasks by demonstration.

In this thesis we have implemented and validated a contact driven robotic manipulation system. First, in Chapter 2 we have taken inspiration from neuroscience studies about human sensorimotor control of manipulation and identified the key components of a contact driven manipulation system. Second, in Chapter 3 we have presented, implemented and validated the manipulation primitives paradigm, a vocabulary of simple sensor-based manipulation actions that are combined to perform complex tasks. Third, we have developed and validated the mechanisms for contact event detection and prediction in Chapters 4 and 5 respectively. Fourth, in Chapter 6 we have presented the software architecture created to integrate all the pieces of the system and some useful abilities such as grasp planning. Fifth, in Chapter 7 an abstraction mechanism that allows the same tasks to be used by different robotic platforms has been presented. Finally, to endow the contact-based manipulation system with visual perception of objects we have presented in Chapter 8 a hierarchical object recognition system based on primate brain mechanisms. Details about the robotic platforms used to develop the work presented in this thesis are provided in Appendix A.

9.1 Contributions

The main contributions of the work presented through this thesis are listed below:

- The main contribution is the development of the manipulation primitive paradigm. A framework to implement and specify atomic reactive actions that can be used as building blocks to define more complex tasks.
- The implementation of a reactive strategy for grasping objects. The value of adaptive sensor-based control strategies is validated through the implementation and testing of the robust grasp controller.
- A complete pipeline to unscrew bottle caps is presented as a use-case of the manipulation primitives framework, a reactive unscrew primitive is also implemented and validated.
- A novel method to detect contacts using the robot model, current arm motion information and RGBD images is provided.
- A sensor fusion framework focused on contact detection and localization is another of the contributions of this thesis. The implementation of several sensor, context and prediction cues into the framework is also provided.
- Study and implementation of robot dynamic simulation to predict the robot-object interaction and the contacts that arise from it.
- An action abstraction architecture using the manipulation primitives paradigm in order to transfer plans between different platforms with different embodiments.

- The control architecture that orchestrates all the system components in a four-layered fashion.

9.2 Open questions and future work

The development of this thesis, has produced several related publications and tried to tackle some of the frequent problems in the robotic manipulation. However, as written in the following paragraphs, each of the aspects presented in this thesis can be improved. Beyond the improvements, the work presented opens several opportunities for future work and propose several open questions that would enhance and expand the research conducted throughout this thesis.

Human grasping experiments

In the neuroscience studies reviewed in Chapter 2, we have highlighted that humans perform corrective movements when there is a mismatch between predicted and perceived sensory input. However, there are no experiments available in the literature that study in detail how those corrections are performed. In this thesis we have performed preliminary experiments to observe how humans perform corrections, nevertheless to better understand and determine how humans adapt to different unexpected situations, more experiments are necessary.

Manipulation primitives

Although in Chapter 3 we have proposed and implemented a set of manipulation primitives, there are still more primitives that should be identified and implemented in order to increase the range of tasks that can be described. Some of the missing primitives are already identified (push, pull) but there might be others related to more specific environments (e.g. cooking) that are still unknown.

An interesting open question that should be investigated is the suitability of learning techniques to replace the implementation of manipulation primitives. Action-phase controllers could be learned instead of programmed. However, as suggested by [Johansson and Flanagan, 2010], corrective movements are learned together with each action-phase controller. Therefore, using learning to acquire new skills would require to learn corrective movements as well. Corrections are triggered by prediction mismatches, hence a prediction mechanism should be introduced into the learning scheme. A possible solution to learn this kind of controllers was proposed by [Pastor et al., 2011], where a grasping manipulation primitive is learned together with sensorimotor memories and corrective movements are performed when prediction errors arise. Unfortunately it is not clear how the learned strategy would generalize for other grasping tasks with different environment, objects and hand configuration.

Perception

During manipulation tasks, humans detect contact events using multi-modal information from different sensory cues [Johansson and Flanagan, 2010]. In Chapter 4 we have shown a sensor fusion framework that gathers contact information from different sources and provides the estimates of the detected contact events.

To the best knowledge of the author, in the literature there are no experiments regarding how the sensor fusion is performed by humans. Whether there is precedence of a sensory cue (e.g. tactile) over other cues (e.g. vision or predictions) is still unknown. Intuitively, our implementation of the sensor fusion mechanism uses a probabilistic approach and the priorities of the sensory cues depend on the confidence of each contact hypotheses generator. Experiments focused on this aspect of human manipulation would be very valuable to improve the sensor fusion methods and the contact detection.

Prediction

The current set-up uses the simulation to foretell where and when contacts will arise. Thus, if the simulator is able to predict when and where contacts will happen, it can be effectively used in the proposed human-inspired contact-based manipulation system. However, the real-time accurate physical simulation required is still not ready. The continuous development of physics engines makes it difficult to select the best engine for our prediction engine. In this thesis we have selected ODE but in the future there might be other engines with better performance. Hence, the implementation of a physics abstraction layer would be very valuable for the prediction engine. However, to obtain accurate simulations, a lot of unknown parameters regarding the object material and inertial properties are required. Furthermore, accurate simulation is computationally too expensive.

The prediction problem can also be approached from different points of view, instead of using a dynamics simulator, the consequences of object interactions can be learned by the robot [Belter et al., 2014]. Another approach is to obtain sensorimotor memories from a successful task execution and use them to monitor and drive the task execution [Pastor et al., 2011]. Hence, an approach that combines sensorimotor memories, learning and dynamic simulation could be the hybrid solution to the prediction problem. First, physics simulation parameters can be tuned by the execution of exploratory interactions with the object. Second, the simulation can be used to provide training data. Finally, the learned sensorimotor memory can be used to drive the execution of tasks and trigger corrections when mismatches are detected.

High level task planning

The task definitions used for the experiments presented in this thesis were manually created. We think that the manipulation primitives vocabulary, together with the perceptual primitives can be used as the base symbols for a higher level task planner that

can convert semantically meaningful orders such as "clear the table" or "mop the floor" into executable task definitions. As recently published by [Yang et al., 2015] such task representations can be automatically obtained from unlabelled video sequences. However, from a single example, it is difficult to obtain the parameters and constraints to tune the task for a specific scenario. Once the task descriptions can be automatically learned from examples, a method to adapt the learned task description to different scenarios will be required. With enough examples of the same task, imitation learning provides methods that could be used for that purpose.

9.3 Publications

Parts of this thesis have previously been published in the following journal and conference papers:

1. Felip, J. and Morales, A. (2009). Robust sensor-based grasp primitive for a three-finger robot hand. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 1811–1816
2. Laaksonen, J., Felip, J., Morales, A., and Kyrki, V. (2010). Embodiment independent manipulation through action abstraction. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2113–2118
3. Chinellato, E., Felip, J., Grzyb, B. J., Morales, A., and del Pobil, A. P. (2011). Hierarchical object recognition inspired by primate brain mechanisms. In *Computational Intelligence for Visual Intelligence (CIVI), 2011 IEEE Workshop on*, pages 1–8
4. Bohg, J., Johnson-Roberson, M., Leon, B., Felip, J., Gratal, X., Bergstrom, N., Kragic, D., and Morales, A. (2011). Mind the gap - robotic grasping under incomplete observation. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 686–693
5. Felip, J., Bernabe, J., and Morales, A. (2012). Contact-based blind grasping of unknown objects. In *12th IEEE-RAS International Conference on Humanoid Robots, HUMANOIDS*, pages 396–401
6. Felip, J., Laaksonen, J., Morales, A., and Kyrki, V. (2013). Manipulation primitives: A paradigm for abstraction and execution of grasping and manipulation tasks. *Robotics and Autonomous Systems*, 61(3):283 – 296
7. Leon, B., Felip, J., Marti, H., and Morales, A. (2012). Simulation of robot dynamics for grasping and manipulation tasks. In *12th IEEE-RAS International Conference on Humanoid Robots, HUMANOIDS*, pages 291–296
8. del Pobil, A. P., Duran, A. J., Antonelli, M., Felip, J., Morales, A., Prats, M., and Chinellato, E. (2013). Integration of visuomotor learning, cognitive grasping and

sensor-based physical interaction in the uji humanoid torso. *Designing Intelligent Robots: Reintegrating AI*, pages pp. 6–11

9. Bernabe, J., Felip., J., del Pobil, A. P., and Morales, A. (2013). Contact localization through robot and object motion from point clouds. In *13th IEEE-RAS International Conference on Humanoid Robots, HUMANOIDS*, Atlanta, GA, USA. IEEE

10. Morales, A., Prats, M., and Felip, J. (2013). Sensors and methods for the evaluation of grasping. In Carbone, G., editor, *Grasping in Robotics*, volume 10 of *Mechanisms and Machine Science*, pages 77–104. Springer London

11. Felip, J. and Morales, A. (2014). Dual arm sensor-based controller for the cap unscrewing task. In *14th IEEE-RAS International Conference on Humanoid Robots, HUMANOIDS*, Madrid, Spain. IEEE

12. Felip, J., Morales, A., and Asfour, T. (2014). Multi-sensor and prediction fusion for contact detection and localization. In *14th IEEE-RAS International Conference on Humanoid Robots, HUMANOIDS*, Madrid, Spain. IEEE

13. Felip, J., Durán, A. J., Antonelli, M., Morales, A., and del Pobil, A. P. (2015). Tombatossals: A humanoid torso for autonomous sensor-based task execution research. In *15th IEEE-RAS International Conference on Humanoid Robots, HUMANOIDS*, Seoul, South Korea. IEEE

Appendices

Appendix A

Robotic platforms

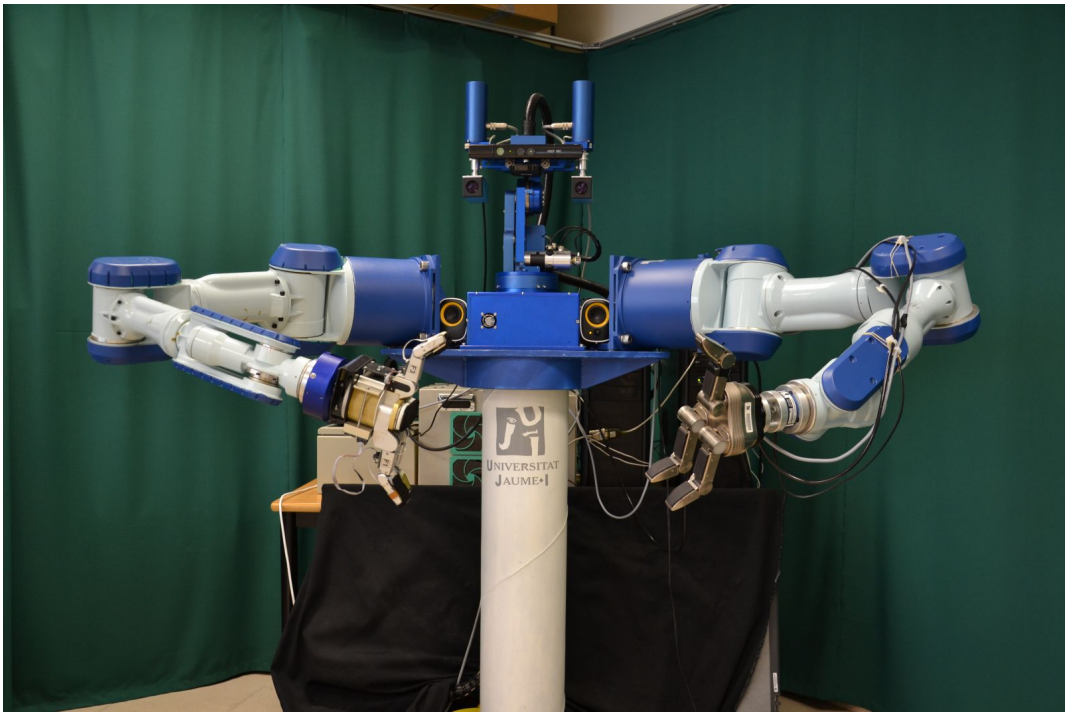


Figure A.1: Tombatossals: The UJI humanoid torso.

A.1 Tombatossals. The UJI humanoid torso

The humanoid torso is composed of two arms, two hands and a head for a total of 29 DOF. Three desktop computers are used for control and processing. The robot is depicted in Fig. A.1. This platform was built and enhanced during the development of this thesis and has been the main platform used to perform the research and the experiments presented through this thesis. All the chapters of this thesis have used this platform for experimental validation.

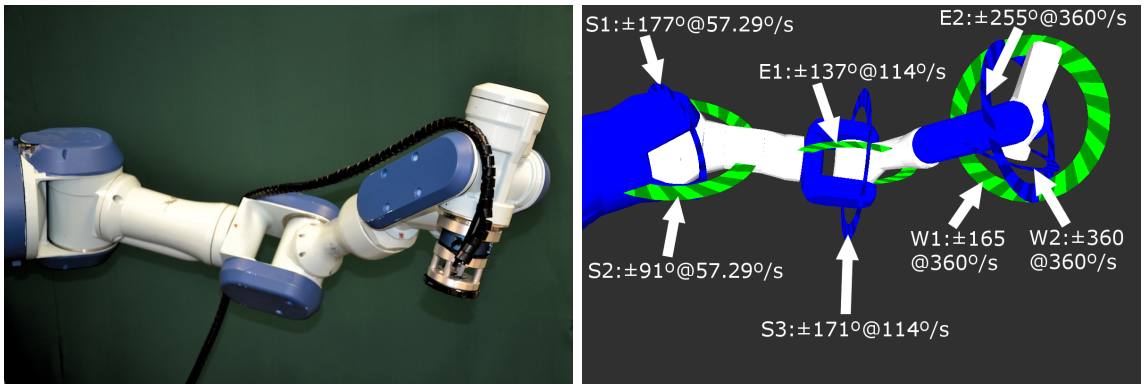


Figure A.2: Left: PA10-7C 7 DOFs arm. Right: Arm model and joints with their angle and speed limits.

A.1.1 The arms

Both arms are Mitsubishi PA10-7C, 7 DOF industrial manipulators with a position repeatability of ±0.1mm. Each arm weights 40Kg and has 10Kg payload. Taking into account the hands, force sensors and tool adapters, the remaining payload is 7.2Kg for the left hand and 7.8Kg for the right hand. Joints can be controlled in position, velocity and effort. The joint names, limits and speed are depicted in Fig A.2. The arms are placed in the same horizontal plane with an aperture angle of 120° (Fig. A.1). See [Elbrechter et al., 2012] for an alternative configuration using these same arms.

A.1.2 The hands and contact sensors

Barrett Hand

It is an under-actuated 3-fingered hand with 4 actuated DOF that can be controlled either in velocity or in position. Each finger has two coupled joints actuated by one motor. The other DOF drives the opposition of two of the fingers. The joint angles and actuation speed are depicted in Fig.A.3. Each finger has an integrated strain-gauge sensor that provides the torque applied to its distal phalanx.

Schunk SDH2 Hand

It is a fully actuated 3-finger hand with 7 DOF, 2 DOF for each finger and 1 DOF to pivot contrary-wise two of the fingers. Joint limits are ±90°for each joint and 210°/s speed. As it can be seen in Fig.A.4 the opposition of the fingers is limited to 90°. Thus, this hand can oppose two of the fingers but cannot perform a hook grasp.

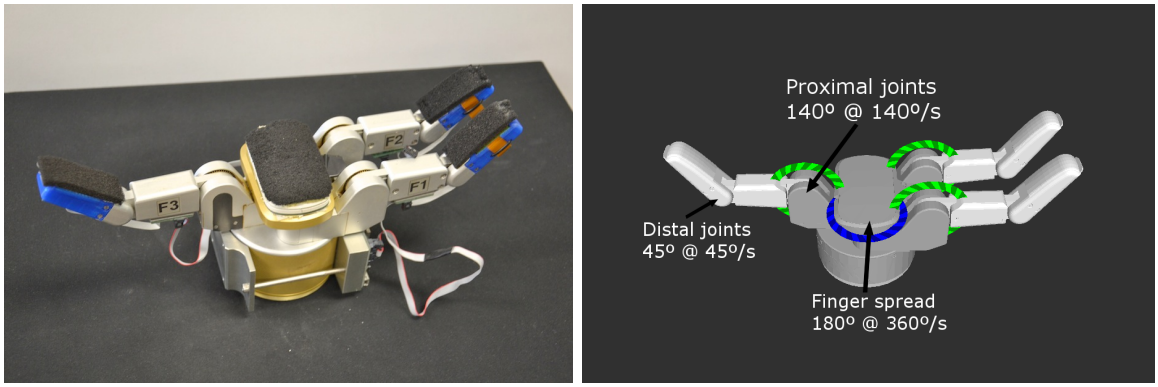


Figure A.3: Left: Underactuated 4 DOFs hand Barrett Hand upgraded with Weiss Robotics tactile sensors. The distal phalanxes are modified for a better integration of the sensors. Right: Hand model and joints with their angle and speed limits.

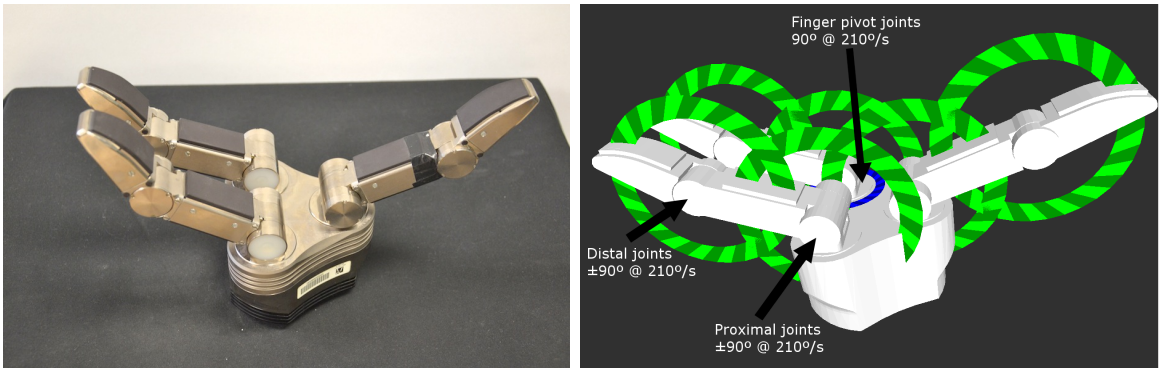


Figure A.4: Left: 7 DOFs Schunk Hand with Weiss Robotics tactile sensors. Right: SDH2 Hand model and joints with their angle and speed limits.

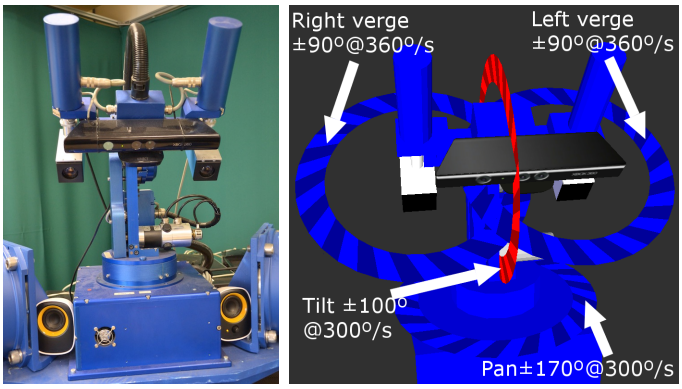


Figure A.5: Left: TO-40 pan-tilt-verge head. Right: Head model and joints with their position and velocity limits.

Sensors

The right hand (Barrett Hand) is upgraded with Weiss Robotics[1] resistive tactile sensors mounted on the palm and on the distal phalanxes. Tactile sensor pads are custom arrays of 5x8 pressure sensors for the phalanxes and 6x14 for the palm. The distal phalanxes are modified for a better integration of the sensors, (see Fig.A.3). The left hand (SDH2) has Weiss Robotics tactile sensors already integrated on the proximal and distal phalanxes. See black patches in Fig.A.4. Tactile sensors have a sampling rate up to 230Hz.

Between each hand and its arm there is a JR3[2] 6 axis force-torque sensor. The force-torque sensors on each wrist provide the other modality of contact sensing, their sample rate can be up to 200Hz.

A.1.3 The head and camera setup

The head is composed of a TO40 pan-tilt-vergence system and a Kinect™. Head joints are depicted in Fig.A.5. The TO40 is a 4 DOFs head with two DFK 31BF03-Z2 cameras. The motorized zoom allows the control of the focal length from 5mm to 45mm. The cameras have a resolution of 1024x768@30fps. The baseline between cameras is 27 cm. The Kinect ™ provides RGB and Depth images with a resolution of 640x480@30fps.

A.1.4 Computers

The robot sensors and actuators are connected to three different computers. The computers are physically connected to each other through a Gigabit Ethernet switch. The communication is handled by ROS, and the computer where the algorithms are running is transparent to the programmer. However, in order to balance the load, each computer

[1]Weiss Robotics sensors. `http://www.weiss-robotics.de/`
[2]JR3 Force-torque sensors. `http://www.jr3.com/`

Main task	Processor	RAM	GPU
2D Vision	Intel E8400 @3.00GHz	8Gb	560GTX 1Gb
3D Vision	Intel i5 650 @3.20GHz	8Gb	580GTX 1Gb
Control	Intel Q9550 @2.83GHz	8Gb	9800GT 512Mb

Table A.1: Hardware specs of each computer

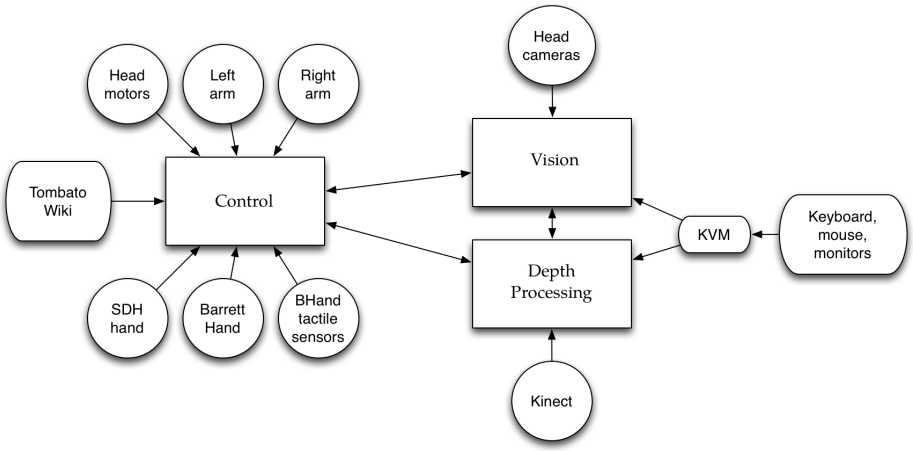


Figure A.6: Tombatossals computer layout. It is composed by three computers: Control, Vision and Depth Processing.

copes with a specific task. It is important that modules that require high bandwidth data sources (e.g. cameras, depth sensors) run on computers that have direct access to those sources. In Fig. A.6 we show how the sensors and actuators are connected and the role of each computer depending on the sensors that are directly available for that computer. However, other roles such as task management or visualization do not have a computer assigned and can be run transparently on any machine. The description of the computers and their main tasks are shown in table A.1.

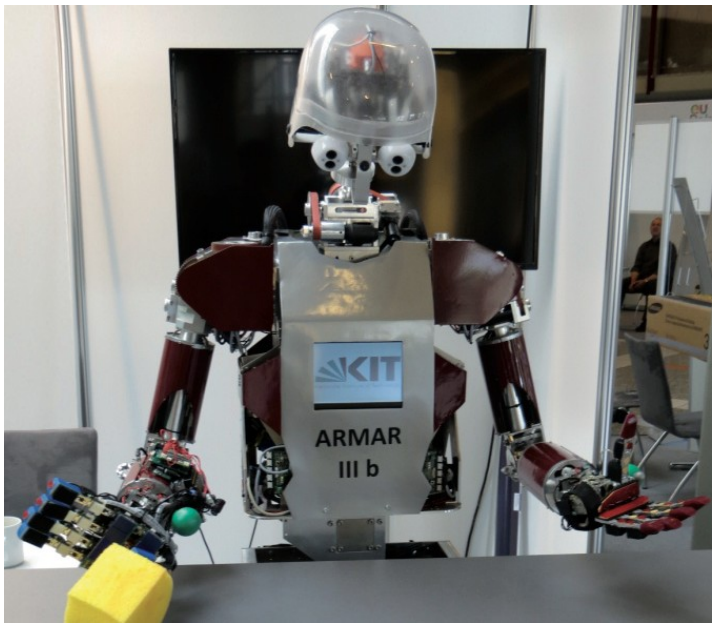


Figure A.7: ARMAR-IIIb a humanoid robot with 43 DOF.

A.2 ARMAR IIIb

ARMAR-IIIa was designed and built in 2006 by the Karlsruhe Institute of Technology, its design closely mimics the sensory and sensorimotor capabilities of the human.

The robot was designed to deal with a household environment and the wide variety of objects and activities encountered in it. ARMAR-IIIa is a fully integrated autonomous humanoid system. It has a total 43 DOF and is equipped with position, velocity and force-torque sensors. The upper body has been designed to be modular and light-weight while retaining similar size and proportion as an average person. For the locomotion, a holonomic mobile platform is used. Two years later, a slightly improved humanoid robot, ARMAR-IIIb (shown in Fig. A.7), was engineered. Detailed information about the robot can be found in [Asfour et al., 2006], where most of the information summarized in this section was extracted from.

This platform was used in this thesis for the research and implementation of the work presented in Chapter 4 during the 4 month research stay at the Karlsruhe Institute of Technology in 2012.

A.2.1 The arms

The arms are designed in an anthropomorphic way: three DOF in the shoulder, two DOF in the elbow and two DOF in the wrist for a total of 7 DOF. The design of the

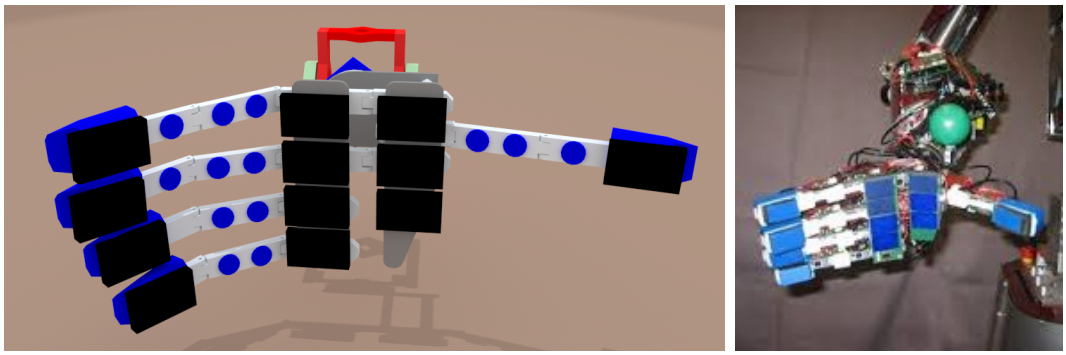


Figure A.8: The Karlsruhe Humanoid Hand. Left: Hand model with black patches showing the tactile sensors. Right: Hand picture.

arms is based on the observation of the motion range of a human arm. Motors can be position, velocity and torque controlled.

A.2.2 The hands and contact sensors

Each arm is equipped with a five-fingered hand with eight actuated DOF. The hand is under-actuated, each finger has 2 DOF and the palm another one. The thumb, index and middle fingers have their 2 DOF actuated, the ring and pinkie are coupled and only have 1 DOF, the last DOF controls the palm. The hand is actuated using compressed air and valves, the position of the actuated joints can be controlled using the feedback provided by the joint encoders. However, when grasping objects or applying forces to the environment with the fingers the exact position cannot be determined only by the encoders.

Sensors

For tactile feedback during manipulation operations, a former version of the Weiss tactile sensors mounted on Tombatossals' Barrett Hand are used in ARMAR-IIIb. The sensors are mounted on the distal phalanxes of each finger and on the palm as depicted in Fig. A.8. The detailed description of the tactile sensors developed for the robotic hand was published in [Kerpa et al., 2003], however the sensors were improved and the first commercial version was provided by Weiss Robotics GmbH & Co.KG[3]. Nowadays, these sensors have evolved and are used in many robotics applications in industry and research. In the wrist, 6D force/torque sensors from ATI Industrial Automation[4] are used.

[3]Weiss Robotics GmbH & Co.KG: `http://www.weiss-robotics.de/en/`
[4]ATI Industrial Automation: `www.ati-ia.com`

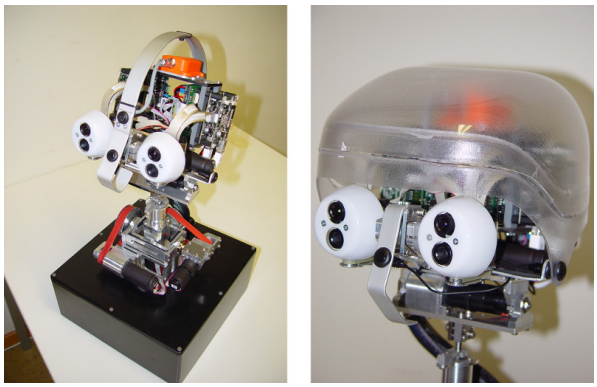


Figure A.9: The Karlsruhe Humanoid Head.

A.2.3 The head and camera setup

ARMAR-IIIb uses a Karlsruhe humanoid head. It possesses two cameras per eye with a wide-angle lens for peripheral vision and a narrow-angle lens for foveated vision. It has a total number of 7 DOF (4 in the neck and 3 in the eyes), six microphones and a 6D inertial sensor. Throughout Europe, there are already ten copies of this head in use. The details about the head mechanism, sensors and control are provided in [Asfour et al., 2008].

A.2.4 Computers

There are five computers inside the robot that are devoted to different tasks. The computers are separated in a three layered architecture: task execution, task coordination and task planning. Each computer has different roles assigned, audio processing and synthesis, visual perception, platform control and navigation, coordination, position and torque motor control. The computer layout is the same as detailed in [Asfour et al., 2008] but the computers have been recently updated to Intel i5 processors for powerful onboard computational capabilities.

The computers are running under Linux, with the Real Time Application Interface RTAI/LXRT-Linux. They are interconnected through a gigabit ethernet network. For the implementation of the control architecture and interprocess communications, the MCA2[5] framework is used.

[5]MCA2: `http://www.mca2.org/`

Figure A.10: Industrial manipulator setup composed of a 6DOF Mitsubishi Melfa RV-3SB 6DOF industrial manipulator and a Schunk PG70 2-Finger Parallel Gripper.

A.3 Mitsubishi Melfa RV-3SB arm

The robotic setup available at the Lappeenranta University of Technology (LUT), consists of a 6DOF Mitsubishi Melfa RV-3SB industrial manipulator and a WRT-102 gripper from Weiss Robotics, see Fig. A.10.

The Melfa RV-3SB is an industrial manipulator with 3Kg payload and a position repeatability of ± 0.02mm. It weights 37Kg and the speed of motion of the joints varies from 187 to 660 degrees per second depending on the joint. The WRT-102 gripper is based on the PG-70 2-Finger Parallel Gripper from Schunk but has tactile sensors on both fingers. Between the gripper and the arm there is a 6DOF JR3 force-torque sensor.

This platform was used for the embodiment abstraction experiments performed in Chapter 7 as one of the results of the GRASP Project funded by the European Commission under the FP7 programme.

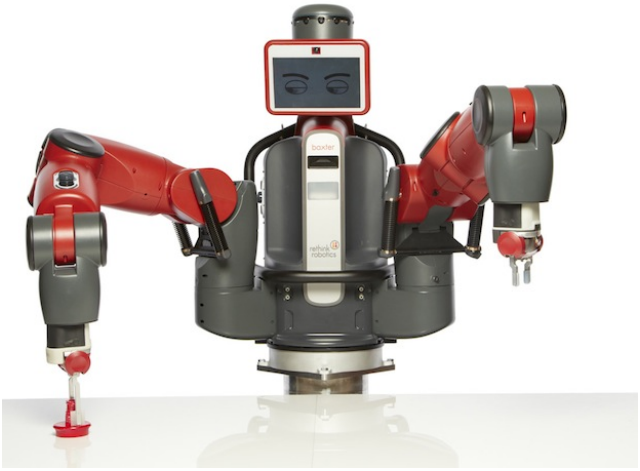


Figure A.11: Baxter collaborative robot.

A.4 Baxter

Baxter is a commercial compliant low-cost manipulator torso manufactured by Rethink Robotics. It features a dual-arm configuration very similar to Tombatossals. With a total weight of 138.79 Kg. This platform was used for the software architecture implementation presented in Chapter 6 and the participation in the APC 2015 by the RobInLab team.

A.4.1 The arms

The arms of Baxter (see Fig. A.12) weigh 21.3 Kg, have 7DOF and a payload of 2.2 Kg already taking into consideration the grippers included with the robot. Although, the manufacturer does not provide information about its position repeatability, it is well known that Baxter's arms are not precision manipulators, some users of the robot have reported that its repeatability is around ±3mm. The arms are compliant and they have a safety mechanism that automatically loosens the arms when the perceived external force is over a certain limit. The joints can be controlled in position, velocity or torque.

A.4.2 The grippers

The gripper provided by the manufacturer has a very small range of movement and cannot grasp the wide variety of objects that are present in household scenarios. Given the 2.2Kg payload of the arms, using a commercial hand such as SDH2 or Barrett Hand, is not an option as they weigh around 2Kg. Inspired by the Festo Fin Ray gripper[6], we have developed our own low-cost and light-weight gripper for Baxter, see Fig. A.13.

[6]Festo Fin Ray gripper: `https://www.festo.com/cms/en_corp/9779.htm`

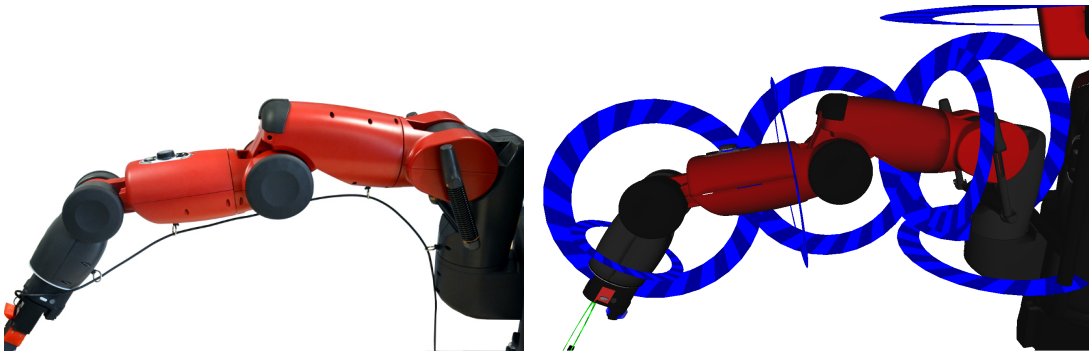


Figure A.12: Left: Baxter robotic arm. Right: Baxter robotic arm model and joints.

Sensors

The controller of the arms provides a virtual 6D force-torque sensor using the computed external forces at the end effector. Although the measurements are very noisy and not accurate, they are useful enough for contact detection and safety corrections. In each end effector there is an embedded fisheye camera that provides up to 1280x800 images at 30Hz and a IR range sensor.

A.4.3 Head and camera setup

The head consists of a screen attached to a pan and tilt mechanism. The pan joint angle can be controlled, however the tilt joint has only two positions, up and down. The screen has an integrated fisheye camera. On top of the head there is a ring of sonar distance sensors that are used to detect the presence of people in the vicinity, the robot head is depicted in Fig. A.14. In order to enhance the robot perceptual capabilities and be able to explore all the bins of the APC shelf, we developed a kinect adapter for the robot elbow.

A.4.4 Computers

The Baxter robot has an embedded computer with a 3rd Gen Intel Core i7-3770 Processor. It can work standalone but for research and more demanding applications it can be connected to an external network with a Gigabit Ethernet cable. The robot is controlled natively using ROS, thus a computer network like the one used for Tombatossals or ARMAR-IIIb can be easily set-up and used to distribute the computational expensive modules over different computers.

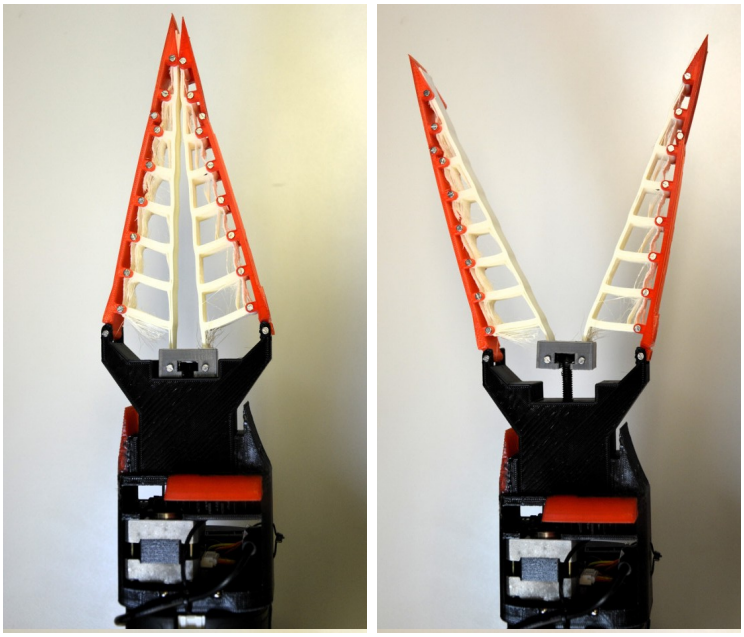


Figure A.13: Low-cost light-weight adaptive grippers for Baxter, developed for the Amazon Picking Challenge by the RobInLab team.

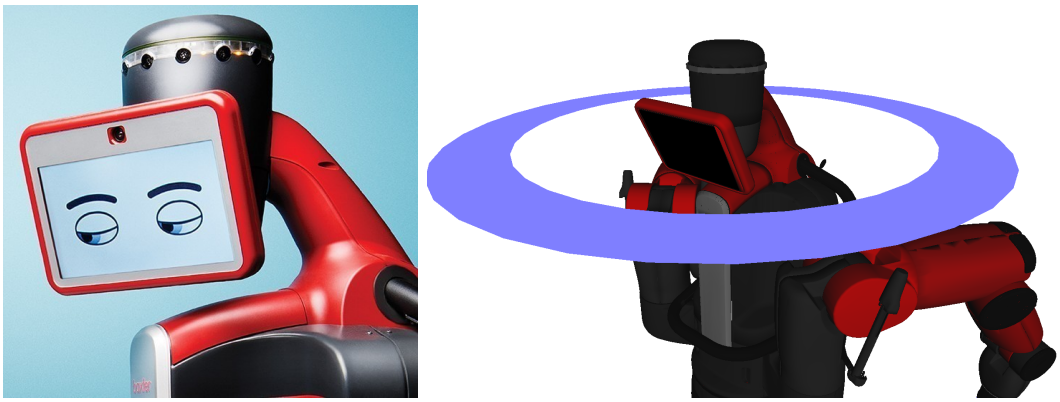


Figure A.14: Left: Baxter's pan-nod head. Mounted on the pan-nod mechanism there is a screen with a camera. On top of the head there is an array of sonar distance sensors. Right: Baxter head model and controllable pan joint.

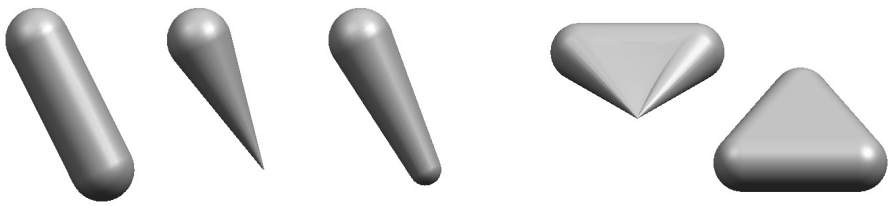
Appendix B

Robot spherical modelling

A geometric model of the robot can be used to reason about the space occupied by it and estimate contacts with objects. It is also a very useful tool to segment out the robot from an image or from a point cloud.

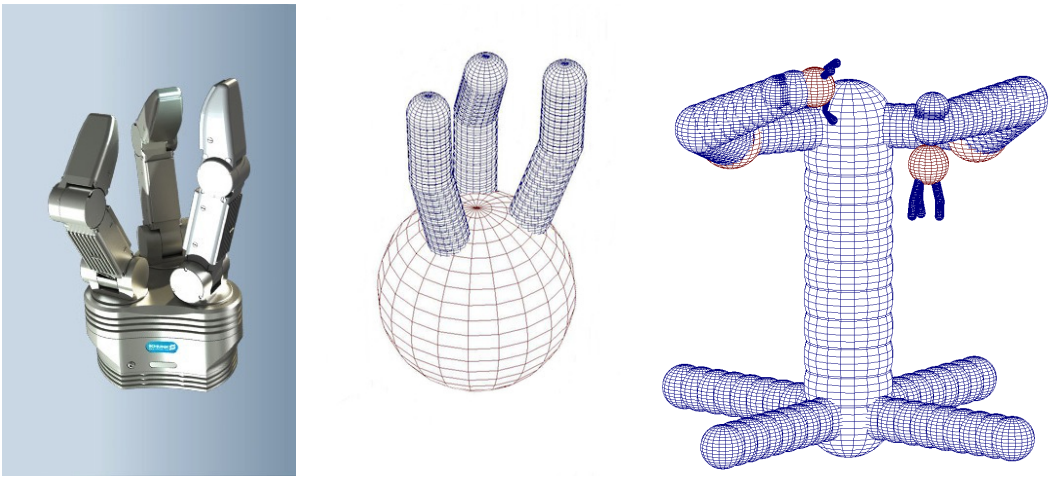
To model a robot, a model based in bounding volume primitives, can be used. For the model presented here, the spherically extended polytopes (*s-topes*) are used. This representation has been widely used [Tornero et al., 1991, del Pobil and Serna, 1995, Gilbert et al., 1988] because of its efficiency in distance computation, specifically in collision detection and path planning. An *s-tope* [Hamlin et al., 1992] is the convex hull of a finite set of spheres $s \equiv (c, r)$, where c is the centre and r its radius. Given the set of n spheres $S = \{s_0, s_1, ..., s_n\}$, the convex hull of such a set, S_s, contains an infinite set of swept spheres expressed by Eq. B.1. Where λ_i is the parameter that determines a specific sphere, radius and centre, of the whole set of spheres.

$$S_s = \left\{ s : s = s_0 + \sum_{i=0}^{n} \lambda_i (s_i - s_0), s_i \in S, \lambda_i \geq 0, \sum_{i=0}^{n} \lambda_i \leq 1 \right\} \tag{B.1}$$



(a) s-tope with two spheres, bi-sphere (b) s-tope with three spheres, tri-sphere

Figure B.1: Examples of simple s-topes: bi-spheres and tri-spheres



(a) Shunk Dextrous Hand (SDH) (b) SDH spherical model (c) Spherical model of our robotic system

Figure B.2: Spherical model of hand and robot

To illustrate the previous equation, Figure B.1 depicts several examples of s-topes defined by two (*bi-spheres*) and three spheres (*tri-spheres*).

We have modelled our robot as a combination of s-topes. Each link is represented as a bi-sphere and some static parts (i.e. hand palm) as single spheres. In addition, each defining sphere has been attached to the corresponding frame of the kinematic chain. Figure B.2(c) depicts the complete model of our robot manipulator system and Figure B.2(b) illustrates a detail with the model of our three-fingered hand.

The distance of a point to the spherical model is calculated as the minimum distance from the point to all the s-topes that compose the robot model. Since our geometric model is composed only of spheres and bi-spheres, we need to apply only two rules to compute each distance. For a single sphere, the distance between a point p_i and the sphere $s \equiv (c, r)$ is computed using Eq.B.2, where c is the centre and r the radius:

$$distance = \|p_i - c\| - r \tag{B.2}$$

The distance between a point p_i and a bi-sphere is calculated as follows: first we need to determine the closest sphere center to the point among the infinite number which define the bi-sphere. Given a bi-sphere defined by the spheres $s_1 \equiv (c_1, r_1)$ and $s_2 \equiv (c_2, r_2)$, Eq. B.3 defines the rule to find the closest sphere $s_{min} \equiv (c_{min}, r_{min})$ to p_i. If $\lambda < 0$ the first sphere is used. Then, Eq. B.2 can be used to compute the distance.

$$\begin{aligned} \lambda_{min} &= -\frac{(c_1 - p_i) \cdot (c_2 - c_1)}{\|c_2 - c_1\|^2}; \quad \lambda_{min} \in [0, 1] \\ c_{min} &= p_i - c_1 + \lambda_{min}(c_2 - c_1) \\ r_{min} &= p_i - r_1 + \lambda_{min}(r_2 - r_1) \end{aligned} \tag{B.3}$$

Acronyms

AIP Anterior Intraparietal Sulcus. 158, 162

ANNs Artificial Neural Networks. 144

AOS Axis Orientation Selective. 162, 165, 170–174

APC Amazon Picking Challenge. 2, 5, 138, 192, *Glossary:* Amazon Picking Challenge

API Application Programming Interface. 103

AVs Approach Vectors. 133, 136

BSD Berkeley Software Distribution. 103, *Glossary:* BSD

CCG Combinatory Categorical Grammar. 145

CIP Caudal Intraparietal Sulcus. 160, 162, 170

CNS Central Nervous System. 18

COLLADA COLLAborative Design Activity. 103, *Glossary:* COLLADA

DMP Dynamic Movement Primitive (DMP). 30

DOF Degrees of Freedom (DOF). 4

DRC DARPA Robotics Challenge. 4, *Glossary:* DARPA Robotics Challenge

GPL General Public License. 103, *Glossary:* GPL

HMMs Hidden Markov Models. 144

ICP Iterative Closest Point. 81

LOC Lateral-Occipital Complex. 159–163

MLS Minimum Least Square. 172

ND Normal Density. 172, 173

NM Nearest Mean. 172

ODE Open Dynamics Engine. 105

OGM Occupancy Grid Map. 79, 81, 82

OMPL Open Motion Planning Library. 132

PbD Programming by demonstration. 144

PCA Principal Component Analysis. 11, 133

RBFs Radial Basis Functions. 144

ROI Region Of Interest. 132

ROS Robot Operating System. 105, 106, 123, 124, *Glossary:* ROS

SOS Surface Orientation Selective. 162, 165, 170–174

V1 Primary Visual Cortex. 159, *Glossary:* V1

V2 Secondary Visual Cortex. 160, *Glossary:* V2

V3 Third Visual Complex. 160, 161, *Glossary:* V3

V4 Visual Area V4. 160, 161, 163, 164, *Glossary:* V4

YAML YAML Ain't Markup Language. 128

YARP Yet Another Robot Platform. 123

Glossary

action-phase controllers object manipulation tasks typically involve a series of action phases in which objects are grasped, moved, brought into contact with other objects and released. Each phase accomplishes a specific goal or subgoal of the task. 16

Amazon Picking Challenge a robotic grasping competition organized by Amazon. The robots, being in front of a shelf, like the ones used in Amazon warehouses, had to autonomously grasp a set of objects and place them into an order bin. The robot is given a file with a list of the target objects and it has to autonomously search, pick and place them into the order bin. There is a scoring system based on the number of objects retrieved, penalty points are received if the wrong object is picked or for each object dropped. More information about the rules and the past edition of the contest can be found at `http://amazonpickingchallenge.org/`. 2, 199

Biomimicry a field of study that not only takes inspiration from nature but replicates the mechanisms that the evolution has designed in order to provide solutions and improvements to current problems. 3

BSD a family of permissive free software licenses, imposing minimal restrictions on the redistribution of covered software. 103, 199

collaborative robotics a branch of industrial robotics where compliant robots are used to work shoulder to shoulder with humans. This robots are more failure tolerant and robust to environment changes and have the ability of dealing with a determined amount of uncertainty. 1

COLLADA defines an XML Namespace and database schema to make it easy to transport 3D assets between applications without loss of information, enabling diverse 3D authoring and processing tools to be combined into a content production pipeline. 103, 199

DARPA Robotics Challenge a competition of robot systems and software teams vying to develop robots capable of assisting humans in responding to natu-

ral and man-made disasters. More information can be found at `http://www.theroboticschallenge.org/`. 4, 199

FAI type I fast adaptation mechanoreceptors, a.k.a. Meissner corpuscles. Respond to stimulation with a burst of firing at the beginning and end of stimulation. Their receptive field is small and are located in the Dermis (just below the epidermis). Better respond to rubbing against the skin or skin movement across a surface. 14

FAII type II fast adaptation mechanoreceptors, a.k.a. Pacinian corpuscles. Respond to stimulation with a burst of firing at the beginning and end of stimulation. Their receptive field is large and are located in the Dermis (deep in subcutaneous fat). Better respond to non uniform stimulation like vibrations. 14

GPL the GNU General Public License is a free, copyleft license for software and other kinds of works. 103, 199

grip force force applied perpendicular to the fingertip surfaces. 18, 20

load force force applied tangential to the fingertip surfaces in order to lift a grasped object. 20

manipulation primitive a reactive controller, designed to perform a specific primitive action on a particular embodiment. 24

micro-neurography a neurophysiological method employed by scientists to visualize and record the normal traffic of nerve impulses that are conducted in peripheral nerves of waking human subjects. 12, 14

Open source refers to a computer program in which the source code is available to the general public for use and/or modification from its original design. 103

ROS a middleware that provides a message passing framework among other features. Its developers community provide a set of open source software libraries and tools oriented for robot applications. 105, 200

safety margin when grasping an object, the difference between the grip force and the slip force. The slip force is the minimum force applied before the object starts slipping. 18

simple tactile reaction time time of reaction to a tactile stimulus in the absence of any cognitive demand of the subject. 17

V1 a region of the functional model of the brain in charge of edge detection and global organisation of the scene. As information is further relayed to subsequent visual areas, it is coded as increasingly non-local frequency/phase signals. 159, 200

V2 a region of the functional model of the brain sensitive to orientation, spatial frequency, and color. 160, 200

V3 a ventral stream region of the functional model of the brain in charge of color extraction, shading and 2D orientation features. 160, 200

V4 a ventral stream region of the functional model of the brain, it is devoted to use the information extracted by V3 and produce viewpoint invariant data of the objects. 160, 200

YAML is a human friendly data serialization standard for all programming languages. 128

YARP a robot middleware that supports building a robot control system as a collection of programs communicating in a peer-to-peer way, with an extensible family of connection types (tcp, udp, multicast, local, MPI, mjpg-over-http, XML/RPC, tcpros, ...) that can be swapped in and out. It also supports flexible interfacing with hardware devices. 123

Bibliography

[Aarno et al., 2008] Aarno, D., Sommerfeld, J., Kragic, D., Pugeault, N., Kalkan, S., Wörgötter, F., Kraft, D., and Krüger, N. (2008). Early reactive grasping with second order 3d feature relations. In Lee, S., Suh, I., and Kim, M., editors, *Recent Progress in Robotics: Viable Robotic Service to Human*, volume 370 of *Lecture Notes in Control and Information Sciences*, pages 91–105. Springer Berlin Heidelberg.

[AgX Dynamics, 2015] AgX Dynamics (2015). AgX Dynamics. `http://www.algoryx.se/products/agx-dynamics/`.

[Aldoma et al., 2012] Aldoma, A., Tombari, F., Rusu, R. B., and Vincze, M. (2012). Our-cvfh–oriented, unique and repeatable clustered viewpoint feature histogram for object recognition and 6dof pose estimation. In *Pattern Recognition*, pages 113–122. Springer Berlin Heidelberg.

[Allen et al., 1997] Allen, P., Miller, A. T., Oh, P., and Leibowitz, B. (1997). Using tactile and visual sensing with a robotic hand. In *IEEE International Conference on Robotics and Automation*, pages 677–681, Albuquerque, New Mexico.

[Allen and Roberts, 1989] Allen, P. and Roberts, K. (1989). Haptic object recognition using a multi-fingered dextrous hand. *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*, pages 342–347 vol.1.

[Antonelli et al., 2011] Antonelli, M., Chinellato, E., and del Pobil, A. P. (2011). Implicit mapping of the peripersonal space of a humanoid robot. In *Computational Intelligence, Cognitive Algorithms, Mind, and Brain (CCMB), 2011 IEEE Symposium on*.

[Antonelli et al., 2013] Antonelli, M., Duran, A., Chinellato, E., and del Pobil, A. P. (2013). *Speeding-Up the Learning of Saccade Control*, volume 8064 of *Lecture Notes in Computer Science*, pages 12–23. Springer Berlin Heidelberg.

[Antonelli et al., 2015] Antonelli, M., Duran, A., Chinellato, E., and del Pobil, A. P. (2015). Adaptive saccade controller inspired by the primates cerebellum. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[Asfour et al., 2006] Asfour, T., Regenstein, K., Azad, P., Schroder, J., Bierbaum, A.,

Vahrenkamp, N., and Dillmann, R. (2006). Armar-iii: An integrated humanoid platform for sensory-motor control. In *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*, pages 169–175.

[Asfour et al., 2013] Asfour, T., Schill, J., Peters, H., Klas, C., Bucker, J., Sander, C., Schulz, S., Kargov, A., Werner, T., and Bartenbach, V. (2013). ARMAR-4: A 63 DOF torque controlled humanoid robot. In *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 390–396. IEEE.

[Asfour et al., 2008] Asfour, T., Welke, K., Azad, P., Ude, A., and Dillmann, R. (2008). The Karlsruhe Humanoid Head. In *Humanoids 2008 - 8th IEEE-RAS International Conference on Humanoid Robots*, pages 447–453. IEEE.

[Azad et al., 2006] Azad, P., Asfour, T., and Dillmann, R. (2006). Combining appearance-based and model-based methods for real-time object recognition and 6D-localization. In *International Conference on Intelligent Robots and Systems*, Beijing, China.

[Balasubramanian and Santos, 2014] Balasubramanian, R. and Santos, V. J. (2014). *The Human Hand as an Inspiration for Robot Hand Development*. Springer Tracts in Advanced Robotics 95, Springer International Publishing, Switzerland.

[Bar et al., 2001] Bar, M., Tootell, R. B., Schacter, D. L., Greve, D. N., Fischl, B., Mendola, J. D., Rosen, B. R., and Dale, A. M. (2001). Cortical mechanisms specific to explicit visual object recognition. *Neuron*, 29(2):529–35.

[Bay et al., 2008] Bay, H., Ess, A., Tuytelaars, T., and Van Gool, L. (2008). Speeded-up robust features (SURF). *Computer Vision Image Understanding*, 110:346–359.

[Bekiroglu et al., 2011] Bekiroglu, Y., Laaksonen, J., Jorgensen, J., Kyrki, V., and Kragic, D. (2011). Assessing grasp stability based on learning and haptic data. *IEEE Transactions on Robotics*, 27(3):616–629.

[Belter et al., 2014] Belter, D., Kopicki, M., Zurek, S., and Wyatt, J. (2014). Kinematically optimised predictions of object motion. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4422–4427. IEEE.

[Bernabe et al., 2013] Bernabe, J., Felip., J., del Pobil, A. P., and Morales, A. (2013). Contact localization through robot and object motion from point clouds. In *13th IEEE-RAS International Conference on Humanoid Robots, HUMANOIDS*, Atlanta, GA, USA. IEEE.

[Billard et al., 2008] Billard, A., Calinon, S., Dillmann, R., and Schaal, S. (2008). Robot programming by demonstration. In Siciliano, B. and Khatib, O., editors, *Springer Handbook of Robotics*, pages 1371–1394. Springer Berlin Heidelberg.

[Blanz et al., 1999] Blanz, V., Tarr, M. J., and Bülthoff, H. H. (1999). What object

attributes determine canonical views? *Perception*, 28(5):575–99.

[Boeing and Bräunl, 2007] Boeing, A. and Bräunl, T. (2007). Evaluation of real-time physics simulation systems. In *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia - GRAPHITE '07*, page 281, New York, New York, USA. ACM Press.

[Bohg et al., 2011] Bohg, J., Johnson-Roberson, M., Leon, B., Felip, J., Gratal, X., Bergstrom, N., Kragic, D., and Morales, A. (2011). Mind the gap - robotic grasping under incomplete observation. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 686–693.

[Bradski and Kaehler, 2008] Bradski, G. and Kaehler, A. (2008). *Learning OpenCV: Computer vision with the OpenCV library.* " O'Reilly Media, Inc.".

[Bülthoff et al., 1991] Bülthoff, H. H., Edelman, S. Y., and Tarr, M. J. (1991). How are three-dimensional objects represented in the brain? *Cerebral cortex (New York, N.Y. : 1991)*, 5(3):247–60.

[Carpin et al., 2007] Carpin, S., Lewis, M., Wang, J., Balakirsky, S., and Scrapper, C. (2007). Usarsim: a robot simulator for research and education. In *Robotics and Automation, IEEE International Conference on*, pages 1400 –1405.

[Castiello, 2005] Castiello, U. (2005). The neuroscience of grasping. *Nature reviews. Neuroscience*, 6(9):726–36.

[Catalano et al., 2012] Catalano, M. G., Grioli, G., Serio, A., Farnioli, E., Piazza, C., and Bicchi, A. (2012). Adaptive synergies for a humanoid robot hand. In *Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on*, pages 7–14.

[Chiaverini et al., 2008] Chiaverini, S., Oriolo, G., and Walker, I. (2008). Kinematically redundant manipulators. In Siciliano, B. and Khatib, O., editors, *Springer Handbook of Robotics*, pages 245–268. Springer Berlin Heidelberg.

[Chinellato, 2008] Chinellato, E. (2008). *Visual neuroscience of robotic grasping.* PhD thesis, Universitat Jaume I, Departament d'Enginyeria i Ciència dels Computadors.

[Chinellato et al., 2012] Chinellato, E., Antonelli, M., and del Pobil, A. P. (2012). *A Pilot Study on Saccadic Adaptation Experiments with Robots*, volume 7375 of *Lecture Notes in Computer Science*, pages 83–94. Springer Berlin Heidelberg.

[Chinellato and Del Pobil, 2008] Chinellato, E. and Del Pobil, A. P. (2008). Neural coding in the dorsal visual stream. In *From Animals to Animats 10*, pages 230–239. Springer Berlin Heidelberg.

[Chinellato and Del Pobil, 2009] Chinellato, E. and Del Pobil, A. P. (2009). Distance and orientation estimation of graspable objects in natural and artificial systems.

Neurocomputing, 72(4):879–886.

[Chinellato and del Pobil, 2016] Chinellato, E. and del Pobil, A. P. (2016). *The Visual Neuroscience of Robotic Grasping*. Springer International Publishing, Switzerland, cognitive edition.

[Chinellato et al., 2011] Chinellato, E., Felip, J., Grzyb, B. J., Morales, A., and del Pobil, A. P. (2011). Hierarchical object recognition inspired by primate brain mechanisms. In *Computational Intelligence for Visual Intelligence (CIVI), 2011 IEEE Workshop on*, pages 1–8.

[Chinellato et al., 2008] Chinellato, E., Grzyb, B. J., and Del Pobil, A. P. (2008). Brain mechanisms for robotic object pose estimation. In *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 3268–3275. IEEE.

[Ciocarlie and Allen, 2009] Ciocarlie, M. T. and Allen, P. K. (2009). Hand Posture Subspaces for Dexterous Robotic Grasping. *The International Journal of Robotics Research*, 28(7):851–867.

[CM Labs, 2015] CM Labs (2015). Vortex Dynamics. `http://www.cm-labs.com/robotics`.

[Coelho Jr. and Grupen, 1997] Coelho Jr., J. and Grupen, R. (1997). A Control Basis for Learning Multifingered Grasps. *Journal of Robotic Systems*, 14(7):545–557.

[Cole and Abbs, 1988] Cole, K. J. and Abbs, J. H. (1988). Grip force adjustments evoked by load force perturbations of a grasped object. *J Neurophysiol*, 60(4):1513–1522.

[Corke, 2011] Corke, P. (2011). *Robotics, Vision and Control*, volume 73 of *Springer Tracts in Advanced Robotics*. Springer Berlin Heidelberg.

[Coumans, nd] Coumans, E. (n.d.). Bullet, Game Physics Simulation. `http://www.bulletphysics.org`.

[Cutkosky, 1989] Cutkosky, M. (1989). On grasp choice, grasp models, and the design of hands for manufacturing tasks. *IEEE Transactions on Robotics and Automation*, 5(3):269–279.

[del Pobil and Serna, 1995] del Pobil, A. and Serna, M. (1995). Spatial representation and motion planning. In *Lecture Notes in Computer Science 1014, Springer-Verlag, Berlin.*

[del Pobil et al., 2013] del Pobil, A. P., Duran, A. J., Antonelli, M., Felip, J., Morales, A., Prats, M., and Chinellato, E. (2013). Integration of visuomotor learning, cognitive grasping and sensor-based physical interaction in the uji humanoid torso. *Designing*

Intelligent Robots: Reintegrating AI, pages pp. 6–11.

[Diankov, 2010] Diankov, R. (2010). *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute.

[Drumwright et al., 2010] Drumwright, E., Hsu, J., Koenig, N., and Shell, D. (2010). Extending open dynamics engine for robotics simulation. In Ando, N., Balakirsky, S., Hemker, T., Reggiani, M., and von Stryk, O., editors, *Simulation, Modeling, and Programming for Autonomous Robots*, volume 6472 of *Lecture Notes in Computer Science*, pages 38–50. Springer Berlin / Heidelberg.

[E. Marchand, 2005] E. Marchand, F. Spindler, F. C. (2005). Visp for visual servoing: a generic software platform with a wide class of robot control skills. *IEEE Robotics and Automation Magazine, Special Issue on "Software Packages for Vision-Based Control of Motion"*, pages 12(4):40–52.

[E. Rohmer, 2013] E. Rohmer, S. P. N. Singh, M. F. ("2013"). V-rep: a versatile and scalable robot simulation framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*.

[Echeverria et al., 2012] Echeverria, G., Lemaignan, S., Degroote, A., Lacroix, S., Karg, M., Koch, P., Lesire, C., and Stinckwich, S. (2012). Simulating complex robotic scenarios with morse. In *SIMPAR*, pages 197–208.

[Edin et al., 1992] Edin, B. B., Westling, G., and Johansson, R. S. (1992). Independent control of human finger-tip forces at individual digits during precision lifting. *The Journal of physiology*, 450:547–64.

[Elbrechter et al., 2012] Elbrechter, C., Haschke, R., and Ritter, H. (2012). Folding paper with anthropomorphic robot hands using real-time physics-based modeling. In *Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on*, pages 210–215.

[Ellenberg et al., 2010] Ellenberg, R., Sherbert, R., Oh, P., Alspach, A., Gross, R., and Oh, J. (2010). A common interface for humanoid simulation and hardware. In *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on*, pages 587 –592.

[Erez et al., 2015] Erez, T., Tassa, Y., and Todorov, E. (2015). Simulation tools for model-based robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4397–4404. IEEE.

[Feix et al., 2015] Feix, T., Romero, J., Schmiedmayer, H.-B., Dollar, A. M., and Kragic, D. (2015). The GRASP Taxonomy of Human Grasp Types. *IEEE Transactions on Human-Machine Systems*, PP(99):1–12.

[Felip et al., 2012] Felip, J., Bernabe, J., and Morales, A. (2012). Contact-based blind grasping of unknown objects. In *12th IEEE-RAS International Conference on Humanoid Robots, HUMANOIDS*, pages 396–401.

[Felip et al., 2015] Felip, J., Durán, A. J., Antonelli, M., Morales, A., and del Pobil, A. P. (2015). Tombatossals: A humanoid torso for autonomous sensor-based task execution research. In *15th IEEE-RAS International Conference on Humanoid Robots, HUMANOIDS*, Seoul, South Korea. IEEE.

[Felip et al., 2013] Felip, J., Laaksonen, J., Morales, A., and Kyrki, V. (2013). Manipulation primitives: A paradigm for abstraction and execution of grasping and manipulation tasks. *Robotics and Autonomous Systems*, 61(3):283 – 296.

[Felip and Morales, 2009] Felip, J. and Morales, A. (2009). Robust sensor-based grasp primitive for a three-finger robot hand. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 1811–1816.

[Felip and Morales, 2014] Felip, J. and Morales, A. (2014). Dual arm sensor-based controller for the cap unscrewing task. In *14th IEEE-RAS International Conference on Humanoid Robots, HUMANOIDS*, Madrid, Spain. IEEE.

[Felip et al., 2014] Felip, J., Morales, A., and Asfour, T. (2014). Multi-sensor and prediction fusion for contact detection and localization. In *14th IEEE-RAS International Conference on Humanoid Robots, HUMANOIDS*, Madrid, Spain. IEEE.

[Fermin et al., 2000] Fermin, I., Okuno, H., Ishiguro, H., and Kitano, H. (2000). A framework for integrating sensory information in a humanoid robot. In *Intelligent Robots and Systems, 2000. (IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, volume 3, pages 1748 –1753 vol.3.

[Flanagan et al., 2006] Flanagan, J. R., Bowman, M. C., and Johansson, R. S. (2006). Control strategies in object manipulation tasks. *Current Opinion in Neurobiology*, 16(6):650 – 659. Motor systems / Neurobiology of behaviour.

[Freeman, 1961] Freeman, H. (1961). On the Encoding of Arbitrary Geometric Configurations. *IEEE Transactions on Electronic Computers*, EC-10(2):260–268.

[Gerkey et al., 2003] Gerkey, B., Vaughan, R., and Howard, A. (2003). Player/stage project: Tools for multi-robot and distributed sensor systems. In *11th International Conference on Advanced Robotics (ICAR 2003)*, pages 317–323, Coimbra, Portugal.

[Gilbert et al., 1988] Gilbert, E., Johnson, D., and Keerthi, S. (1988). A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal of Robotics and Automation*, 4(2):193 –203.

[Gilster et al., 2012] Gilster, R., Hesse, C., and Deubel, H. (2012). Contact points during multidigit grasping of geometric objects. *Experimental brain research*, 217(1):137–

51.

[Goodale, 2004] Goodale, M. (2004). An evolving view of duplex vision: separate but interacting cortical pathways for perception and action. *Current Opinion in Neurobiology*, 14(2):203–211.

[Goodale and Milner, 1992] Goodale, M. A. and Milner, A. D. (1992). Separate visual pathways for perception and action. *Trends in neurosciences*, 15(1):20–5.

[Goodwin et al., 1998] Goodwin, A. W., Jenmalm, P., and Johansson, R. S. (1998). Control of Grip Force When Tilting Objects: Effect of Curvature of Grasped Surfaces and Applied Tangential Torque. *J. Neurosci.*, 18(24):10724–10734.

[Grill-Spector and Kanwisher, 2005] Grill-Spector, K. and Kanwisher, N. (2005). Visual recognition: as soon as you know it is there, you know what it is. *Psychological science*, 16(2):152–60.

[Grill-Spector et al., 1999] Grill-Spector, K., Kushnir, T., Edelman, S., Avidan, G., Itzchak, Y., and Malach, R. (1999). Differential processing of objects under various viewing conditions in the human lateral occipital complex. *Neuron*, 24(1):187–203.

[Grill-Spector et al., 1998] Grill-Spector, K., Kushnir, T., Hendler, T., Edelman, S., Itzchak, Y., and Malach, R. (1998). A sequence of object-processing stages revealed by fMRI in the human occipital lobe. *Human brain mapping*, 6(4):316–28.

[Grzyb et al., 2008] Grzyb, B. J., Chinellato, E., Morales, A., , and del Pobil, A. P. (2008). Robust grasping of 3D objects with stereo vision and tactile feedback. In *International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines (CLAWAR)*, pages 851 – 858, Coimbra, Portugal.

[Grzyb et al., 2009] Grzyb, B. J., Chinellato, E., Morales, A., and del Pobil, A. P. (2009). A 3D grasping system based on multimodal visual and tactile processing. *Industrial Robot: An International Journal*, 36(4):365–369.

[Hackett and Shah, 1990] Hackett, J. K. and Shah, M. (1990). Multi-sensor fusion: a perspective. In *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pages 1324–1330 vol.2.

[Häger-Ross and Johansson, 1996] Häger-Ross, C. and Johansson, R. (1996). Nondigital afferent input in reactive control of fingertip forces during precision grip. *Experimental Brain Research*, 110(1).

[Hamlin et al., 1992] Hamlin, G., Kelley, R., and Tornero, J. (1992). Efficient distance calculation using the spherically-extended polytope (s-tope) model. In *IEEE International Conference on Robotics and Automation*, pages 2502 –2507 vol.3.

[Harris and Stephens, 1988] Harris, C. and Stephens, M. (1988). A combined corner

and edge detector. In *Proc. Fourth Alvey Vision Conference*, pages 147–151.

[Hasegawa et al., 2003] Hasegawa, Y., Higashiura, M., and Fukuda, T. (2003). Object manipulation coordinating multiple primitive motions. In *Computational Intelligence in Robotics and Automation, 2003. Proceedings. 2003 IEEE International Symposium on*, volume 2, pages 741 – 746 vol.2.

[Hebert et al., 2011] Hebert, P., Hudson, N., Ma, J., and Burdick, J. (2011). Fusion of stereo vision, force-torque, and joint sensors for estimation of in-hand object location. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5935–5941.

[Hegdé and Van Essen, 2000] Hegdé, J. and Van Essen, D. C. (2000). Selectivity for complex shapes in primate visual area V2. *The Journal of neuroscience : the official journal of the Society for Neuroscience*, 20(5):RC61.

[Hesse and Deubel, 2010] Hesse, C. and Deubel, H. (2010). Advance planning in sequential pick–and–place tasks. *Journal of Neurophysiology*, 104(1):508–516.

[Hsiao et al., 2010] Hsiao, K., Chitta, S., Ciocarlie, M., and Jones, E. (2010). Contact-reactive grasping of objects with partial shape information. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 1228–1235.

[Hubel and Wiesel, 1977] Hubel, D. H. and Wiesel, T. N. (1977). Ferrier lecture: Functional architecture of macaque monkey visual cortex. *Proceedings of the Royal Society of London B: Biological Sciences*, 198(1130):1–59.

[Huber and Grupen, 2002] Huber, M. and Grupen, R. (2002). Robust finger gaits from closed-loop controllers. *IEEE/RSJ International Conference on Robotics and Intelligent Systems*, 2:1578–1584 vol.2.

[Ishikawa et al., 1996] Ishikawa, T., Sakane, S., Sato, T., and Tsukune, H. (1996). Estimation of contact position between a grasped object and the environment based on sensor fusion of vision and force. In *Multisensor Fusion and Integration for Intelligent Systems, 1996. IEEE/SICE/RSJ International Conference on*, pages 116–123.

[Jackson, 2007] Jackson, J. (2007). Microsoft robotics studio: A technical introduction. *Robotics Automation Magazine, IEEE*, 14(4):82 –87.

[James et al., 2001] James, K. H., Humphrey, G. K., and Goodale, M. A. (2001). Manipulating and recognizing virtual objects: where the action is. *Canadian journal of experimental psychology = Revue canadienne de psychologie expérimentale*, 55(2):111–20.

[Janssen et al., 2000] Janssen, P., Vogels, R., and Orban, G. A. (2000). Selectivity for 3D shape that reveals distinct areas within macaque inferior temporal cortex. *Science (New York, N.Y.)*, 288(5473):2054–6.

[Jenmalm and Johansson, 1997] Jenmalm, P. and Johansson, R. S. (1997). Visual and Somatosensory Information about Object Shape Control Manipulative Fingertip Forces. *J. Neurosci.*, 17(11):4486–4499.

[Jerez and Suero, nd] Jerez, J. and Suero, A. (n.d.). Newton Game Dynamics. `http://newtondynamics.com/`.

[Johansson and Flanagan, 2008] Johansson, R. and Flanagan, J. (2008). Tactile sensory control of object manipulation in humans. *ces.iisc.ernet.in*, 6:67–86.

[Johansson and Westling, 1991] Johansson, R. and Westling, G. (1991). Afferent Signals During Manipulative Tasks in Humans. *Information Processing in the Somatosensory Sysytem. Macmillan Press.*, pages pp 25–48.

[Johansson and Flanagan, 2010] Johansson, R. S. and Flanagan, J. R. (2010). Sensorimotor Control of Manipulation. In *Encyclopedia of Neuroscience*, pages 583–594. Elsevier Ltd.

[Johansson et al., 1992] Johansson, R. S., Häger, C., and Riso, R. (1992). Somatosensory control of precision grip during unpredictable pulling loads. *Experimental Brain Research*, 89(1):192–203.

[Johansson and Vallbo, 1983] Johansson, R. S. and Vallbo, A. k. B. (1983). Tactile sensory coding in the glabrous skin of the human hand. *Trends in Neurosciences*, 6:27–32.

[Johansson and Westling, 1984] Johansson, R. S. and Westling, G. (1984). Roles of glabrous skin receptors and sensorimotor memory in automatic control of precision grip when lifting rougher or more slippery objects. *Experimental brain research. Experimentelle Hirnforschung. Experimentation cerebrale*, 56(3):550–564.

[Johansson and Westling, 1987] Johansson, R. S. and Westling, G. (1987). Signals in tactile afferents from the fingers eliciting adaptive motor responses during precision grip. *Experimental brain research. Experimentelle Hirnforschung. Experimentation cerebrale*, 66(1):141–154.

[Johansson et al., 2001] Johansson, R. S., Westling, G., Backstrom, A., and Flanagan, J. R. (2001). Eye-Hand Coordination in Object Manipulation. *J. Neurosci.*, 21(17):6917–6932.

[Kanehiro et al., 2002] Kanehiro, F., Fujiwara, K., Kajita, S., Yokoi, K., Kaneko, K., and Hirukawa, H. (2002). Open architecture humanoid robotics platform. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 24–30.

[Kaneko et al., 2011] Kaneko, K., Kanehiro, F., Morisawa, M., Akachi, K., Miyamori, G., Hayashi, A., and Kanehira, N. (2011). Humanoid robot hrp-4 - humanoid robotics

platform with lightweight and slim body. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 4400–4407.

[Karayiannidis et al., 2014] Karayiannidis, Y., Smith, C., Vina, F., and Kragic, D. (2014). Online contact point estimation for uncalibrated tool use. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*.

[Kazemi et al., 2012] Kazemi, M., Valois, J.-S., Bagnell, J. A. D., and Pollard, N. (2012). Robust object grasping using force compliant motion primitives. Technical Report CMU-RI-TR-12-04, Robotics Institute, Pittsburgh, PA.

[Kerpa et al., 2003] Kerpa, O., Weiss, K., and Worn, H. (2003). Development of a flexible tactile sensor system for a humanoid robot. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, volume 1, pages 1–6. IEEE.

[Khatib, 1985] Khatib, O. (1985). Real-time obstacle avoidance for manipulators and mobile robots. In *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, volume 2, pages 500 – 505.

[Koenig and Howard, 2004] Koenig, N. and Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IEEE International Conference on Intelligent Robots and Systems.*, volume 3, pages 2149–2154.

[Kopicki et al., 2011] Kopicki, M., Zurek, S., Stolkin, R., Morwald, T., and Wyatt, J. (2011). Learning to predict how rigid objects behave under simple manipulation. In *2011 IEEE International Conference on Robotics and Automation*, pages 5722–5729. IEEE.

[Kourtzi and Kanwisher, 2001] Kourtzi, Z. and Kanwisher, N. (2001). Representation of perceived object shape by the human lateral occipital complex. *Science (New York, N.Y.)*, 293(5534):1506–9.

[Krüger et al., 2011] Krüger, N., Geib, C., Piater, J., Petrick, R., Steedman, M., Wörgötter, F., Ude, A., Asfour, T., Kraft, D., Omrcen, D., Agostini, A., and Dillmann, R. (2011). Object-action complexes: Grounded abstractions of sensori-motor processes. *Robotics and Autonomus Systems*, 59:740 – 757.

[Laaksonen et al., 2010] Laaksonen, J., Felip, J., Morales, A., and Kyrki, V. (2010). Embodiment independent manipulation through action abstraction. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2113–2118.

[Lacheze et al., 2009] Lacheze, L., Guo, Y., Benosman, R., Gas, B., and Couverture, C. (2009). Audio/video fusion for objects recognition. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 652–657.

[Laue and Hebbel, 2009] Laue, T. and Hebbel, M. (2009). Automatic parameter opti-

mization for a dynamic robot simulation. In Iocchi, L., Matsubara, H., Weitzenfeld, A., and Zhou, C., editors, *RoboCup 2008: Robot Soccer World Cup XII*, volume 5399 of *Lecture Notes in Computer Science*, pages 121–132. Springer Berlin / Heidelberg.

[Lele et al., 1954] Lele, P. P., Sinclair, D. C., and Weddell, G. (1954). The reaction time to touch. *The Journal of physiology*, 123(1):187–203.

[Leon et al., 2012] Leon, B., Felip, J., Marti, H., and Morales, A. (2012). Simulation of robot dynamics for grasping and manipulation tasks. In *12th IEEE-RAS International Conference on Humanoid Robots, HUMANOIDS*, pages 291–296.

[León et al., 2010] León, B., Ulbrich, S., Diankov, R., Puche, G., Przybylski, M., Morales, A., Asfour, T., Moisio, S., Bohg, J., Kuffner, J., and Dillmann, R. (2010). OpenGRASP: A toolkit for robot grasping simulation. In *Proceedings of the Second international conference on Simulation, modeling, and programming for autonomous robots*.

[Lowe, 1999] Lowe, D. (1999). Object recognition from local scale-invariant features. In *IEEE International Conference on Computer Vision*, volume 2, pages 1150 –1157 vol.2.

[Meeussen et al., 2006] Meeussen, W., Rutgeerts, J., Gadeyne, K., Bruyninckx, H., and Schutter, J. D. (2006). Particle Filters for Hybrid Event Sensor Fusion with 3D Vision and Force. In *Multisensor Fusion and Integration for Intelligent Systems, 2006 IEEE International Conference on*, pages 518–523.

[Meier et al., 2011] Meier, M., Schopfer, M., Haschke, R., and Ritter, H. (2011). A Probabilistic Approach to Tactile Shape Reconstruction. *Robotics, IEEE Transactions on*, 27(3):630–635.

[Metta et al., 2008] Metta, G., Sandini, G., Vernon, D., Natale, L., and Nori, F. (2008). The icub humanoid robot: An open platform for research in embodied cognition. In *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems*, PerMIS '08, pages 50–56, New York, NY, USA. ACM.

[Metzinger and Gallese, 2003] Metzinger, T. and Gallese, V. (2003). The emergence of a shared action ontology: building blocks for a theory. *Consciousness and cognition*, 12(4):549–71.

[Michel, 2004a] Michel, O. (2004a). Cyberbotics ltd. webots tm : Professional mobile robot simulation. *Int. Journal of Advanced Robotic Systems*, 1:39–42.

[Michel, 2004b] Michel, O. (2004b). Webots: Professional mobile robot simulation. *Journal of Advanced Robotics Systems*, 1(1):39–42.

[Michelman and Allen, 1994] Michelman, P. and Allen, P. (1994). Forming complex dextrous manipulations from task primitives. In *Robotics and Automation, 1994.*

Proceedings., 1994 IEEE International Conference on, pages 3383 –3388 vol.4.

[Miller and Allen, 2004] Miller, A. and Allen, P. (2004). Graspit!: A versatile simulator for robotic grasping. *IEEE Robotics & Automation Magazine*, 11:110–112.

[Moisio et al., 2012] Moisio, S., Leon, B., Korkealaakso, P., and Morales, A. (2012). Simulation of tactile sensors using soft contacts for robot grasping applications. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 5037 –5043.

[Morales et al., 2013] Morales, A., Prats, M., and Felip, J. (2013). Sensors and methods for the evaluation of grasping. In Carbone, G., editor, *Grasping in Robotics*, volume 10 of *Mechanisms and Machine Science*, pages 77–104. Springer London.

[Morales et al., 2006] Morales, A., Sanz, P., del Pobil, A., and Fagg, A. (2006). Vision-based three-finger grasp synthesis constrained by hand geometry. *Robotics and Autonomus Systems*, 54(6):496–512.

[Morrow and Khosla, 1997] Morrow, J. and Khosla, P. (1997). Manipulation task primitives for composing robot skills. In *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 4, pages 3354 –3359 vol.4.

[Mouri et al., 2007] Mouri, T., Kawasaki, H., and Ito, S. (2007). Unknown object grasping strategy imitating human grasping reflex for anthropomorphic robot hand. *Journal of Advanced Mechanical Design, Systems, and Manufacturing*, 1(1):1–11.

[Murphy et al., 1993] Murphy, T., Lyons, D., and Hendriks, A. (1993). Stable grasping with a multi-fingered robot hand: A behavior-based approach. In *IEEE/RSJ International Conference on Robotics and Intelligent Systems*, volume 2, pages 867–874, Yokohama, Japan.

[Nakanishi et al., 2012] Nakanishi, Y., Asano, Y., Kozuki, T., Mizoguchi, H., Motegi, Y., Osada, M., Shirai, T., Urata, J., Okada, K., and Inaba, M. (2012). Design concept of detail musculoskeletal humanoid 'Kenshiro' - Toward a real human body musculoskeletal simulator. In *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, pages 1–6. IEEE.

[Nguyen and Trinkle, 2010] Nguyen, B. and Trinkle, J. (2010). dvc3d: a three dimensional physical simulation tool for rigid bodies with intermittent contact and coulomb friction. In *First Joint International Conference on Multibody System Dynamics*.

[Nowak and Hermsdörfer, 2009] Nowak, D. A. and Hermsdörfer, J. (2009). *Sensorimotor Control of Grasping: Physiology and Pathophysiology*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 8RU, UK.

[Pastor et al., 2011] Pastor, P., Righetti, L., Kalakrishnan, M., and Schaal, S. (2011). Online movement adaptation based on previous sensor experiences. In *IEEE Inter-*

national Conference on Intelligent Robots and Systems (IROS), pages 365–371.

[Petersson et al., 1999] Petersson, L., Egerstedtt, M., and Christensen, H. (1999). A hybrid control architecture for mobile manipulation. In *Proc. IEEE/RSJ IROS'99*, pages 1285–1291.

[Platt et al., 2002] Platt, R., Fagg, A. H., and Gruppen, R. (2002). Nullspace composition of control laws for grasping. In *IEEE International Conference on Robots and Intelligent Systems*, pages 1717–1723, Lausanne, Switzerland.

[Prats et al., 2013] Prats, M., Pobil, A. P. D., and Sanz, P. J. (2013). *Robot Physical Interaction through the combination of Vision, Tactile and Force Feedback - Applications to Assistive Robotics*, volume 84 of *Springer Tracts in Advanced Robotics*. Springer.

[Prats et al., 2010] Prats, M., Sanz, P., and del Pobil, A. (2010). A framework for compliant physical interaction. *Autonomous Robots*, 28:89–111. 10.1007/s10514-009-9145-8.

[Rombokas et al., 2012] Rombokas, E., Brook, P., Smith, J., and Matsuoka, Y. (2012). Biologically inspired grasp planning using only orthogonal approach angles. In *Biomedical Robotics and Biomechatronics (BioRob), 2012 4th IEEE RAS EMBS International Conference on*, pages 1656–1661.

[Rusu and Cousins, 2011] Rusu, R. and Cousins, S. (2011). 3d is here: Point cloud library (pcl). In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1–4.

[Sakagami and Watanabe, 2002] Sakagami, Y. and Watanabe, R. (2002). The intelligent ASIMO: System overview and integration. *Intelligent Robots . . .*, 3(October).

[Sakata et al., 1998] Sakata, H., Taira, M., Kusunoki, M., Murata, A., Tanaka, Y., and Tsutsui, K. (1998). Neural coding of 3D features of objects for hand action in the parietal cortex of the monkey. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, 353(1373):1363–73.

[Santello et al., 1998] Santello, M., Flanders, M., and Soechting, J. F. (1998). Postural hand synergies for tool use. *The Journal of Neuroscience*, 18(23):10105–10115.

[Schaal, 2006] Schaal, S. (2006). Dynamic movement primitives. a framework for motor control in humans and humanoid robotics. *Adaptive Motion of Animals and Machines*, pages 261–280.

[Schaal et al., 2005] Schaal, S., Peters, J., Nakanishi, J., and Ijspeert, A. (2005). Learning movement primitives. In Dario, P. and Chatila, R., editors, *Robotics Research*, volume 15 of *Springer Tracts in Advanced Robotics*, pages 561–572. Springer Berlin / Heidelberg.

[Schettino et al., 2003] Schettino, L. F., Adamovich, S. V., and Poizner, H. (2003). Effects of object shape and visual feedback on hand configuration during grasping. *Experimental brain research*, 151(2):158–66.

[Sereno et al., 1995] Sereno, M. I., Dale, A. M., Reppas, J. B., Kwong, K. K., Belliveau, J. W., Brady, T. J., Rosen, B. R., and Tootell, R. B. (1995). Borders of multiple visual areas in humans revealed by functional magnetic resonance imaging. *Science (New York, N.Y.)*, 268(5212):889–93.

[Shikata et al., 1996] Shikata, E., Tanaka, Y., Nakamura, H., Taira, M., and Sakata, H. (1996). Selectivity of the parietal visual neurones in 3D orientation of surface of stereoscopic stimuli. *Neuroreport*, 7(14):2389–94.

[Shmuelof and Zohary, 2005] Shmuelof, L. and Zohary, E. (2005). Dissociation between ventral and dorsal fMRI activation during object and action recognition. *Neuron*, 47(3):457–70.

[Singhal et al., 2007] Singhal, A., Culham, J. C., Chinellato, E., and Goodale, M. A. (2007). Dual-task interference is greater in delayed grasping than in visually guided grasping. *Journal of Vision*, 7(5):5.

[Smith, 2007] Smith, R. (2007). Open Dynamics Engine - ODE. `http://www.ode.org`.

[Smits, 2015] Smits, R. (2015). KDL: Kinematics and Dynamics Library. `http://www.orocos.org/kdl`.

[Speeter, 1991] Speeter, T. (1991). Primitive based control of the utah/mit dextrous hand. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 866 –877 vol.1.

[Sucan and Kavraki, 2012] Sucan, I. and Kavraki, L. E. (2012). A Sampling-Based Tree Planner for Systems With Complex Dynamics. *IEEE Transactions on Robotics*, 28(1):116–131.

[Sucan and Chitta, 2015] Sucan, I. A. and Chitta, S. (2015). Moveit! `http://moveit.ros.org/`.

[Sugio et al., 1999] Sugio, T., Inui, T., Matsuo, K., Matsuzawa, M., Glover, G. H., and Nakai, T. (1999). The role of the posterior parietal cortex in human object recognition: a functional magnetic resonance imaging study. *Neuroscience letters*, 276(1):45–8.

[Tellez et al., 2008] Tellez, R., Ferro, F., Garcia, S., Gomez, E., Jorge, E., Mora, D., Pinyol, D., Oliver, J., Torres, O., Velazquez, J., and Faconti, D. (2008). Reem-B: An autonomous lightweight human-size humanoid robot. In *Humanoids 2008 - 8th IEEE-RAS International Conference on Humanoid Robots*, pages 462–468. IEEE.

[Tenorth et al., 2012] Tenorth, M., Perzylo, A., Lafrenz, R., and Beetz, M. (2012). The roboearth language: Representing and exchanging knowledge about actions, objects, and environments. In *IEEE International Conference on Robotics and Automation*, Saint Paul, MN, USA.

[Thrun et al., 2005] Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press.

[Todorov et al., 2012] Todorov, E., Erez, T., and Tassa, Y. (2012). MuJoCo: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033.

[Tornero et al., 1991] Tornero, J., Hamlin, J., and Kelley, R. (1991). Spherical-object representation and fast distance computation for robotic applications. In *International Conference on Robotics and Automation*, pages 1602 –1608 vol.2.

[Ullman, 1996] Ullman, S. (1996). *High-level Vision. Object recognition and visual cognition.* The MIT Press.

[Vahrenkamp et al., 2012] Vahrenkamp, N., Kröhnert, M., Ulbrich, S., Asfour, T., Metta, G., Dillmann, R., and Sandini, G. (2012). Simox: A robotics toolbox for simulation, motion and grasp planning. In *International Conference on Intelligent Autonomous Systems (IAS)*.

[van der Heijden et al., 2004] van der Heijden, F., Duin, R. P. W., de Ridder, D., and Tax, D. M. J. (2004). *Classification, Parameter Estimation and State Estimation: An Engineering Approach Using Matlab*. Wiley, New York.

[Waldron and Schmiedeler, 2008] Waldron, K. and Schmiedeler, J. (2008). Kinematics. In Siciliano, B. and Khatib, O., editors, *Springer Handbook of Robotics*, pages 9–33. Springer Berlin Heidelberg.

[Weitnauer et al., 2010] Weitnauer, E., Haschke, R., and Ritter, H. (2010). Evaluating a physics engine as an ingredient for physical reasoning. In *Proceedings of the Second international conference on Simulation, modeling, and programming for autonomous robots*, pages 144–155.

[Yang et al., 2015] Yang, Y., Aloimonos, Y., Fermuller, C., and Aksoy, E. E. (2015). Learning the Semantics of Manipulation Action. *The 53rd Annual Meeting of the Association for Computational Linguistics (ACL) 1 (2015) 676-686*.

[Zhang et al., 2010] Zhang, L., Betz, J., and Trinkle, J. (2010). Comparison of simulated and experimental grasping actions in the plane. In *First Joint International Conference on Multibody System Dynamics*.

[Zhang, 1994] Zhang, Z. (1994). Iterative point matching for registration of free-form curves and surface. *International Journal of Computer Vision*, 13(2):119–152.