Institut für Informatik
Technische Universität München

Compact Models of Objects for Skilled Manipulation

Dissertation

Federico Ruiz Ugalde

Lehrstuhl für Bildverstehen und wissensbasierte Systeme
Institut für Informatik
Technische Universität München

Compact Models of Objects for Skilled Manipulation

Federico Ruiz Ugalde

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr.-Ing. Darius Burschka
Prüfer der Dissertation:	1. Univ.-Prof. Michael Beetz, Ph.D., Universität Bremen
	2. Univ.-Prof. Gordon Cheng, Ph.D.
	3. Univ.-Prof. Gudrun J. Klinker, Ph.D.

Die Dissertation wurde am 18.12.2014 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 20.04.2015 angenommen.

Abstract

Today robots are becoming more capable of adaptive behavior and skilled manipulation. These are important and necessary capabilities needed in robots that will work together with humans in uncontrolled environments. An important aspect with this kind of environments is the difficulty to predict the manner in which the objects will behave. The possibility to predict object behavior can greatly improve and increase the robots capabilities to adapt to new situations and to be more skillful during manipulation.

Traditionally, roboticists have been concentrated to program robot behavior by programing motor motion. In this thesis, we take a different approach: instead of concentrating on the motion of the robot, we put the attention to the objects that are being manipulated. The idea is that once an instruction to manipulate an object is given to the robot, the object will dictate what the robot must do. *The object directs the robot motion.* To achieve this, the robot must have knowledge of how the object behaves. If the robot knows how the object will respond to certain stimuli, then the robot can use these stimuli to produce the desired object response.

By using an object model, the robot can estimate an internal belief of how the object behaves. Using this model the robot can predict the effects or consequences of its actions on the object. Also, this can be used to create a system that implements the inverse: a controller that given a desired outcome produces robot actions.

Because these models are composed of object properties, and the mathematical descriptions of the object behavior are compact, a symbolic representation may be generated to represent them. Thus, these models have the potential to facilitate the communication between artificial intelligence and robotics.

To demonstrate such a system a proof-of-concept object model was implemented: pushing a box. In particular, we selected the challenging case when the robot only uses one finger to push. Such an example represents a case of a skilled manipulation that requires non-linear feedback control for a non-holonomic, time-variant and non-linear dynamical system. The implementation was tested on a real humanoid robot, TUM-Rosie, and its successful performance even under strong perturbations verified.

Kurzfassung

Roboter werden heutzutage zunehmend fähiger, ihr Verhalten anzupassen und geschickt Gegenstände zu handhaben. Dies sind wichtige und notwendige Fähigkeiten, die für Roboter erforderlich sind, welche gemeinsam mit Menschen in unkontrollierten Umgebungen arbeiten. Ein wichtiger Aspekt dieser Art von Umgebungen ist die Schwierigkeit, vorherzusehen, wie die Gegenstände sich verhalten werden. Die Möglichkeit, das Verhalten von Objekten vorherzusehen, kann die Fähigkeiten der Roboter, sich an neue Situationen anzupassen und geschickter zu manipulieren, in starkem Masse optimieren und erweitern.

Herkömmlicherweise hat sich die Robotik darauf konzentriert, das Verhalten der Roboter zu programmieren, indem sie motorische Bewegungen programmierte. In dieser Dissertation wählen wir einen unterschiedlichen Ansatz: Statt uns auf die Bewegungen des Roboters zu konzentrieren, richten wir unsere Aufmerksamkeit auf die gehandhabten Gegenstände. Die zugrundeliegende Idee ist, dass nachdem der Roboter eine Anweisung erhält, einen Gegenstand zu manipulieren, es der Gegenstand sein wird, welcher dem Roboter diktiert, was er zu tun hat. *Der Gegenstand lenkt die Bewegungen des Roboters.* Um dies zu erreichen, muss der Roboter das Wissen besitzen, wie sich der Gegenstand verhält. Wenn der Roboter weiß, wie das Objekt auf bestimmte Stimuli reagieren wird, kann der Roboter diese Stimuli benutzen, um die gewünschte Reaktion des Objekts zu bewirken.

Durch Verwendung eines Objekt-Modells kann der Roboter eine Wahrscheinlichkeitsvermutung dazu schätzen, wie sich das Objekt verhält. Die Verwendung dieses Modells ermöglicht dem Roboter, die Auswirkungen oder Folgen seiner Aktionen für das Objekt vorherzusehen. Dies kann auch zur Schaffung eines Systems genutzt werden, welche das Umgekehrte umsetzt: Eine Steuerung, die aufgrund eines vorgegebenen gewünschten Ergebnisses seinerseits Aktionen des Roboters bewirkt.

Da diese Modelle aus Objekteigenschaften zusammengesetzt sind und die mathematischen Beschreibungen des Objektverhaltens kompakt sind, kann zu deren Darstellung eine symbolische Notation generiert werden. Somit besitzen diese Modelle das Potenzial, die Kommunikation zwischen künstlicher Intelligenz und der Robotik zu erleichtern.

Um ein solches System vorzuführen, wurde als Proof of Concept ein Objektmodell im-

plementiert, nämlich das Schieben einer Kiste. Insbesondere haben wir als erschwerenden Umstand den Fall gewählt, dass der Roboter nur einen Finger dazu benutzt, um zu schieben. Ein solches Beispiel stellt den Fall einer geschickten Manipulation dar, die eine nicht-lineare Regelung für ein nicht-holonomisches, zeitvariantes und nicht-lineares, dynamisches System erfordert. Die Umsetzung wurde an einem realen, humanoiden Roboter, TUM-Rosie, getestet, und ihre erfolgreiche Leistung sogar unter starkem Störeinfluss verifiziert.

Acknowledgements

I would like to give special thanks to my advisor Prof. Michael Beetz for allowing me to be part of his research group, the Intelligent Autonomous System (IAS). His guidance and the environment in his group immensely changed my life for the better. I would also like to thank Prof. Gordon Cheng and his team from the Institute for Cognitive Systems (ICS) for all the great suggestions and insights during the development of this work.

I would like to thank my parents Clementina and Francisco for all their support along my whole life, which up to this point culminates with the realization of this thesis.

I am also extremely grateful with my colleagues at IAS for all the discussions, ideas, collaborations and hangouts that we all experience together. Special thanks goes to Alexis Maldonado, Ingo Kresse and Jonathan Kleinehellefort for all the many ways they were able to help during my PhD, specially for the many contributions to the VFClik system that we developed and tested together, as well as all the puzzling discussions about the meaning of life and of the PhD.

Many thanks go to the people at the Italian Institute of Technology (IIT), in particular to Giorgio Metta, Francesco Nori and Lorenzo Natale for allowing me to participate in their great iCub summer school for many years and to grant us one iCub robot.

This work received substantial funding from the Cognition for Technical Systems (CoTeSys) Cluster of Excellence (part of the Deutsche Forschungsgemeinschaft (DFG) Excellence Initiative). Also, additional financial support was awarded by the Oficina de Asuntos Internacionales y Cooperación Externa (OAICE) of the Universidad de Costa Rica (UCR).

Contents

List of Figures

List of Tables

List of Algorithms

Glossary of Notation

Notation	Description
black box learning	Learning with a system that doesn’t have a predefined structure (model) for the particular plant. 29
pose	Position and orientation. 7, 8, 35, 36, 66, 68–70, 80, 88, 97–100, 117, 121
white box learning	Learning with a system that has a predefined structure (model) for the particular plant. 30

Abbreviations

AI	Artificial Intelligence. 11, 14
COM	center of mass. 46, 52, 54–57, 60
COP	center of pressure. 41, 44, 46, 47, 51, 52, 70, 78, 79, 81, 87, 99
COR	center of rotation. 18, 45
CoTeSys	Cognition for Technical Systems. VII, 2
DFG	Deutsche Forschungsgemeinschaft. VII
DOF	degrees of freedom. 105, 106, 108, 115
HAL	human activity language. 14
IAS	Intelligent Autonomous System. VII, 103, 105
ICS	Institute for Cognitive Systems. VII
IIT	Italian Institute of Technology. VII
LS	limit surface. 18, 19, 45–47, 51–53, 55

OAC object action complex. 11, 12, 20

OAICE Oficina de Asuntos Internacionales y Cooperación Externa. VII

OM Object Model. 5, 7–10, 20, 21, 25, 26, 29–32, 34, 35, 37, 39, 40, 59, 64, 70, 71, 88, 92, 95–100, 121

OMS Object Model System. 5, 9, 21–25, 37, 40, 48, 59, 88, 95, 96, 98, 100, 103, 109, 110, 115, 121

PPC pushing point of contact. 9, 19, 20, 39, 47, 54, 55, 57, 78, 79, 99

RRT rapidly-exploring random trees. 15, 16

UCR Universidad de Costa Rica. VII

VFClik vector field based close loop inverse kinematics. VII, 109, 119

Chapter 1

Introduction

We want robots to understand instructions, behave naturally, react fast (when necessary) and anticipate outcomes of actions in the presence of objects and other actors (humans and robots). All this is necessary for a robot to be able to work properly in the presence of humans and within its associated environment with common everyday objects all around.

Currently, robot skills and dextrous reactions are still insufficient for proper behavior of an assistant robot in a human everyday environment. We believe, there are two reasons for this: 1) the lack of capabilities given to the robot to work with the objects and subjects present on the scene. Can a robot detect and react to an unexpected object behavior without preprogramming such possible event in one particular scenario?, and 2) a proper way to communicate instructions and knowledge of the necessary actions to be executed to the robot.

Latest advancements in robotics have made possible robots to cook or prepare meals [Beetz et al. 2011] or to fold towels [Maitin-Shepard et al. 2010] autonomously. These robots are able to perceive the objects they need and use them in some particular tasks. Also, there are robots able to perform very skilled tasks like swinging a tennis racket [Ijspeert, Nakanishi, and Schaal 2002], or to move rapidly in difficult situations like falling [Raibert et al. 2008]. However, a lot of attention has been put into the movement of the robot itself [Schaal et al. 2005] [Schaal, Kotosaka, and Sternad 2000], but not so much on the objects to manipulate [1].

If a human requires an object (in case the human expresses such need or if the robot could infer such need), then the robot could perform the necessary actions to bring this object in reach of the human (Fig. 1.1). In this situation, the robot needs to know how to interact with the object to be able to fetch it.

We want a robot that knows how things work (objects, humans, other robots);

[1] A good exception to this is, the grasping problem, extensively studied by [Han et al. 2000] [Han, Trinkle, and Li 2000] [Han and Trinkle 1998] [Buss, Hashimoto, and Moore 1996] [Buss and Schlegl 1997], and more practical approaches [Maldonado, Klank, and Beetz 2010]

know how they will behave and know how to use them.

Such robot would require to possess several capabilities. It needs to:

1. Know what is happening or will happen including:
   a) Actions already in progress.
   b) Expected outcomes of such actions.
   c) Intentions of actors.
   d) What is the robot itself currently doing.
2. Know how to get involved. (e.g. collaborate or give space and wait)
3. Detect unexpected situations and react properly to them.

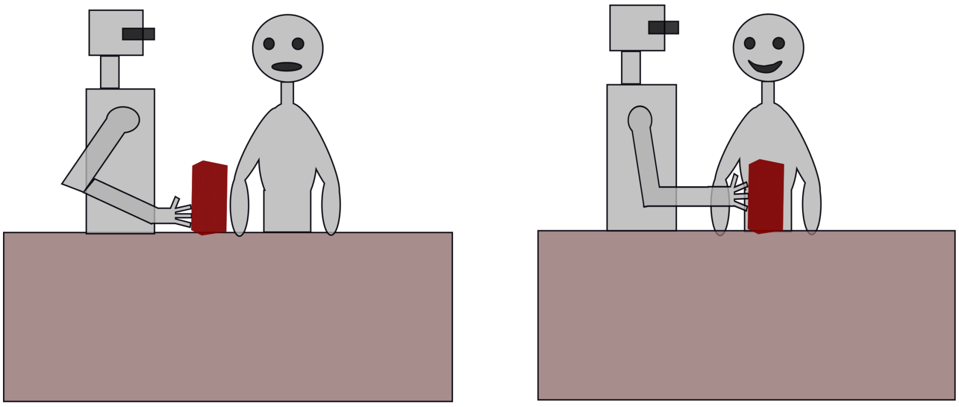


FIGURE 1.1. Robot assisting human.

For the accomplishment of some of these capabilities the robot may need to do reasoning and planning, have a very complete knowledge database, a good perception system and action control loops (Cognition for Technical Systems (CoTeSys) cognitive architecture [Beetz, Buss, and Wollherr 2007] and iCub cognitive architecture [Sandini, Metta, and Vernon 2007]). For example, if the robot is helping a human to cook eggs, the robot may find the situation where there are no eggs and may have to drive to the nearest grocery store to buy them. This alternate solution (of driving to a grocery store), needs the help of a knowledge database to know where to find more eggs, and reason to find out that if there are no eggs the robot needs to find an alternate solution that is acceptable (plan transformation [Beetz 2002], [Beetz 2005], knowledge [Beetz, Mösenlechner, and Tenorth 2010]). Also, the robot may need to, for example, know where or how to strike the egg to break it properly in order to fry it on a pan. The robot needs to learn to manipulate objects to achieve desired effects while avoiding undesired ones. This is more at the level of each manipulation action or object related action; the robot needs to manipulate, modify or use an object for some purpose.

All this defines **two levels of abstraction**: 1) the high level where reasoning and planning happens and 2) the low level or physical interaction level (see figure 1.2). Finding a way to bind these two levels is key in building an assistant robot that is able to perform in the ways we want and need [Domingos 2006].

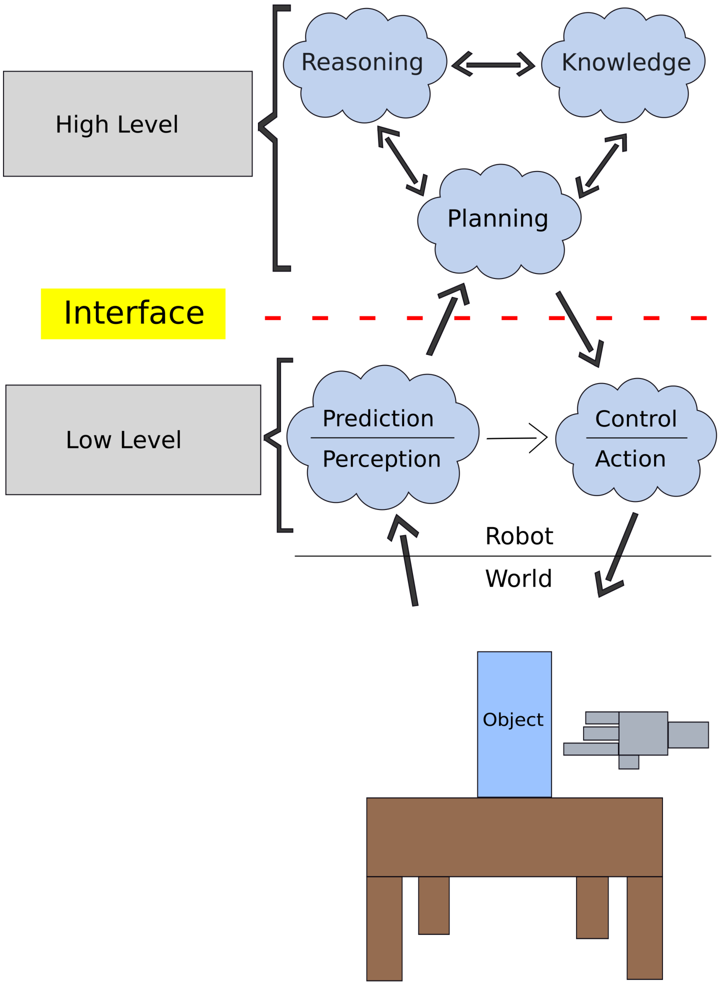


FIGURE 1.2. Levels of abstraction.

Also needed is the ability to **use prediction constantly**. Currently, robots lack or are deficient in this respect, but humans seem to do it [Wing and Flanagan 1998] [Wolpert and Kawato 1998]. This ability gives humans the feeling that everything is under control, things are behaving as expected, but when not, humans feel strongly this lack of normality and go into an alert state and finally change the behavior to try to get the situation under control [Oztop, Kawato, and Arbib 2012]. An example would be to press the brake pedal in a car and the pedal goes straight to the floor (mechanical failure in the brake system), one would feel the lack of "normality", will go into alert state, and will try to react to it immediately. This ability is very important in ever changing environments.

All objects in the scene will be different in some way, but we can always classify them according to some common features. These ***types* of objects**, are for example, cylinders, boxes, liquids, cars, etc. Each of them will have different properties. These **properties must be explored** in some way. Usually a robot that wants to learn how an object behaves and later wants to interact with this object must have the necessary perception capabilities to recognize and characterize the object well enough to be able to use it.

Grounding [Harnad 1990] [Coradeschi, Loutfi, and Wrede 2013] and embodiment [Duffy and Joue 2000] [Ziemke 2001] are two important concepts to take into consideration when designing a robotic system. If the robot can extract the object properties with its own means, then the values of those properties are better adjusted to that robot. As a result, future use of that object by that robot will give better prediction results and therefore better overall manipulation (it is embodied to the robot hardware). Also, if the robot can extract the object properties by itself and can express object behavior and its properties in a way that can be easily translated into or from an action language, such system is facilitating low to high level interfacing and giving to the action language symbols (grounding object knowledge).

In this work we concentrate more on the lower level object knowledge: how objects behave and how to control them. We will also discuss how our low level system could interface with the high level and why we believe it is very suitable to integrate with high level reasoning and planning.

1.1 Proposed solution

Know the environment! Know the objects!

We believe that if the robot knows how things behave around it, it will be able to interact more properly with them. First, the robot should be able to predict object behavior, i.e, how will an object behave given the current situation. Second, using this predictive knowledge use it to control the object to obtain a particular result, and finally if things didn't work as expected, be able to detect this wrong behavior by comparing it with the predicted outcome.

To be able to predict object behavior it is convenient to look at the objects as mechanical entities in a physical world. As an additional benefit, this physical behavior of objects happens to have a higher level symbolic representation, e.g, rising the temperature of an egg will change it chemically, but a human will think of it as “cooking it”. In this work, we will concentrate on examining the physical behavior of objects but will also discuss possible ways this work can be used to obtain a symbolic representation of such behavior.

As a proof of concept, we will examine in detail how a box behaves on top of a table (chapters 4 and 5).

The proposed solution revolves around the idea of mathematical models for objects which we call Object Models (OMs). Object Models (OMs) will bring the capabilities of prediction and control to a robot.

Prediction: What would happen if a force is applied to an object? When the robot exerts forces to an object and the robot is able to know what is going to happen when he applies such forces, it will be able to detect out-of-the-normal situations and improve the control of his actions.

Control: What forces must be applied to an object to achieve a particular final state? The robot needs to know how the object reacts to forces, its properties, and his own capabilities to know how to control the object.

To create an Object Model (OM) we require **object modelling**. How do we characterize object behavior? By analyzing physically and mechanically the object one may find compact mathematical representations of behavior; a set of object models can be derived and given to the robot. With an Object Model (OM) associated with the current object and its properties, the robot can then predict object behavior and control action outcomes.

And to make it work we require to **identify the object and its properties**. What are the current properties of the object? By playing, and poking with the object the robot can calculate and/or infer what its properties are. The identification of these properties is necessary for completing the Object Model (OM). A particular instance of a type of object is here identified. The properties can also be given to the robot from prior knowledge (knowledge database).

In the present work, we are contributing to the connection of two levels of abstraction, mentioned previously (see figure 1.2), by developing a system (object model and motor controllers) that provides an interface that facilitates the communication of information between these two layers.

1.2 Example: Pushing and toppling

The proposed solution is a system architecture that will contain as the main component an Object Model System (OMS). This system will be tested using a proof-of-concept example:

pushing and toppling a box that is on top of a table (figures 1.3 and 1.4).

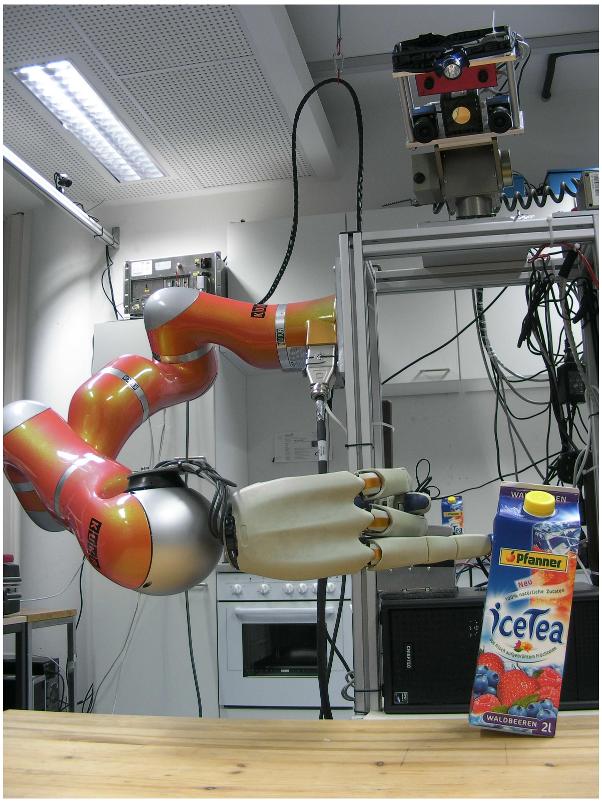


FIGURE 1.3. Robot toppling an ice tea box.

The main task of this example is to bring an object to a particular desired position on top of a table and/or topple an object if another orientation of the object is desired.

Desired effects of such action are to bring objects near in order to grasp them, or to push them away to make space or to give them to somebody else [Chang, Zeglin, and Pollard 2008] [Dogar and Srinivasa 2010b]. Repositioning of the object may be desirable to improve the current kinematic capabilities of the robot [Zacharias, Borst, and Hirzinger 2007]. The action may require less effort in some robot joints than for example lifting the object. This is specially important if the object is too heavy and the robot fingers are not capable of holding the object.

Some undesired effects of such action could be to push another object that was not intended because it was in the way of the intented object pushing trajectory. Also the hand and arm must avoid any colitions with other objects while pushing. To avoid such possible undesired effects the robot will need a comprehensive perception system that is capable to detect all these objects and obstacles, a reasoning system that is able to select which actions to take prior to the main pushing action and command a low level system capable of dealing with

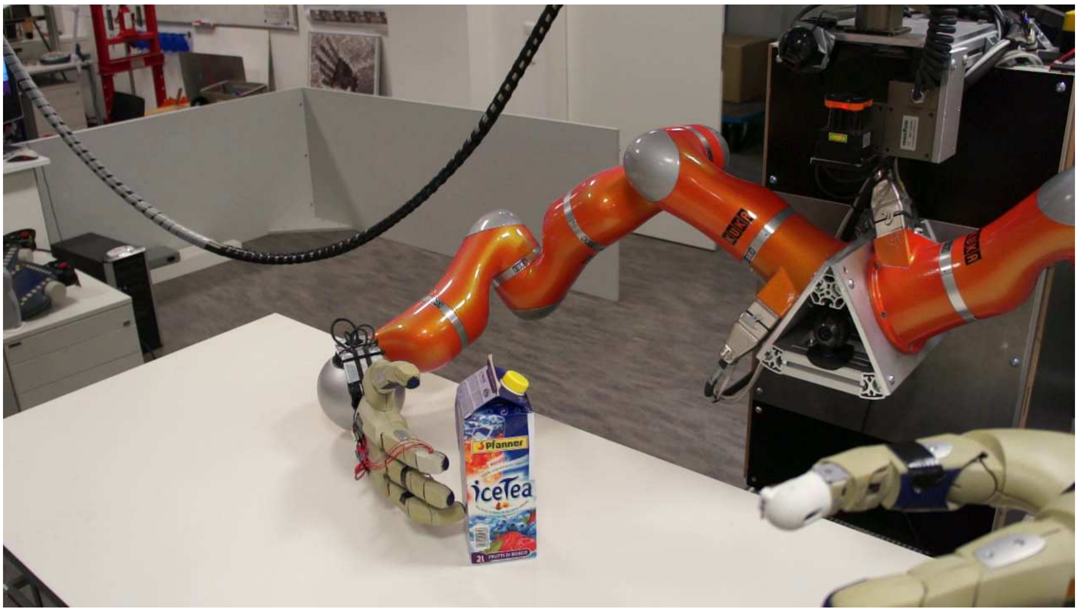


FIGURE 1.4. Robot sliding an ice tea box.

such actions. Such situations require the help of many different systems that are part of a cognitive architecture that will be mentioned in the following chapters.

Pushing with only one contact point (using one finger) is particularly difficult because the system is very unstable in practice. It is a skilled manipulation task that shows not only the importance of a proper perception-action loop but also requires (as we will see in chapter 5) some type of low level planning.

> *Our final goal is to have a robot capable of exploring the object properties in order to learn the Object Model (OM) constants. Then using only one finger, guide the object towards a particular position and orientation on the table by pushing on a side of the object. All this while the Object Model (OM) is evaluating forces to predict the next object position.*

Pushing in one contact point is a hard robotic task that requires a well-designed software and a hardware platform that has force sensing, object position estimation, fast control loop, precise finger positioning and speed control, sensor calibration, interaction with the environment and non-linear, non-holonomic, time variant feedback control. Humans can also accomplish this task, but it is also difficult for them and requires some training, specially when trying to achieve goal position and orientation (pose) at the same time with only one finger, in one single shot [2].

Also a toppling action is considered. The toppling action consists on pushing an object from a side and predicting if it's going to slide or topple, none or both. One would expect that, depending on the geometry of the object, there are more chances that the object will topple if pushed on a higher location, and will slide if pushed on a lower location. Also of importance is, how much force the robot needs to apply to the object for motion to occur at all.

Toppling may be important in situations were toppling an object to catch it with the other hand in another orientation is necessary (reorientation) or convenient. This action may be faster than grasping the object and putting it back on the table in another orientation to later lift it. This may require less effort than to lift the object with one hand and give the object to the other hand in another orientation. Also this predictive action may be necessary in order to slide the object; the robot needs to be sure that the object will slide and not tilt, then it needs to choose a pushing height that will not tilt the object, based on the predictions of the Object Model (OM).

Therefore, the object model will consist of two different mathematical models, one for toppling (*toppling/sliding model*) and one for pushing (*planar sliding model*).

1.3 Contributions

We present a framework capable of predicting action outcomes and of controlling objects. As a proof-of-concept to this system we implemented on a real robot (TUM-Rosie Beetz et al. 2011) an object model for a box on top of a table which can slide or rotate, and that the robot is able to predict and control. Also, the robot is able to acquire most of the object properties by itself, or to get them from a knowledge database or from another external perception system.

The main contributions are:

- An object model system for prediction and control.
- An object model for a box type of object, during sliding and toppling motion.
- A prediction and control system of sliding motion when pushing with a single point of contact. (most challenging situation)
- A system for prediction and control for toppling of a box.

[2] One single shot: The subject touches the object with only one finger and never removes the finger of the object surface during the whole control episode. Also the pushing action must arrive to the final goal without using another finger or without needing to reposition the pushing finger on another object surface.

- A feedback controller that is highly robust for a non-holonomic, non-linear, time-variant system that pushes an object from any pose (position and orientation) to another.

- A robot able to explore objects to obtain their properties (grounding, embodiment).

- A practical implementation and successful execution of a complete system that integrates the object model system, manipulator motion (reaching and pushing) and perception that runs on the TUM-Rosie robot.

1.4 Future Work

The general architecture here presented (Object Model System (OMS)) is expected to be used on objects or parts of objects. The possibility to use this system to predict and control parts of the robot itself have not been considered (for example, for free motion generation). Some systems may be more appropriate for such use.

Although the system for prediction of sliding motion is limited to one pushing point of contact (PPC), expanding it to include more pushing points of contact (PPCs) should be possible (maybe by calculating an equivalent total force). For controlling using more than one pushing points of contact (PPCs), some solutions are already available in the literature, and this situation is more simple than the one presented in our work as long as the pushing force is inside the friction cone [Kumar 2009] (no sliding on the pushing surface is assumed).

Not all behavior is completely modelled, for example, deformation of the pushing surfaces of the object is not modelled, these may or may not be important under some more uncommon situations/objects.

Inertial effects are not taken into account in our solution. This may become important if we want to increase the speed of the system (when the object turns it will experience higher accelerations), or if we want to model more interesting motion behavior (for example, pushing so hard to let the object slide alone and later stop in some far away position, or to throw the object away).

Our model uses some approximations, that during prediction, gives small deviations from the real observed behavior. Although not important, this could be improved.

Future work to address some of these limitations will be discussed in chapter 6 of this thesis.

1.5 Thesis Outline

This thesis is organized to be read in a linear fashion, but the reader may choose to read only parts of it. If the reader has interest on the general idea of Object Models (OMs) then chapter 3 is the appropriate choice. Chapters 4 and 5 are concern with the pushing example. More in detail:

In chapter 2 we will explain previous work and related work to object models and some other related topics to this thesis. In later chapters we may also expand more on related work, if necessary, for improving exposition.

Chapter 3 explains the general idea of Object Models (OMs). We discuss possible ways to build the Object Model (OM) and how is the robot related to the Object Model (OM). A general system architecture is here presented.

Chapter 4 explains the Object Model (OM) of our proof-of-concept example. It first talks about the fundamental theory for pushing and the mechanical considerations that are necessary to develop our specific object model. In particular, friction, toppling and sliding, among other concepts, will be analyzed and explained. Later the box model is derived for the planar sliding and the toppling sliding sub-models. The chapter ends explaining our methods to acquire the object properties, and with experimental results to acquire such properties. This object properties will be used in the results sections of the following chapters.

Prediction and control are explained in chapters 4 and 5. The results are also discussed there. Conclusions and future work are presented in chapter 6.

The hardware and software systems are explained in Appendix A. We go into the details about some of the systems that were used in our robots to run the object model system.

CHAPTER 2

Skillful Manipulation and Related Work

If we intent to create robots that can manipulate objects receiving spoken commands and execute them using very natural, reliable and safe motion, they will need to connect these commands (high level instruction) to a low level motor control in a meaningful manner. This type of robot execution we call *skillful manipulation*; the robot understands the command and produces a skillful handling of the object.

We are interested in a system that can understand a high level instruction by extracting the type of manipulation required and the corresponding action parameters from it. This information could be used to configure a low level motor controller. Since we are also very interested to obtain a smooth natural motion, we would like to find a manipulation strategy to control the objects at the lowest levels.

In this chapter we present work related to methods that try to connect instructions to motion, then the concept of affordances is discussed and an implementation of it (Object Action Complexes (OACs)). Black box and white box learning are compared and the reasons why white box learning may be more appropriate to our goal discussed. The selected proof-of-concept *pushing action* and its usefulness in everyday situations is commented. Previous work on reaching and sliding (which are part of the *pushing action*) is investigated.

2.1 Mapping High Level Instructions to Low Level Motion Control

Unifying Artificial Intelligence (AI) and robotics (in particular high level reasoning and planning with low level motor control) has been challenging but some like [Domingos 2006], [Jenkins and Matarić 2004] have shown possible ways of interfacing this two domains. Using language as an interface has shown some promising results.

In [Iwahashi 2006] and [Sugiura and Iwahashi 2007] the robot learns arm trajectories and object relative positions associated with command verbs from a human instructor, therefore, providing a way to unify AI and robotics. Such a system may prove insufficient when a robot needs to use a feedback control system to achieve the desired goal. When the human needs to use feedback to execute actions properly, learning trajectories will not provide feedback to the system. The problem is that the behavior will depend more on the object than the human, therefore, imitating the human motions for an action that requires feedback control may give poor results.

A concept that can be useful for skillful manipulation are the *affordances* [Gibson 1986] [Norman 2002]. Affordances are action possibilities that are perceivable by an actor. Given an object and an actor, the affordances suggests how this object can be used by this actor. Action possibilities are given by the output of the predictions from a model, given all the set of possible forces the robot could apply to an object. We understand the word *action*, as the outcome (immediate or long term) of a robot actuation. For example, a chair that was moved was an action executed by a robot, but the robot actually just applied some particular forces to some particular points of a chair.

Previous work has been done in providing robots with affordances. In [Fitzpatrick et al. 2003], a robot observes movements executed by himself of pushing/pulling and poking of an object, then it associates these actions with the corresponding outcomes, and finally it is able to execute commands to achieve a desired outcome. Vision is used to feedback the information of the generated movement, a histogram based learning algorithm is used to learn the model. The setup is two dimensional, with objects sliding on a table, and the learning is also done using a 2d world. The experiment maps initial hand positions to final object movements, it doesn't consider forces.

Related to the concept of affordances are the inverse and forward models described by [Wolpert and Kawato 1998 and Oztop, Kawato, and Arbib 2012]. Using these inverse and forward models it could be possible to construct a system that can predict and execute action based on a previously learnt behavior. It could be used to learn the behavior of objects and how to execute actions on them.

Also building on top of the affordances concept, is the notion of Object Action Complexes (OACs) [Wörgötter et al. 2009]. Object Action Complexes (OACs) state, that objects are constantly suggesting actions (to be executed with them or upon them) which will ultimately transform the object to a new state. This can also be interpreted as that the action transforms the usefulness of the object.

One of the first examples of Object Action Complexes (OACs) applied to a robot is shown by [Omrcen et al. 2009]. They demonstrate a robot learning a pushing rule, i.e. how an object reacts if it is pushed, and also learning the inverse pushing rule, i.e. how to push an object in order to accomplish some goal. They build the pushing rule with an artificial neural network.

These two previous systems use black box learning for building the object model (figure 2.1 left). With black box learning a system takes all the sensor information and classification for many examples and tries to build a model. Because of the way this model is constructed or adjusted it can't be easily understood (semantically) by a human. Also, this model usually has no (practical) ways to be adjusted to new conditions [1]. For example, if the robot learnt the model of a box with 4x4x4cm dimensions and afterwards it is presented with a 2x8x4cm box, the model can not be successfully used for this new object, unless it is trained again for the new box. One could give the robot, as part of the training, not only an object with a fixed property value, but also more objects of the same type with different property values. In this situation, the training space will increase significantly, specially if we are including not only one changing property but multiple ones. This is the so-called *curse of dimensionality* [Bellman 1961] [Mitchell 1997]; learning becomes more difficult with more dimensions. Also, out of this situation other problems may arise, for example, what if we don't have all the objects of the same type but with a different property value during the learning process? Will the model predict correctly with the unseen property values? Is it possible to produce all the objects with different property values? All these represent two practical problems: 1) too much training needed and 2) availability of all the objects during learning. Both are not impossible to overcome, in 1) we could just train for longer times and wait, in 2) we could just not train for all objects, but when new objects appear then train again, much similar to what humans do (on-line learning). We will show that our system requires few training examples, and can also improve its models when new objects appear without the need of extended periods of time.

Another important reason for which black box learning is not convenient for our purposes, is because, as mentioned before, we want a model that can be as easy as possible to connect to a high level system for reasoning and planning, or in other words, that our models and system can have a semantic meaning or representation. When we use parameter estimation (e.g. [Walter and Pronzato 1997]) together with a carefully designed model, based on mechanical behavior, we obtain a model that has properties with semantic meaning, which we believe will make it easier to express with an action language.

Our approach is model-based and it adapts the object to the model by identification of its parameters. This white box learning approach (model-based learning), give the following main

[1] On-line learning allows a robot to adjust to new conditions at the cost of gradually loosing the learnt information for the old conditions, and requires a lot of more training examples

advantages: 1) fast learning and 2) semantic description of the parameters. The fast learning means that the system requires fewer “training” examples to adapt the model to the particular object (this is done using parameter identification), and also it allows to generalize to different objects sufficiently without having to increase the quantity of training examples to learn too much.

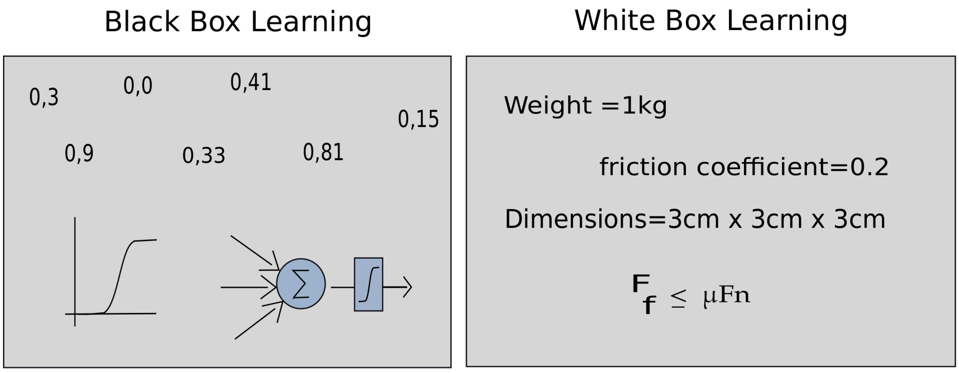


FIGURE 2.1. Black box and white box learning.

We expect that with an action language we can facilitate the interfacing between a low level motor system and Artificial Intelligence. [Guerra-Filho and Aloimonos 2007] present a human activity language (HAL). They explain ways of connecting sentences to particular actions. Also [Inamura et al. 2004] show a system for generating actions based on *proto-symbols*. Something in common with these two solutions is that they make some sort of mapping between the sentence or part of the sentence and the low level motor representation. This is something we intend to do, but instead of mapping the sentences to human acquired movements, we want to map these sentences to object models and their properties.

2.2 Pushing Action for Manipulation

Pushing can be used to accomplish many tasks and to increase the robustness of other tasks such as grasping [Chang et al. 2008] [Dogar and Srinivasa 2010a]. This is accomplished by positioning the objects in desired locations that are convenient to the robot and maybe to other actors (robots or humans). These convenient locations may be selected for optimizing or improving reachability, visibility or stability, or for avoiding to touch other objects. Also, a particular robot may not have enough dexterity [Zacharias, Borst, and Hirzinger 2007] or actuation to accomplish tasks that requires prehensile manipulation, but if the robot uses external forces such as gravity, friction, centrifugal or even Coriolis to get the necessary additional degrees of freedom [Lynch and Mason 1999] it may still be able to succeed. One possible way is to use a non-prehensile manipulation action such as pushing.

2.3 Manipulator Motion

Before pushing an object, first it is necessary to move the robot end-effector near to the object and then approach the object until contact with the desired surface occurs, then additional motion is necessary to push the object in some particular way. We can categorize this movements in two classes: 1) reaching motion and 2) pushing motion. Reaching motion happens when the robot is not touching the desired object and the manipulator must move (or navigate) to the contact point (or initial pushing point). Pushing motion is necessary during contact with the desired object and since there is an interaction with the object, the object model becomes the driving element for the movement. Having a compliant robot [Beetz et al. 2011] [Albu-Schäffer et al. 2007] [Liu et al. 2008] during pushing motion is particularly advantageous, because contact with the object becomes soft giving a smooth motion compared to a non-compliant contact. The arms and fingers of our robot are compliant.

2.3.1 Reaching Motion

For reaching motion there are different motion generators available for our use. For our purposes, it is important to be able to set the position and orientation of the goal, and to also have some way to avoid obstacles on its way; we don't want to touch the object unintentionally. Classical motion planning [Barraquand and Latombe 1991], potential fields [Khatib 1986b] [Khatib 1986a] [Beetz et al. 2010] and dynamic movement primitives [Schaal 2006] are the typical solutions for this task.

Classical motion planning consists on precalculating a trajectory free of collisions to reach the final goal. One notable example of classical motion planning for arm navigation comes from the Rapidly-exploring Random Trees (RRTs) [LaValle and Kuffner Jr 2000]. Previous to any real motion, this algorithm virtually explores possible movements that provide no collisions towards the goal. Each time a collision is detected, the Rapidly-exploring Random Trees (RRTs) back-off and try another *path* towards the goal. They present the advantage of having a previously calculated collision-free trajectory, but are not reactive in case of changes in the obstacles. In such case, the robot can detect the difference and stop and wait for a new collision-free path to be calculated. Some technics exist to make Rapidly-exploring Random Trees (RRTs) more reactive [Yershova et al. 2005].

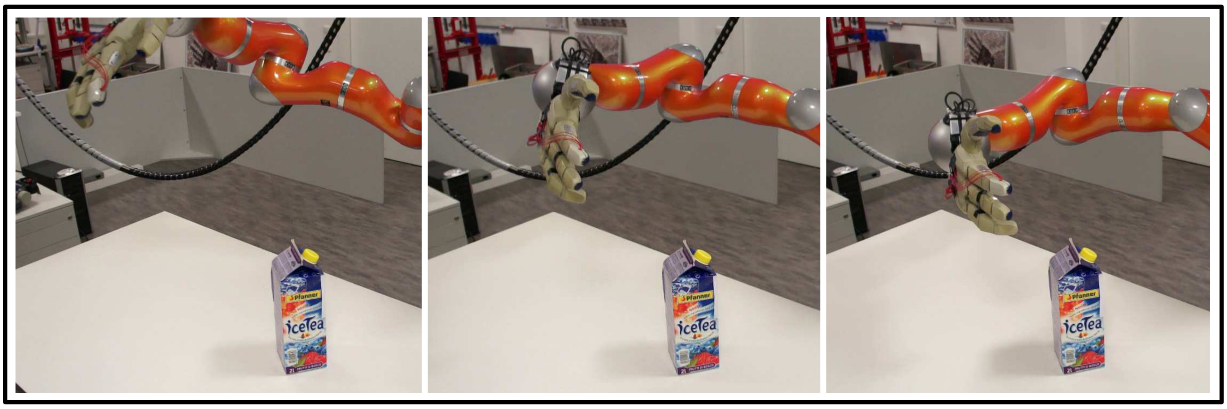


FIGURE 2.2. Reaching Motion

First uses of potential fields for robot motion start back in the 80's with the motivation to have a lower level system (instead of a high level system) able to execute feedback control for robot movement [Khatib 1986b]. Later, it was found that frogs (and possibly many more animals) produce a similar behavior: frogs produce vector fields of forces depending on the particular stimulation activated in the spinal cord [Giszter, Mussa-Ivaldi, and Bizzi 1993]. Potential fields present the advantage of being reactive to changes of the obstacles and goal while the manipulator is in motion. They are fast to calculate, but they suffer from local minima solutions. Informally this means that the manipulator could get "stuck" in situations involving multiple obstacles near to each other. Potential fields can generate all sort of movements, which just depend on the differential equations describing them. The typical way to use potential fields is to assign repellers and attractors for the obstacles and goal respectively.

Dynamic movement primitives are inspired by the idea that complex biological movements may be based on a sequence of simpler movements: primitive movements [Schaal 2006]. After some learning and training a human tends to perform a task in a particular way. Once the motion primitive is learnt the human unconsciously simply *activates* this motion primitive and the motion is executed. The same can be applied to robots. This methods require a series of trajectory examples from humans, a model of the trajectory is built and then can be executed in a robot when a proper representation is chosen. These dynamic movement primitives allow to change the goal while maintaining the *shape* of the motion. An integration of motion primitives with obstacle avoidance using potential fields is explained by [Park et al. 2008].

These systems in the context of a compliant robot arm for reaching motion present different behavior during an unexpected collision. Imagine the robot arm moving towards the goal and then suddenly a human grabs the robot arm and holds the movement. A motion planner that just executes a precalculated trajectory, if it observes for unexpected situations, may respond with a stopping motion, adding a new obstacle right there and recalculating a new collision-free trajectory, this may cause the robot to stop for a while or jitter.

On the other hand, a potential field has a more favorable behavior: if the motion is stopped by an object, it adds a new obstacle and moves away from it to avoid it and once it is far away enough from the obstacle the goal will attract the manipulator in a new quasi-direct route to the goal. If the manipulator is grabbed and moved away to another position and then released the potential field will nicely just go in a new straight line to the goal instantaneously. Or better said, the potential field reacts immediately and with an appropriate response to changes in the environment [2].

Because of this nice adaptability, easily implementable and simplicity we opted to use the potential field solution.

In very much every aspect, the object model system is completely unaware of the reaching motion, therefore, any system can be used here.

2.3.2 Pushing Motion

Advancements in robot technology like the DLR-III lightweight arms [Albu-Schäffer et al. 2007], DLR-HIT hands [Liu et al. 2007] and the iCub v2 [Fumagalli et al. 2010], allow robots to interact in more flexible ways with the environment. Robots like these have a technology called impedance control. With it, these robots can produce movement and touch or collide with objects without producing damage; the force is regulated and controlled through the stiffness selection. Since our system is more centered on the object, the arm and hands motion systems are not the main concern, but still our systems needs a manipulator that can touch and push objects in a careful way, and impedance control gives exactly that.

During pushing the robot applies a force to the object in a contact point, this force may produce an object movement. Two factors will determine the quality of this movement: 1) the type of low level control that is applied to move the arm/finger and 2) the pushing rule that regulates object movement. The first one acts on the finger tip and the second one on the object. During contact, the object and robot systems become a complete new kinematic/dynamic chain, and the object behavior will have an effect on the motions of the robot if the robot is compliant. The systems described on chapters 4 and 5, model the object considering the inertial effects non-significant. This is a valid assumption as long as the object-table friction is high and the object velocities change slowly.

[2] Except, as mentioned before, when a local minima appears. For a local minima to appear, usually more than one obstacle is necessary, and the obstacles must be near to each other

2.4 Mathematical Models of Planar Sliding

Planar sliding has been a problem studied extensively [Michalowski and Mroz 1978] [Mason 1982] [Goyal, Ruina, and Papadopoulos 1989] [Mróz and Stupkiewicz 1994] [Goyal, Ruina, and Papadopoulos 1991] [Hjiaj et al. 2004]. One of the most important initial friction models was shown by Charles-Augustin de Coulomb (1785) (possibly based on Leonardo da Vinci and Guillaume Amontons previous works on friction), this model is limited to the case of a contact point or when motion is straight (figure 2.3).

The problem with Coulomb friction is that it is insufficient to predict rotational motion with friction in case of contact surfaces. For this situation, calculus must be applied. [Mason 1982] and [Goyal, Ruina, and Papadopoulos 1989] developed the theory and mechanical analysis to describe the planar sliding motion when a force is applied to an object. To find a solution to this problem, it is required to solve three double-integral equations over the surface of the object for every possible instantaneous center of rotation (COR). Typically, the surface is sampled for some center of rotations (CORs) and an approximate surface plot can be generated (figure 2.4). This produces a surface, called the limit surface (LS) which is analogous to the Coulomb friction law but extended to more general frictional contacts (surfaces). The limit surface (LS) is able to relate applied force/torque to velocity direction in a particular point of the object. The limit surface (LS) doesn't directly calculate the magnitude of the speed of the movement.

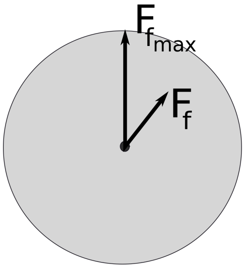


FIGURE 2.3. Coulomb friction.

The limit surface (LS) provides a way to analyze and predict object behavior based on applied forces, but the model presents some difficulties when applied to real objects. The limit surface is constructed by **sampling** the center of rotation (COR)'s space, and the extraction of the normal is also a **search** in the resultant surface. This is so, because there's no analytical general solution for the double-integral equations over the contact surface for every possible instantaneous center of rotation for every possible object shape. An additional problem is that all this has to be calculated again for every new pushing point of contact (PPC). For this reason it's worth to fit this surface to a more parametric representation. [Howe and Cutkosky

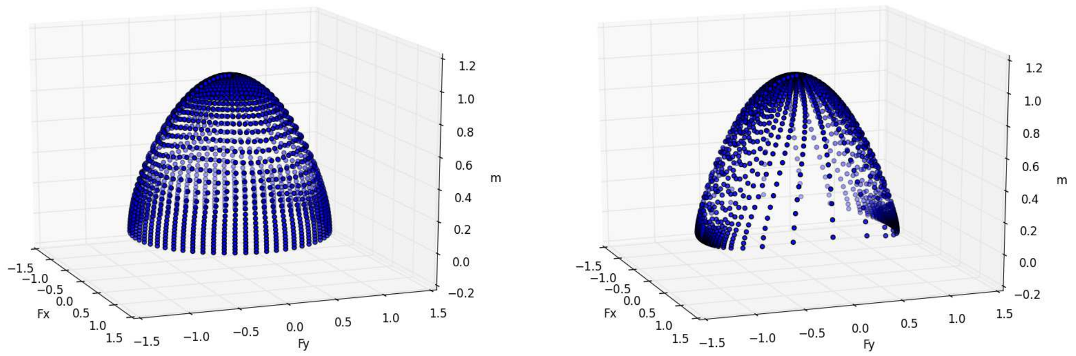


FIGURE 2.4. Limit Surface for a box. Left: Dimensions 1x1, Right: Dimensions 4x1

1996] fit the limit surface to an ellipsoid. They derive some conclusions and guidelines on using an elliptical approximation and its drawbacks. One drawback is that the precision of the approximation decreases when the object contact surface is more discrete (like a tripod) or more elongated (like a squared bar, lying horizontally). Particularly useful for us, is the fact, that since the ellipsoid is a parametric description, a formula for its normal is available. This will be used in section 4.2.2 to derive the planar sliding kinematic model.

[Mason 2001] explains the qualitative characteristics of pushing objects. He describes under which circumstances an object will turn to the right, to the left, if the finger will slide to the right or the left, and so on.

Using Goyal's limit surface or simplifications of it, [Lynch and Mason 1996], [Lynch and Mason 1999] and [Bernheisel and Lynch 2006a] are able to accomplish stable pushing tasks. These works have in common that they try to exploit the friction between the actuator and the objects to achieve naturally stable pushing. Since movement is stable, it is easily reproducible and therefore, they can use a motion planner (open loop) to guide the objects to the desired goal. The planners are careful to maintain the system in this stable pushing configuration, but respecting some minimum and maximum control values. Instability happens when the manipulator looses fixed relative position to the object. In our work we wanted to accomplish one single contact point pushing with only forces and not torques to move the object, therefore our system is working in an unstable configuration; while pushing, any disturbance, error or imprecision can move the object rapidly away from our desired trajectory. [Peng Cheng 2008] use uncertainty and robustness to slide an object using robust stable *motion primitives*; they use a simplified three-support-points friction model. The stable motion primitives always involve two pushing points of contact, and they explain that when there's only one pushing point of contact, robustness is lost and motion becomes “unpredictable”.

Previous work on feedback controllers for planar sliding with only one contact consist on 1) using a linearized model that only works for a particular pushing point of contact and can do trajectory tracking control [Lakhani 1990] [Alexander and Lakhani 1996] and, 2) using a finger tip that can detect angle of touch and therefore orientation of the object, execute a stable constant trajectory controller, which basically means maintaining the angle between the object and the finger constant, but no direct control over the trajectory is explained [Lynch, Maekawa, and Tanie 1992].

In our work we extend on the previous results by designing, implementing and testing successfully a point stabilizing feedback controller, that adapts the pushing forces even when the pushing finger changes position with respect to the object surface (the finger can slide on the object).

2.5 Summary

This chapter presents a discussion of previous work related to skillful manipulation, modelling object behavior and the particular pushing action. The problem of mapping high level instructions to low level motor control is of great importance and affordances, inverse and forward models and Object Action Complexes (OACs) will play a key role in this thesis to derive the methods explained in the next chapters. In particular, white box learning combined together with the concept of inverse and forward models will be of special importance to develop the concept of Object Models (OMs) in chapter 3.

The pushing action is shown to be increase robustness and complement many more tasks such as grasping. Previous work related to the different sub-movements of pushing and, in particular, the sliding motion are discussed in detail.

CHAPTER 3

System Architecture and Object Models (OM)

To generate the low level motor control from a high level instruction we need a system that is able to understand instructions coming in the form of a language (an *action language*) and translate them into motor commands. These motor commands will strongly depend on the object being manipulated and the action to execute on/with the object. To relate the object with the action to execute on/with it we make use of the concept of forward and inverse models (chapter 2). A mathematical description that relates a forward and inverse model of a particular object could provide a method to control and predict object behavior. If such mathematical description also integrates an object description that is carefully designed (object parameters that are meaningful), such system could map the object and goal properties present on the high level instruction (type of object, particular object, final desired state) to the low level motor commands in a simple manner. We call this system Object Model System (OMS) and a particular model of an object we call an Object Model (OM).

This chapter presents a system architecture that shows how an Object Model (OM) could be used to translate a high level instruction to low level motor commands. Also presented in detail is the Object Model (OM), how to build it and to use it. We will go deeper into the questions stated in section 1.1. To demonstrate the feasibility of Object Models (OMs), in chapter 4 a model for a *box sliding on a table* composed of two internal mathematical descriptions is presented. Later chapters (4 and 5) will quantify the prediction and control capabilities of such model experimentally.

3.1 System Architecture

The complete system is show in figure 3.1. A reasoning, planning and knowledge system preprocesses the commanded instruction to transform it into a lower level instruction, this lower level instruction can be presented using a special representation (an action language). This lower level instruction is taken by the Object Model System (OMS) and processed to generate motion commands at object coordinates (desired direct object movements). Finally these object motion commands must be converted to robot motor commands (typically using inverse kinematics).

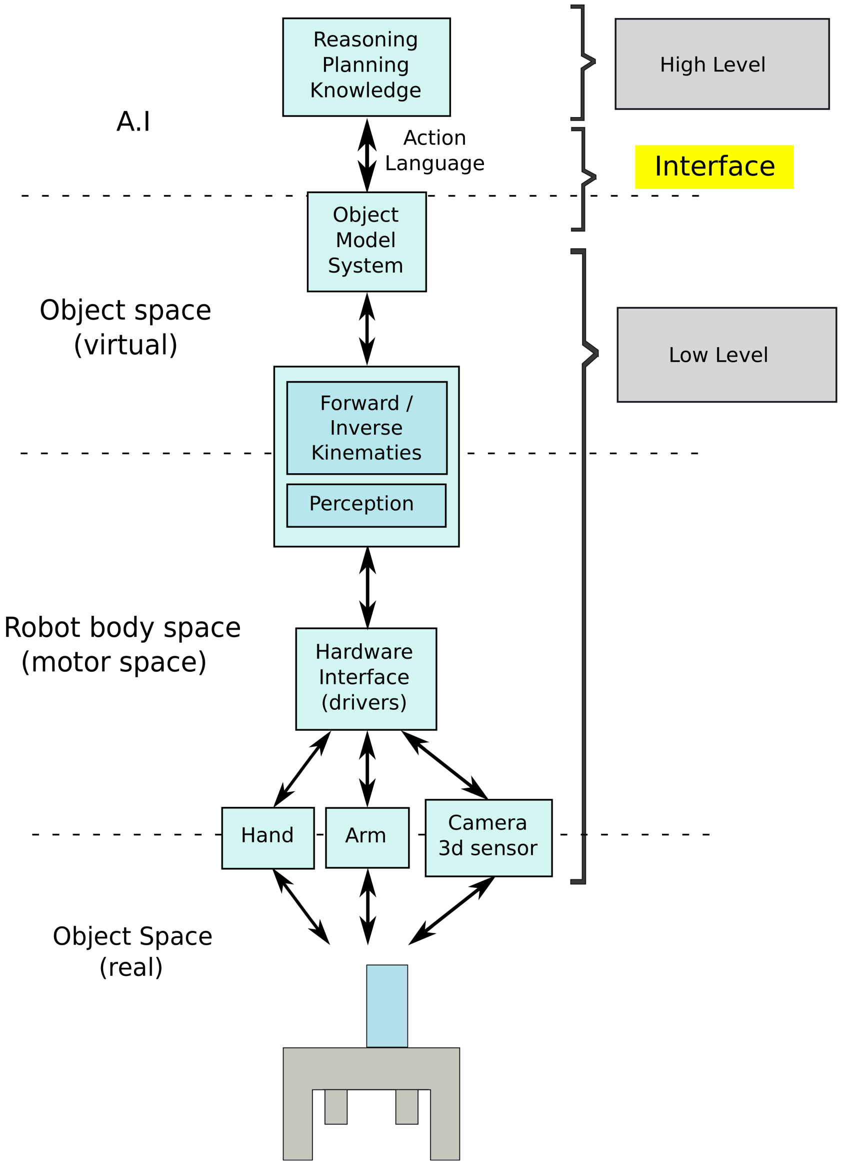


FIGURE 3.1. System architecture.

More in detail, when the reasoning, planning and knowledge system produces a lower level instruction the Object Model System (OMS) maps the different parts of this instruction into parameterizations, model selection, object selection, and property selection. All this will in turn completely specify the Object Model System (OMS) (figure 3.2).

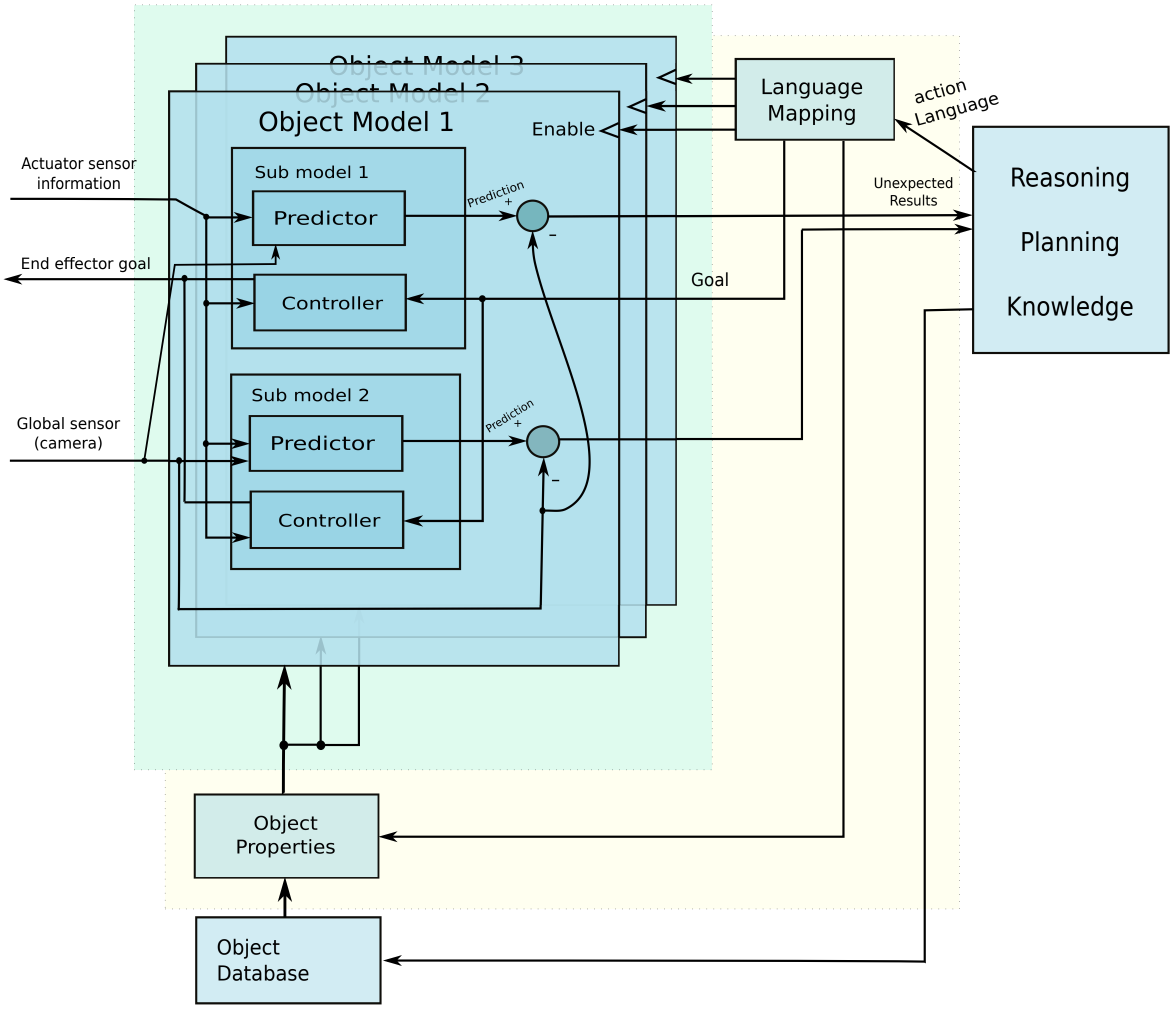


FIGURE 3.2. Object model system.

The output from the Object Model System (OMS) consists of motion commands in object coordinates (object space); the system doesn't control the motors of the robot directly, the Object Model System (OMS) commands the motor control system to move the end effector (the touching contacts) in some particular ways to achieve the desired object behavior.

The Object Model System controls the object, not the robot.

Thus, the instructions generated by the Object Model System (OMS) must be translated by a robot motor controller to fit the particular robot body. One typical example of a motor controller is velocity inverse kinematics; given a desired end-effector velocity (that acts on the object) this controller calculates the necessary motor velocities.

The movements generated by the motor controller on the end-effector, touches and changes the object state. This object state is perceived through the use of sensors. This sensor information may be used for the motor controller, but most importantly it is used by the Object Model System (OMS) to predict future behavior. The sensory information needs to be in the *object space* to be useful for the object model, therefore, it may need to be processed before it is given to the Object Model System (OMS).

The perceived information is processed in the Object Model System (OMS), and future (expected) behavior is calculated as a result. This expected behavior is compared with what really happened and if there is a big difference an alert is generated (unexpected behavior). This alert is dispatched to the high level systems to reason about it and decide the best next (corrective) action to take. An additional association map, that maps the unexpected behavior or results from execution to the a high level description language, may be needed.

Blocks in the light-green area of figure 3.2 have been implemented extensively in our *box on top of a table* example explained in detail in chapters 4 and 5. The light-yellow area has been implemented in a simple way, with the purpose of exemplification. In the future, when we increased the quantity of object models, we expect to need a more sophisticated implementation for the yellow area blocks, in which case we would consider integrating our Object Model System (OMS) with the systems described by [Mösenlechner and Beetz 2011], [Tenorth and Beetz 2013] and [Tenorth and Beetz 2010].

Figure 3.1 also shows how all the parts of our system correspond or fit into the cognitive levels of abstraction (see figure 1.2 of chapter 1). One can see how part of our Object Model System (OMS) together with the motor controller and perception loops correspond to the low level section. The Object Model System (OMS) together with the action language correspond to the interface between the high and the low level sections.

Our system is strongly motivated on the MOSAIC model described by [Haruno, Wolpert, and Kawato 2001]. Such system is focused to be used on the low level motor control. We addapted such architecture to be used in our model-based approach to be used as an interface between the high level reasoning and planning and the low level motion control. Our system also examplifies a real world application of the ideas of an inverse and forward model.

3.2 Object Models

The Object Model System (OMS) will select one (or possibly multiple) Object Model (OM) to operate. Once this selection is made, the Object Model System (OMS) main task is to predict and control object behavior.

An Object Model (OM) is a mathematical description of the behavior of an object. From a mathematical and systems analysis perspective an Object Model (OM) is a dynamical system where the inputs to the system will be anything the robot can transmit to the object, and the outputs are any physical variable that the robot can measure or infer and that are significant to the given task.

If a robot has an Object Model (OM) for the current object, then it is able to immediately **predict** the object's behavior based on available perception information, and for example could answer the question of what happens when the robot applies certain force on a particular object location, if the object is on top of a table (Figure 3.3). Also the Object Model (OM) will give us the tools to derive a feedback controller to manipulate the object.

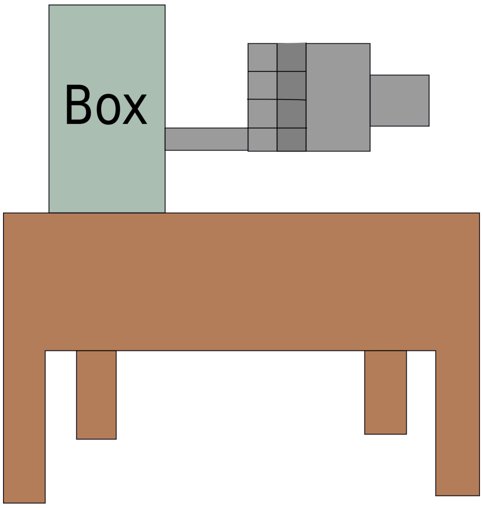


FIGURE 3.3. Robot applying force to a box at a particular point.

This prediction-control system has many similarities to the forward and inverse models mentioned by [Wolpert and Kawato 1998]. The idea is that the robot can constantly monitor the prediction output to see if everything is happening as expected, and if not, it is able to detect out-of-the-normal situations (and act accordingly) or improve the prediction and control. We used a similar system (shown in figure 3.4) than the one described by [Wolpert and Kawato 1998] to implement our Object Model (OM).

Predictions could deviate from the perceived consequences because 1) the predictor is not

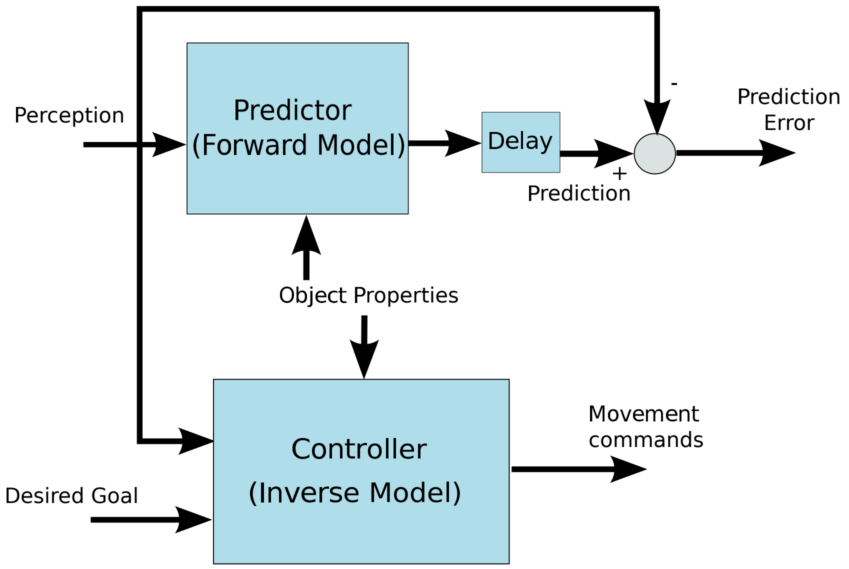


FIGURE 3.4. Forward and inverse model.

perfect or because the system changed (because of aging), 2) an external perturbation that is not modeled or taken into account. Under this situations a robot could adapt its behavior respectively by 1) improving or updating the predictor, and/or 2) taking active corrective actions to improve the situation.

With respect to the practicality of object models, several questions arise:

1. What method(s) can be used to build an object model?
2. How feasible it is to construct an object model?
3. How detailed must the Object Model (OM) describe the object's behavior?
4. How much can the robot control the object?

The first question will be discussed in section 3.2.2.

The feasibility of making an Object Model (OM) depends on the method used to build the object model and how much of the object behavior we want or need to describe. In general, as more details of the behavior we want to model, the more difficult it becomes to obtain a compact mathematical description.

The detail of the object model will largely depend on the task and on the necessary/desired precision/accuracy[1] for prediction and control. Experimental testing will be used to determine if the object model is detailed enough, i.e., if predictions satisfy the precision/accuracy needs (which depend on the capability to differentiate between normal behavior and unexpected or undesired behavior), and if control of the action outcomes on the object is as desired. Therefore, as for now, this is a refinement process, that requires to design the model, predict, control and see the results, and if necessary improve the model and try all this again (figure 3.5).

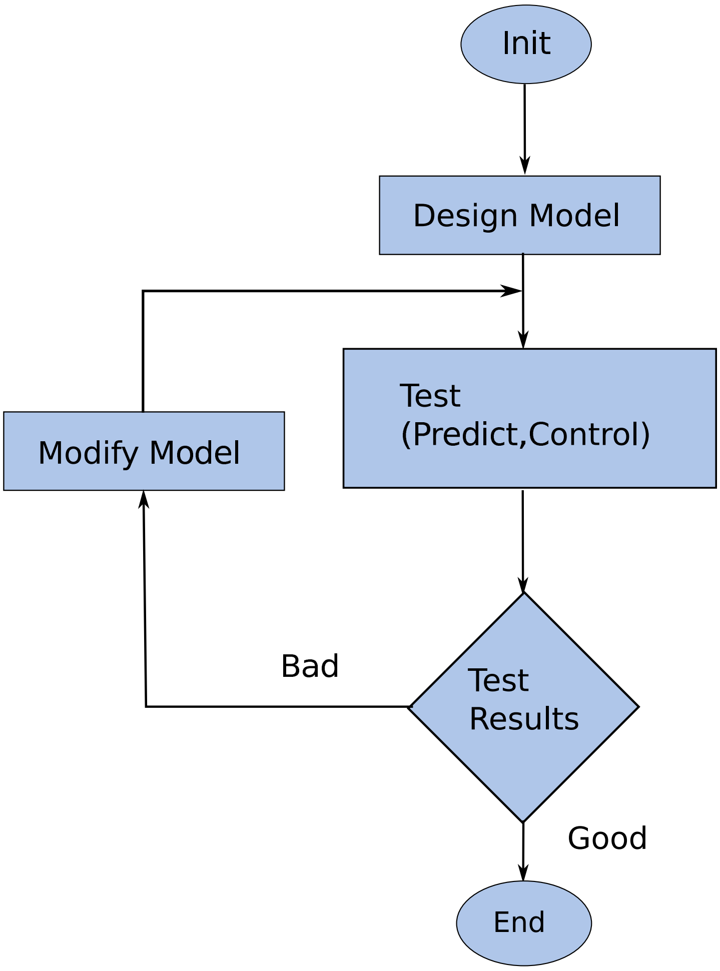


FIGURE 3.5. Design Workflow.

The body of the robot (the forces it can exert, kinematic reachability, capability maps [Zacharias, Borst, and Hirzinger 2007]), type and size of end effector actuator, the object safe limits among other things will mainly determine how much the robot can **control** the

[1] In general, we will use precision or accuracy in this document to mean both at the same time, but the main meaning is accuracy because imperfections and/or incompleteness of the model, according to our experience, will most strongly affect the accuracy of the prediction results, while the factors that determine precision are less related to the model, and probably more related to the robot body.

object. With respect to the object safe limits: what are the control limits for a particular object? It depends on what we want from the object, if we want to maintain structural stability and functionally intact, the object will certainly have some maximum control limits (think of grasping an egg to store it). We may have to increase these limits for some actions (break an egg to cook it). But in general, under this object safe limits the robot can do as much as it wants as long as the robot body is able to. This range of possible actions that this robot can do with a particular object are the **affordances** ([Gibson 1986] and [Norman 2002]).

If we want to manipulate an object we usually know what we want from it, i.e. what would be the final state of the object. This final state (goal) and the object will in turn define which forces must be applied to the object to generate this final state. In principle,

> *The goal or intention together with the object strongly determine what the robot must do to the object* [2].

After that, the robot will know which forces must be applied to the object to accomplish the task[3]. To apply such forces the robot can use well-known control schemes or more novel approaches to robot motion, this can be even dependent on the particular robot. From our object-system perspective it doesn't matter as long as it reproduces the force (or movement) as needed.

It may happen however, that the robot can not exert such particular force or movement because it is not strong, delicate enough, or because it is not kinematically feasible. This will be part of the affordances of the object, which will change constantly depending on what is possible or not. If the robot is able to recognize its own limitations and how objects react to his actions, then the robot becomes aware of the object affordances. The robot will be able to know what actions it can exert on the object and the corresponding consequences, therefore determining the complete usefulness of the object.

3.2.1 Object Properties and State Variables

Are all the object properties necessary? The desired task at hand, and the object will determine which particular object model will be used. This model will in turn determine the **necessary**

[2] The environment and the robot body plays also an important role. The environment factor is integrated in our object model: the table is incorporated in our model through the use of the table-object friction coefficient. The robot body may affect the way the robot may apply the necessary forces. But in essence, the object (with the inmediate interactions with the environment) and the goal will determine the actions of the robot on the object

[3] This is a simplification because there may be several ways to accomplish the same task (using the nullspace of the system), and this can be used by the robot to select one solution that fits some optimization principle for the robot body [Kresse and Beetz 2012].

object properties and together with the particular controller goal which forces must be applied to the object. It's important to remark that not all object properties are necessary for a given task, and that under some tasks a particular object property may become an state variable. For instance, for cooking, the temperature of an object may be important (an state variable and goal variable), but it's not important for a pushing action (temperature is nor an object property neither an state variable) [4].

It is important to note that the possible state variables that can be controlled may strongly depend on the current end-effector. A robot can therefore, change this by using tools. For example, cooking requires to use an oven which can produce enough heat to cook, which the robot alone would not be able to do. Therefore, tools represent a way to extend object affordances.

3.2.2 How to Build an Object Model

To build an Object Model (OM) we could use 1) black box or 2) white box learning. We will discuss them and draw some conclusions to decide which one is more convenient.

In **black box learning** the robot would look at the inputs and outputs of the object system and learn the relationship using a carefully selected black box machine learning algorithm. Since machine learning algorithms work by basically trying to best fit an internal model to the observed input-output relationship, it will describe the Object Model (OM) very well (accurate). But there are some inconveniences when using black box learning, the most important one is the dimensionality problem: The black box learning algorithm could learn an Object Model (OM) of a particular object instance, but as soon as we start to add to the system more objects of the same class but with different properties, then the system could take forever to learn the model, because for each property an additional dimension is added to the learning problem, also it would be necessary to show the robot too many different objects and poke with them before it makes a complete enough Object Model (OM). Maybe the objects are not even available at the moment.

To worsen this situation, our experience tell us that most black box machine learning algorithms usually seem to require an important quantity of training examples (compared to white box learning), increasing even more the learning phase. For example, the robot would have to learn the model of an ice tea box by playing with boxes of different sizes, weights, fill levels, friction coefficients against the table, different tables; the combinations are huge. This poses a terrible practical problem. Humans appear to do this to some degree, and one can see babies

[4] Unless it is too hot or cold and this may harm/damage the robot hand. (this may be implemented as a monitor (exclusively predictive model) that always runs for protection).

and kids playing, touching, throwing things constantly; they must learn with a huge combination of possible objects and situations. It may take a lot of time and training [5]: no wonder why babies are so dependent on their parents for so long and also why it takes so long to babies to develop good general manipulation skills. With black box learning there is also the problem of over learning, modelling noise and unimportant features. But, the most important limitation is that the model is not easily parameterizable for different object properties. You can not easily change the model learnt for an ice tea box with 1L of tea and 2L of tea to a model of a 3L of tea. In this case the robot will have to train again with the new box.

In **white box learning** of the Object Model (OM), the robot is given a carefully designed Object Model (OM) and the robot only has to adjust the internal parameters (properties) of this model. Afterwards, the model is ready to be used. If the perception detects an ice tea box, then an ice tea box Object Model (OM) is selected, the robot then needs the parameters of this models to be able to predict (to complete the model). The parameters are the object properties that identifies this particular object type instance, in case of an ice tea box sliding/toppling situation, an object model would need weight, object-table friction coefficient, finger-object friction coefficient, object size and object shape. This parameters can be given to the robot by a knowledge database or acquired in the moment by poking a bit with the object, or even a combination where the robot uses a rough initial estimate (from a knowledge database) and starts using the object, and during the interaction with the object the robot may be able to calculate and update the properties of the object by direct measurement or to better fit the results. The poking/playing actions can be directed to find this parameters as fast as possible, usually requiring only one or two repetitions per Object Model (OM) parameter.

The white box learning approach for the Object Model (OM) has several advantages:

1. It is very compact, only describes what is needed for the current task, and therefore only relevant constants are required, and control parameterizations are limited to the given task, simplifying the interconnection between the high-level reasoning and planning system and the low-level system.

2. It can be grounded to the robot because the robot can poke with the object to find the Object Model (OM) parameters, this means that the Object Model (OM) is adapted to the sensory information that this particular robot has. But if other robots have sensors calibrated in a similar way, these parameters could be shared with them, avoiding the poking actions and allowing other robots to start doing useful work right away.

[5] Some machine learning technics may be able to interpolate to obtain results even when not exposed to sufficient training examples, but results may not be predictable, and chances are that results are different each time a machine learns the process [Mitchell 1997].

3. Very few training examples are required. If the robot gets the Object Model (OM) parameters externally, then no poking is needed at all, but if the robot has to do the poking itself, then the number of movements are usually very small and limited to find the parameters and not the Object Model (OM), for example to find the weight, the robot could simply grab the object and lift it, or tilt it a bit. With one single action it already gets the value of a constant.

We decided to use the white box learning because it has the potential to connect the Object Model (OM) system with a high-level reasoning and planning system using an **action-language** with simple action specification parameters indicating the object properties and the task definition. This decision doesn't exclude us from also using the black box learning approach in some situations. For instance, if the robot is going to perform a given task with the same object very often, it would be useful to have a more precise model for prediction and control purposes. In this case the Object Model (OM) could be learnt using black box learning in parallel with the white box learning, or the errors of the white box predictor could be learnt using black box learning. We think of this as a *refinement stage*.

In following chapters we will show how to build such an object model using a white box learning approach. The most challenging aspect of building objects using a mathematical/mechanical description is that objects are *macro* entities that will usually require multiple integrals over surfaces.

Usually most of the mechanical/physical formulae is in terms of particles or point-wise models. In physics this is not a problem, even simulations (and therefore predictions) are possible using particle analysis (or finite elements). This works well [6] for prediction but for deriving feedback controllers it becomes something very difficult, especially for non-linear behavior. The system is huge, having almost all the complete degrees of freedom for each particle. Therefore, the difficulty lies on finding analytical solutions or good enough compact representations of the behavior.

We choose one of the most difficult behaviors with everyday objects: friction. And we will show the most important models related to object behavior in this respect in later chapters.

3.2.3 Using the Object Model to Control

If we want to control the object, then our approach is to use the Object Model (OM) and try to derive a stable feedback controller that solves the given task. The difficulty to design such controller will depend on the Object Model (OM) dynamics. Because we are working

[6] Usually requiring huge amounts of computational power, therefore very difficult to run in real-time.

in a setting where everything can happen, we also want a global controller if possible. If the system is holonomic and linear, then the design of the controller should be easy and to some degree possible to automate, but when the system is non-linear and also non-holonomic then it is more challenging. In the case of pushing with one finger on a table surface the system is non-linear, non-holonomic and time-variant [7] and the derived controller takes this into account to produce the desired behavior.

An additional feature our system can have is that the control could be improved by means of **predictive control**, where we could use the predictor output to anticipate or complete information that can be useful to the controller.

3.2.4 Accuracy of the Object Model (OM) Predictions

In general, for prediction, the more precise the model is, the better it is. In practice a very precise model is not necessarily more convenient or useful. But before going into details here, the factors that could affect this precision will be explained.

3.2.4.1 Factors that Determine the Object Model (OM) Prediction Accuracy

The accuracy of the prediction can depend on several factors, but most importantly, it depends on 1) how detailed we are describing the object behavior, 2) how noisy and distorted is the robot perception and 3) how sensitive to initial conditions the corresponding model is.

If the system is very sensitive to initial conditions, the prediction quality will decrease rapidly as we try to predict further in the future. If the Object Model (OM) is very inaccurate the predictions on any future time will get worse faster. As a result of this, the qualitative prediction after some time could go completely wrong. In general, we can only trust the predictions up until some future time.

There are two possible sources of imperfection on the Object Model (OM), 1) unmodeled behavior, and 2) inexact object parameters.

Unmodeled behavior consists of object reactions that are not taken into account in the mathematical description of the Object Model (OM). There can be two possible reasons to have unmodeled behavior: because we explicitly don’t want to model such behavior (because it’s not interesting or important to the given task or because it’s difficult to model), and/or because we were not aware of such behavior (either because we didn’t notice or because it is difficult to measure). The imprecision and inaccuracy of the object parameters will depend on the way

[7] One of the most difficult types of systems to model and control.

the robot acquires this knowledge and the quality of the sensors, calibration and perception algorithms.

Because we desire to have the most accurate mathematical description (for better prediction), and yet only the most meaningful parameters to set (for high level reasoning coupling), and a simple enough prediction and control system, a compromise between accuracy and simplicity of the model must be reached. For example, if the task is to slide an object from a particular position on a table to another, we desire to only have to set object parameters that the robot can measure or acquire, that are important to the task, and that can be used for reasoning. In our case, weight, dimensions and friction coefficients should be enough. We could derive a more complicated model that takes into account object deformation, liquid oscillation, temperature, and so on. This may improve predictability (assuming the robot can acquire all the parameters), but will give no important improvement for the controlling and reasoning parts. In fact, it will most likely complicate things unnecessarily.

3.3 Action Language

As we mentioned before, having an object model that is compact and has meaningful parameters gives our system the possibility to connect it to a high level reasoning and planning. We are considering that this may be possible defining an action language and a mapping that connects the different parts of a sentence in such language to the controller selection, current object, the object properties and desired outcome (Fig. 3.6).

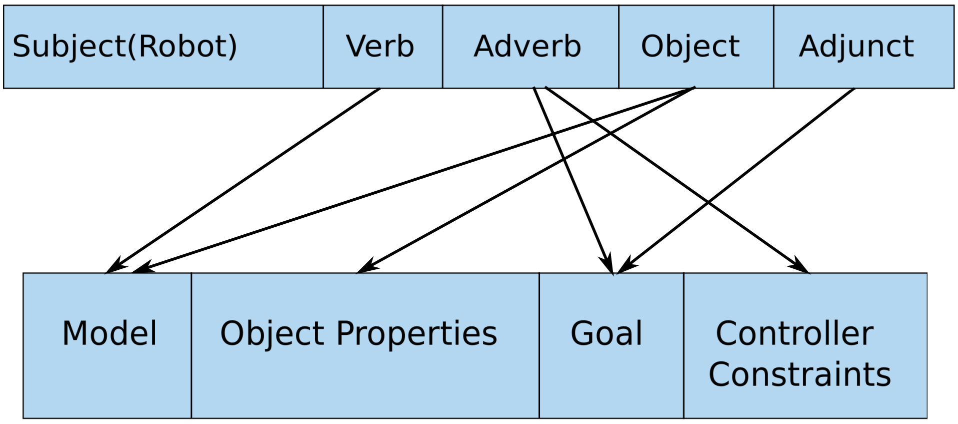


FIGURE 3.6. Action Language mapping.

Previous experiences (stored in a knowledge database [Tenorth and Beetz 2013]) could be used to help reason about what things can be done with the objects (based on previous action outcomes or simulations with the object models) and based on this decide the best current course of action. It also helps to determine more small refined sentences to accomplish a higher level instruction. For example “Push the ice tea box away”, first requires to “look for” where the ice tea box is on the table in order to push it. This action of “looking for” is not explicitly specified in the original sentence, but it’s a requirement for the pushing action to finish successfully. It also requires to know that there must be a table and what a table is and where the table is. Also if there are other objects on the table and the robot can not push the ice tea right away, the robot must first be aware of this situation and adapt its behavior by inserting additional pre-instructions before the actual final instruction. Also once the robot is really ready to “push the ice tea box away”, missing still, is to determine what “away” and “push” mean. Therefore, a system for preprocessing sentences into more specific sentences and plans is necessary.

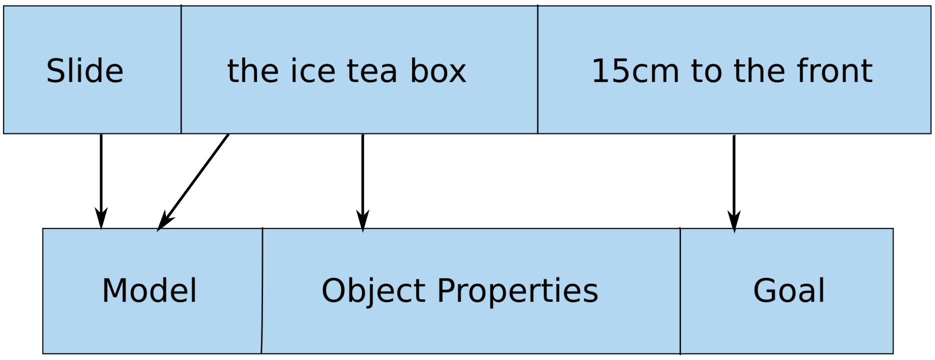


FIGURE 3.7. Mapping example.

Once the specific *movement sentence* is presented it must be transformed into something more quantitative, like “Slide ice tea box 15 cm to the front”. Such a sentence can be more easily mapped (Fig. 3.7) to the system described in this thesis. A lot of work regarding the segmentation of a complex action into smaller simpler actions has been address in [Müller, Kirsch, and Beetz 2007], [Tenorth and Beetz 2010] and [Mösenlechner and Beetz 2011].

Current work is addressing how to do the mapping of a the specific *movement sentence* with the Object Model (OM), and we will clarify possible ways such mappings could be done with the object models described along this thesis.

3.4 The Object Model and the Robot

Up to now, we have talked about the Object Model (OM) but not about the robot. Is the robot body a factor in our Object Model (OM) system? Are the kinematics of the robot important to the Object Model (OM)?

The object model is designed in such a way that it is abstracted away from the robot (as much as possible). The object model reference frame is relative to the object. This makes the model more robot independent. One important aspect of our solution is that the robot can extract the object properties itself, by “poking” with the object in different ways. If this is done, the object properties will be “calibrated” to the same robot sensors and therefore, we can have more precise prediction and control.

There are some considerations that must be taken into account when choosing the *right* robot.

Mechanically, the robot 1) must be able to exert the necessary actuation given by the Object Model (OM) controller output, and 2) have sensors to acquire object and context state. Since our proof-of-concept example is pushing, we need a robot that is able to push, but also able to sense force, position and speed in the pushing contact. We also need to know the object pose globally. This is necessary to be able to position the robot end-effector to exert forces in a particular location of the object. Our robot TUM-Rosie has the necessary hardware capable of all this (section A.1.3).

Also, the robot needs to be equipped to process this sensory information to a representation that the object model can understand, that is, in object coordinates. This requires forward kinematics of velocity, position and force/torque. For exerting forces an inverse velocity and position kinematics is also required (or an equivalent system). The camera or 3D sensor information must be processed to obtain a global estimate of the object’s pose.

If the robot has all this requirements then it should be able to work with an object model like the planar sliding model explained in later chapters. Figure 3.8 shows all the hardware parts.

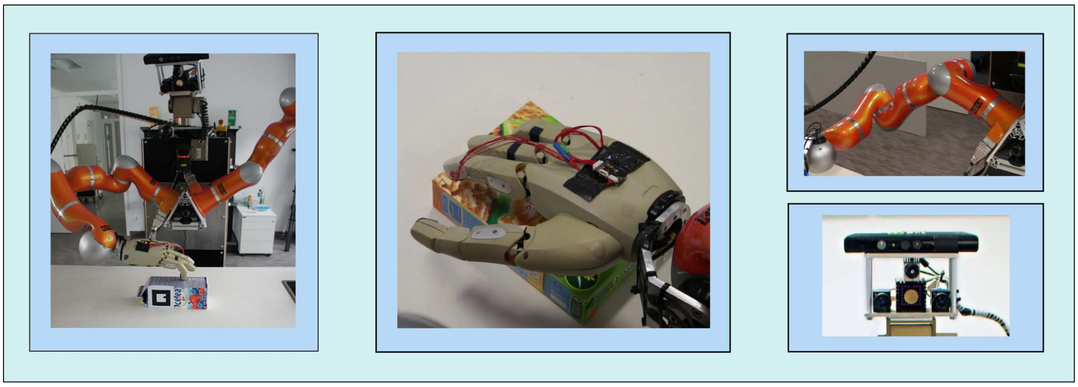


FIGURE 3.8. Hardware parts.

In practice, there are a lot of challenges to get such a system running, and that it gives proper sensory data and actuation. Noise levels, unmodeled non-linearities, construction imperfections, uncalibrated sensors and joint angles, crosstalk between torque sensors in different axes, bad documented interfaces and specifications, can complicated results a lot. Particularly challenging for this work was to get good enough force data at the finger tips. This will be explained in detail in sections A.2.3.1 and 4.4. Also difficult is to get a precise enough global object pose estimation. For this we used a camera-based marker tracker (artoolkit plus, section A.2.3.2).

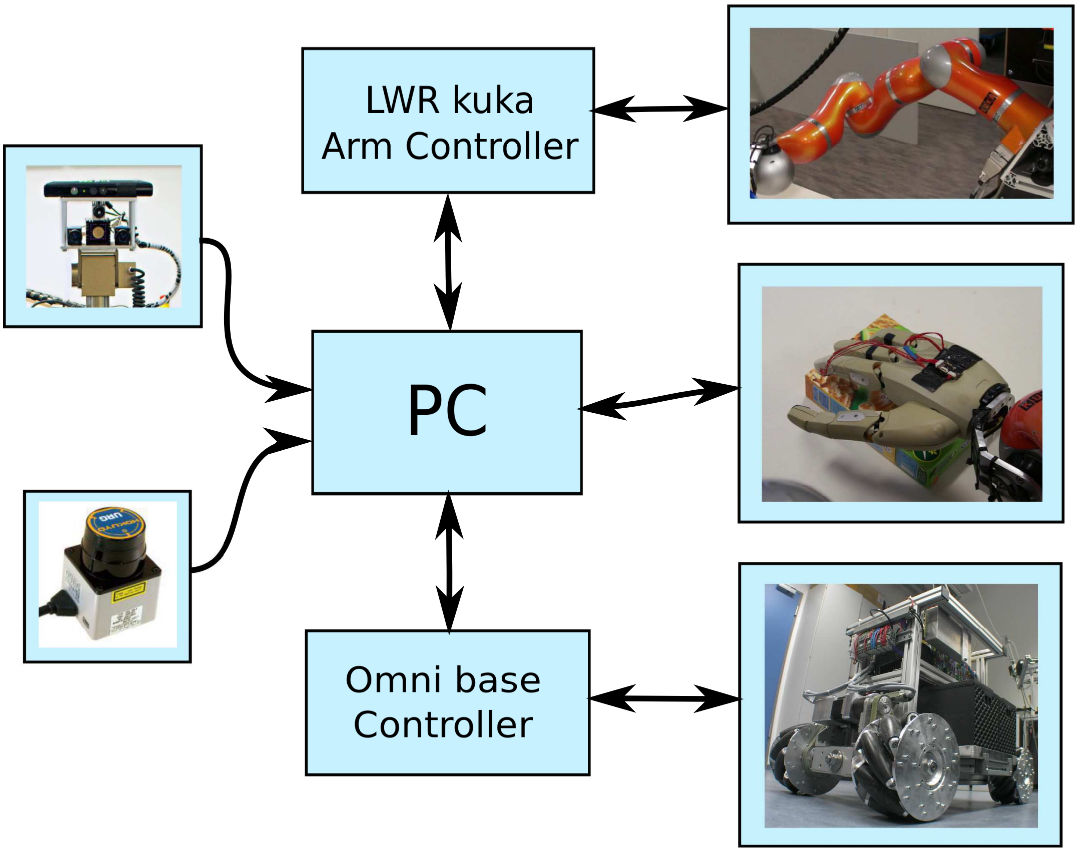


FIGURE 3.9. Hardware diagram.

This system can be described in terms of hardware and software parts. A hardware diagram is shown in figure 3.9. A software diagram is shown in figure 3.10. The software modules are interconnected using ROS and YARP and almost all modules are open source and can be downloaded from [Ruiz-Ugalde, IAS-TUM, and ARCOS-Lab 2013a] [Ruiz-Ugalde, IAS-

TUM, and ARCOS-Lab 2013b] [8].

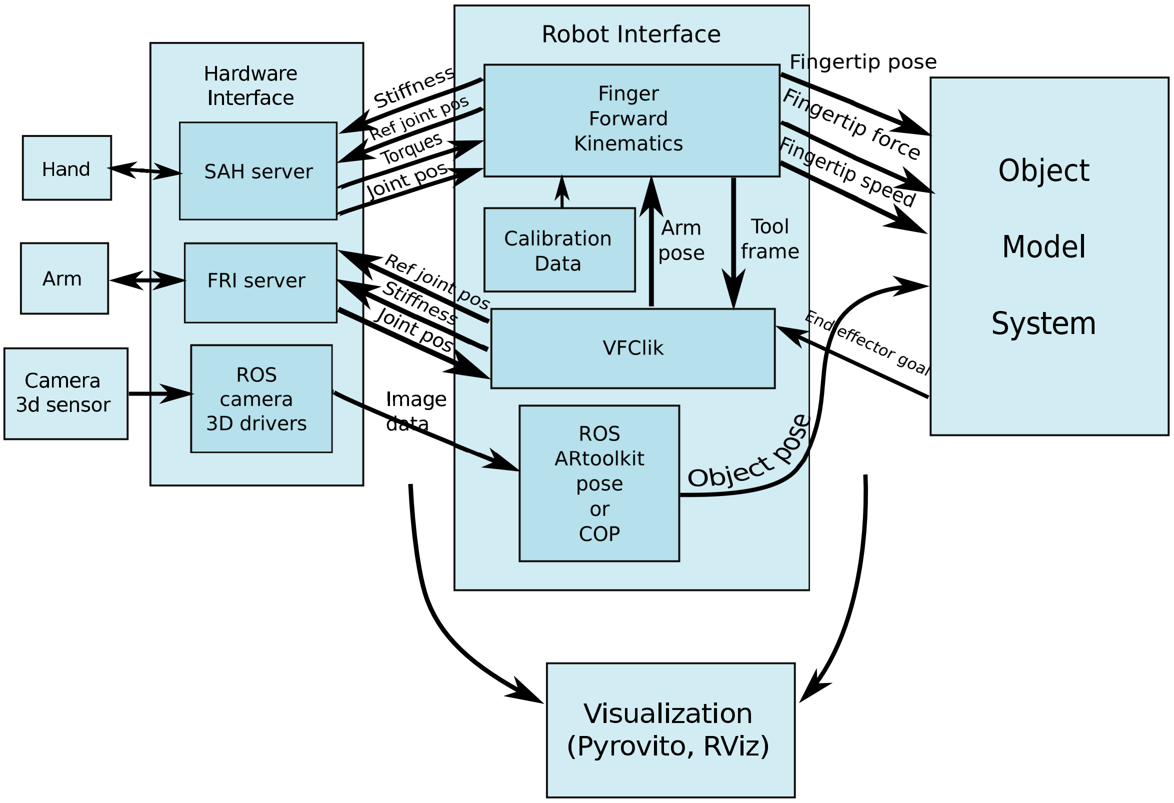


FIGURE 3.10. Software architecture.

3.5 Summary

The design of the Object Model System (OMS) is based on the inverse and forward models studied in chapter 2. In this chapter we show how this design could fit together with a high level reasoning and planning system and a low level motor control system. The Object Model System (OMS) acts like an interface between these two worlds translating instructions into motor commands, and perception information into high level signals.

An Object Model (OM) is designed using the white box modelling approach explained in chapter 2. This is convenient because the model is very compact, it *grounds* the model to the robot and very few training examples are required to be able to use the model.

A discussion of how the Object Model (OM) can be interfaced with the high level reasoning

[8] All modules for running a simulated environment of the robot and the object are completely open-sourced. Only parts containing or using libraries for communicating with some parts of the hardware may be proprietary and can not be released to the public.

and planning system using an action language is presented. Because we used a white box approach to design the model the Object Model (OM), and its parameters are very compact, the mapping will generally be very straight forward.

There will be some minimum requirements for the robot as shown in section 3.4. If these requirements are met, the robot may be able to extract itself the object parameters. This allows the robot to *ground* the object knowledge to the real object and its robot body.

CHAPTER 4

A Test Case: Pushing A Box (prediction)

Pushing objects is a general and basic capability of humans. It allows us to move objects that are usually on top of horizontal surfaces (tables). A person may push an object in order to bring it near to use it, to push it away to clear the space to work with something else or to help another person by pushing the object within his reach. Pushing may be more appropriate than picking up and placing in some situations; it may be more comfortable or only possible at all to first push an object than to pick it up, it could possibly also consume less energy, or require a smaller working space for the manipulator [Chang, Zeglin, and Pollard 2008] [Chang, Srinivasa, and Pollard 2010].

An important phenomenon during pushing is friction. Friction is ubiquitous in our everyday life. Without it, manipulation, in general, will be completely different; friction, in many ways, facilitates manipulation. When we push an object that is on top of a table, friction becomes extremely important in the description of the object's movement.

To achieve skillful manipulation during pushing, the robot must ensure that the object motion and final goal is as desired. A stable push can be achieved when using two or more pushing points of contact (PPCs) [Bernheisel and Lynch 2006b] [Lynch and Mason 1996] [Kumar 2009]. Although, using two or more pushing points of contact (PPCs) may be possible in many situations, we are interested to have a robot capable to use and predict object's behavior reliably in as many as possible situations, therefore, we are interested in the more general situation where any force and torque is applied to the object. Such situation is best exemplified when one pushing point of contact (PPC) is used. In this situation, to obtain a stable push, a feedback controller is necessary. Also, the predictive Object Model (OM) becomes a dynamical system (a system of differential equations), and therefore, a more sophisticated example and test for our system.

An important part of our "pushing a box" example is the exploration for the object parameters. The box model needs the box parameters to complete the model; the model can only predict and control when the object parameters are known. We will show how the same robot acquires most of the object parameters itself, therefore helping to "ground" (grounding [Harnad 1990] [Coradeschi, Loutfi, and Wrede 2013]) the parameters to the robot body (embodiment [Duffy and Joue 2000] [Ziemke 2001]). Because the values of the object properties are acquired using the same system that will predict and control the object, our robot will be able to predict and control the object more precisely.

Because pushing a box 1) is an important task in everyday manipulation, 2) uses an important mechanical phenomenon (friction), and 3) requires a feedback controller to generate proper behavior (a challenging example of an skillful task) we decided to use it as our proof-of-concept example for our Object Model System (OMS); this chapter explains (with an example) how to design and develop the predictive part of an Object Model (OM) (forward model), how to "fill-in" the Object Model (OM) parameters and how to use the Object Model (OM) to predict object behavior. To realize "Pushing a box", the concepts, ideas and methods presented in chapter 3 are here heavily used; the reader is advised to review that chapter first. Chapter 5 explains the control part of the Object Model (OM) (inverse model).

We will first show some general considerations, from a robotics and mechanics perspective, that explain how to model such a system. Afterwards, we present the derivation of the predictive box model, explaining the toppling/sliding and the pushing/sliding models.

To complete the model, the object properties are necessary. At the end of this chapter the methods to obtain these properties using a real robot are explained.

Chapters 4 and 5 will explain how to predict and control the object using the model derived in this chapter.

4.1 General Considerations

During a pushing action a robot exerts a force against the object, which depending on some factors, may cause movement or not. In case of movement, there are additional factors that will determine how this movement will evolve.

The object-table-finger system is shown in picture 4.1.

When the robot is not touching the object, the only force experimented by the object is the global gravitational pull which can be equivalently assigned to the center of mass (COM). If we assume that the table is perfectly horizontal, the table will generate a reactive force, cancelling completely the gravitational force out, therefore no acceleration and movement will occur. In reality, the table is never perfectly horizontal, therefore there's always a small horizontal (to the table) component of force from gravity, not cancelled out. This small component

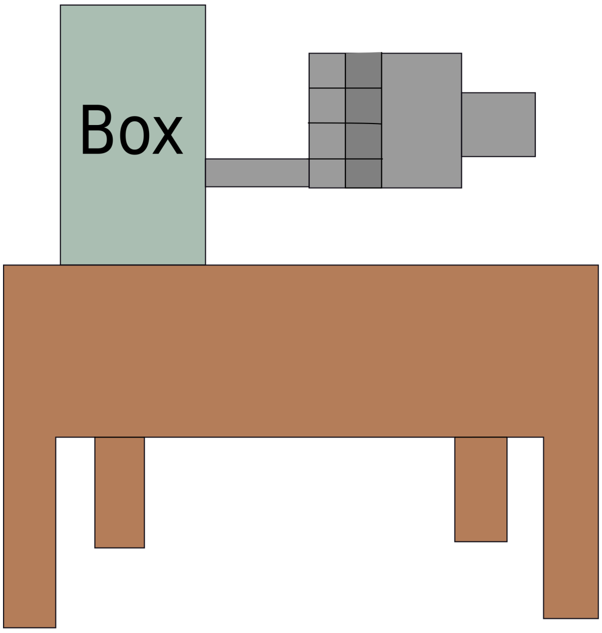


FIGURE 4.1. Robot applying force to a box at a particular point.

pushes the object sideways. If there is friction between the object and the table, there will be a reactive force against this lateral movement. This force may or may not cancel completely this horizontal force out. Since our scenario is a kitchen table, we assume that the table is sufficiently horizontal, also the friction coefficient is usually high enough to stop any possible movement produced by this, not cancelled out, gravitational force component. Therefore, the object will not slide when there is no pushing action from the robot. In our calculations, we will assume that the table is perfectly horizontal, this assumption is in general valid as long as the table has an insignificant inclination.

When the robot is touching the object, apart from gravity, now there is an additional force. If the robot aims to slide or topple the object, this force will usually have a large horizontal component. This horizontal component will interact with the friction force produced along the whole surface of the object. For now we will simplify the analysis and will suppose that the friction happens only at a single point of the object in particular at the center of pressure (COP) [Howe and Cutkosky 1996] [Goyal 1989]. In this case we have the situation presented in picture 4.2.

In this situation, when pushing starts, a typical friction force profile (in time) will look like the one depicted in figure 4.3(green). This profile corresponds to the case that the speed constantly increases.

We decided to use the coulomb friction law as an approximation to this behavior (figure

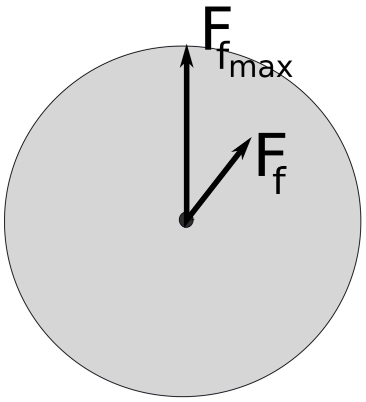


FIGURE 4.2. Coulomb friction.

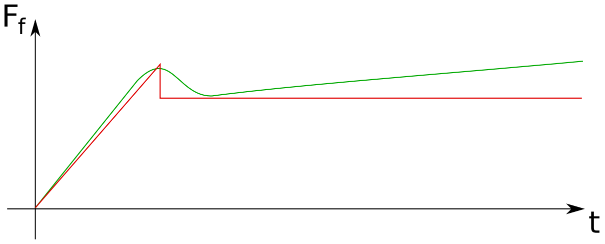


FIGURE 4.3. Coulomb friction (red) vs. typical real friction (green) vs. time.

4.3(red)) [1]. (eq. 4.3).

$$F_f \leq \mu F_n \tag{4.1}$$

The friction force depends on the normal force and the parallel force to the table. The normal force consists of the gravity force plus the normal (to the table) component of the force applied by the robot. The parallel force is the parallel component of the force applied by the robot. Picture 4.2 shows the coulomb law. The coloumb equation describes the area and the boundary of the circle of possible friction forces the object can experiment.

If the total parallel force component is bigger than the maximal friction force, given by the equation, then the object will start sliding. If the force is smaller, then it will not slide or stop sliding. If it is equal, this is a discontinuity, where motion is not defined: it is the transition from not sliding to sliding. It is also important to notice that the friction coefficient is different during sliding (kinetic friction coefficient) and when the object is not sliding (static friction coefficient). In general the static friction coefficient is bigger. Graphically, this means that

[1] There are other more complete and precise models of friction that also model, for example, the change of the friction coefficient with speed, but we consider these effects not so important in our application where speeds are usually low.

there are two circles, one bigger than the other (Fig. 4.4). The bigger one is “active” when the object is not sliding, and the smaller one is “active” when the object is sliding.

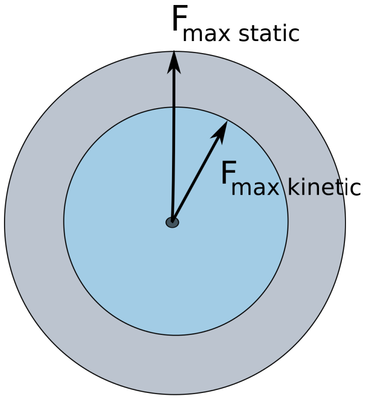


FIGURE 4.4. Coulomb friction static and kinetic.

Therefore, if the object is static, and the robot applies some force, the bigger circle is “active” and will determine the maximum force possible before movement starts. If the robot wants to slide the object, it first has to overcome this force, at this moment the object experiments an acceleration and the friction coefficient changes to the kinetic one. Since the friction force during motion is smaller, the object accelerates even more if the robot maintains the same force. If the robot wants to slide an object and maintain a particular speed, it first needs to accelerate the object to the desired speed by increasing the force above the maximum static friction force and then it has to reduce the force until it becomes equal to the maximum kinetic friction force. If the force momentarily drops and the object stops (“gets stuck”), the robot needs to start again by increasing the force and repeating the previous steps. The conclusion is that the robot needs a regulator or controller on top of the force controller. A velocity regulator usually works fine.

Once the object starts to slide, the robot keeps applying force until the desired speed is achieved. What is the behavior of this motion? It depends on the inertia of the object and the dampening produced by the object-table friction. Analysis of the inertial effects is necessary in this case. For our setup, since the speeds are low, and we use a velocity regulator, we will consider the inertial effects negligible. Otherwise, inertial effects may become more important.

4.1.1 Toppling

When we start to consider real objects, other situations, apart from sliding, may arise. For example, if we try to push an ice tea box like the one in picture 4.8, it may topple (rotate) instead of slide if we push it on a high point and the friction force against the table surface is high. This happens because a torque is generated with respect to one of the edges of the box support surface, and this torque may be higher than the torque generated by the weight of the object. Static equilibrium equations on forces and torques are necessary to explain this kind of behavior.

4.1.2 Planar Sliding

If the object will not topple and the force is big enough, the object will slide. Objects sliding on top of a table have three degrees of freedom: linear x and y, and rotational z movements (Fig. 4.5). When we consider the complete contact surfaces to explain object motion, the assumption of using a single contact point in the center of pressure (COP) to model its behavior is not good enough to explain motion. One can notice that the object could additionally experiment rotational friction. For example, if we push an object in the x direction the object may, rotate, slide in x and slide in y directions all at once. With just a single point model, the expected movement will be only in x. This is a very inadequate behavior description. Therefore, we will consider the whole frictional surface to explain planar sliding motion.

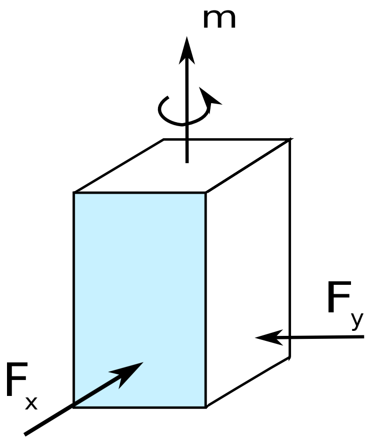


FIGURE 4.5. Object degrees of freedom and possible forces/torques.

The planar sliding problem consists on explaining motion behavior for objects with contact surfaces that experience friction. Friction was originally studied by [Coulomb 1785], [Leonardo da Vinci] and [Amontons]. Coulomb friction law can only explain motion behavior for objects that don't rotate. [Goyal, Ruina, and Papadopoulos 1991] expands this friction law to cover the case of friction surfaces.

For objects that rotate, the friction load consists not only of linear frictional forces but also of a friction in rotation, i.e, the complete friction force vector (friction load) consists not only on the x and y force components of friction (parallel components to the friction surface) but also of a torque on the z axis (Fig. 4.5).

The method described by [Goyal, Ruina, and Papadopoulos 1991] calculates this total friction vector by integration (calculus) of the friction forces over the region of contact for each axis (figure 4.6). The result is three equations with double integrals (equations 4.2). This integration is repeated for all possible center of rotation (COR) and produces a 3D representation called limit surface (LS) (figure 4.7). During slip, the *friction load* (x, y forces and z torque) is on this surface, and the direction of motion corresponds to the normal of this surface for this particular friction load. If the object is not sliding, the friction load is inside of the surface. The limit surface (LS) is the generalization of the coulomb friction law to surfaces (see figure 4.2).

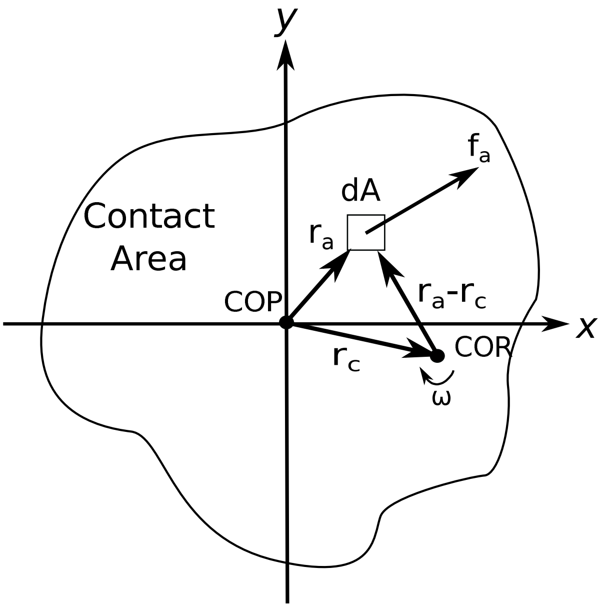


FIGURE 4.6. Infinitesimal diagram for calculating total friction load.

The formulas for sampling the limit surface (LS) are:

$$F_x = \int_A f_{ax}\, da \tag{4.2a}$$

$$F_y = \int_A f_{ay}\, da \tag{4.2b}$$

$$M = \int_A (r_{ax} f_{ay} - r_{ay} f_{ax})\, da \tag{4.2c}$$

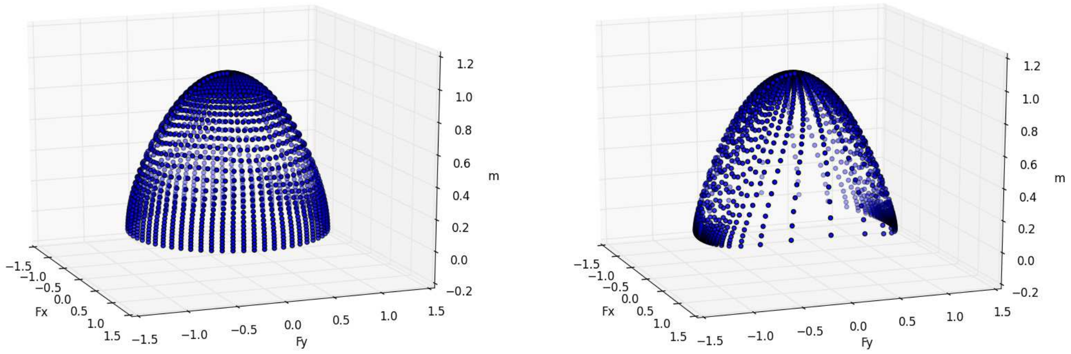


FIGURE 4.7. Limit Surface for a box. Left: Dimensions 1x1, Right: Dimensions 4x1

For constructing a limit surface (LS) equations 4.2 must be evaluated for multiple centers of rotation. If we sample more centers of rotation, our limit surface (LS) has more resolution, and therefore, better motion description.

The limit surface (LS) gives a graphical relation between forces and direction of speed with respect to a particular point (typically the center of pressure (COP)). (It doesn't explain anything about the magnitude of the speed)

For a robotic setup like ours, where we can only push on the sides of the box we need a model that is able to relate:

- Force vector in contact point vs velocity on center of mass (COM).
- Velocity of contact point vs velocity of center of mass (COM).
- Velocity of contact point vs velocity of pushing finger.

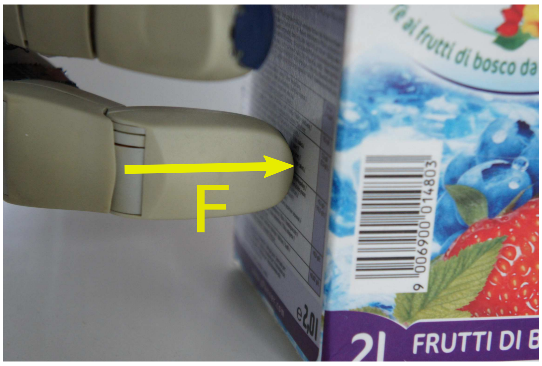


FIGURE 4.8. Finger tip force.

The first one is necessary for prediction; we use the force sensing on the finger tip to evaluate and predict the velocity of the objects in the center of mass (Fig. 4.8).

The second and the third ones are necessary for object control; we push the object with a finger tip using a velocity controller. The contact point can be anywhere in the object and can even change during motion. Figure 4.9 shows the finger tip sliding during a pushing motion.

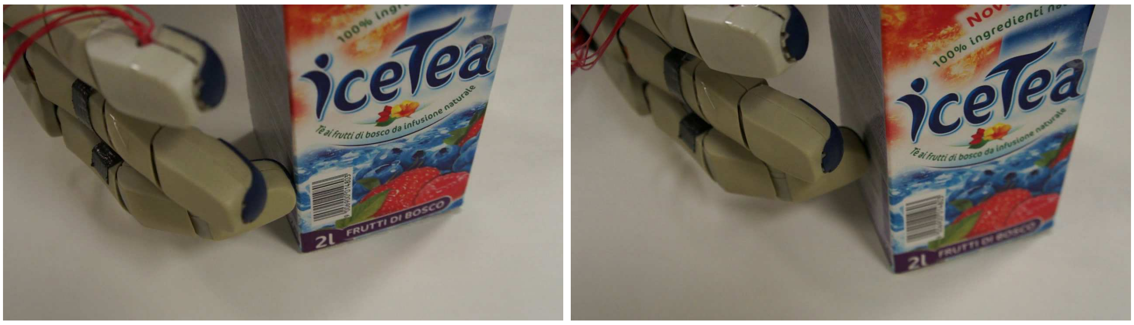


FIGURE 4.9. Finger tip sliding during pushing.

The methodology used to derive these additional relations consists on:

- Finding an analytical representation for the limit surface (LS) and the normal vector to the limit surface (LS).
- Translating the limit surface (LS) from the center of pressure (COP) to the pushing point of contact (PPC).
- Relating the force vector in the pushing point of contact (PPC) with the velocity of the pushing point of contact (PPC), by calculating the normal from the limit surface (LS) of the previous step.
- Relate velocity of pushing point of contact (PPC) with the velocity of the center of pressure (COP) (kinematic model).
- Calculate the velocity of the finger tip vs. velocity of center of pressure (COP) using a single point coulomb friction model for the friction contact between the pushing finger and the object.

We use this methodology, to derive our planar sliding model for a box (section 4.2.2).

4.2 Box Model

The Object Model System (OMS) of our box pushing example is shown in figure 4.10. Such system is based on the MOSAIC model described by [Haruno, Wolpert, and Kawato 2001] and modified to work for a model-based object description and to connect with a high level reasoning and planning system.

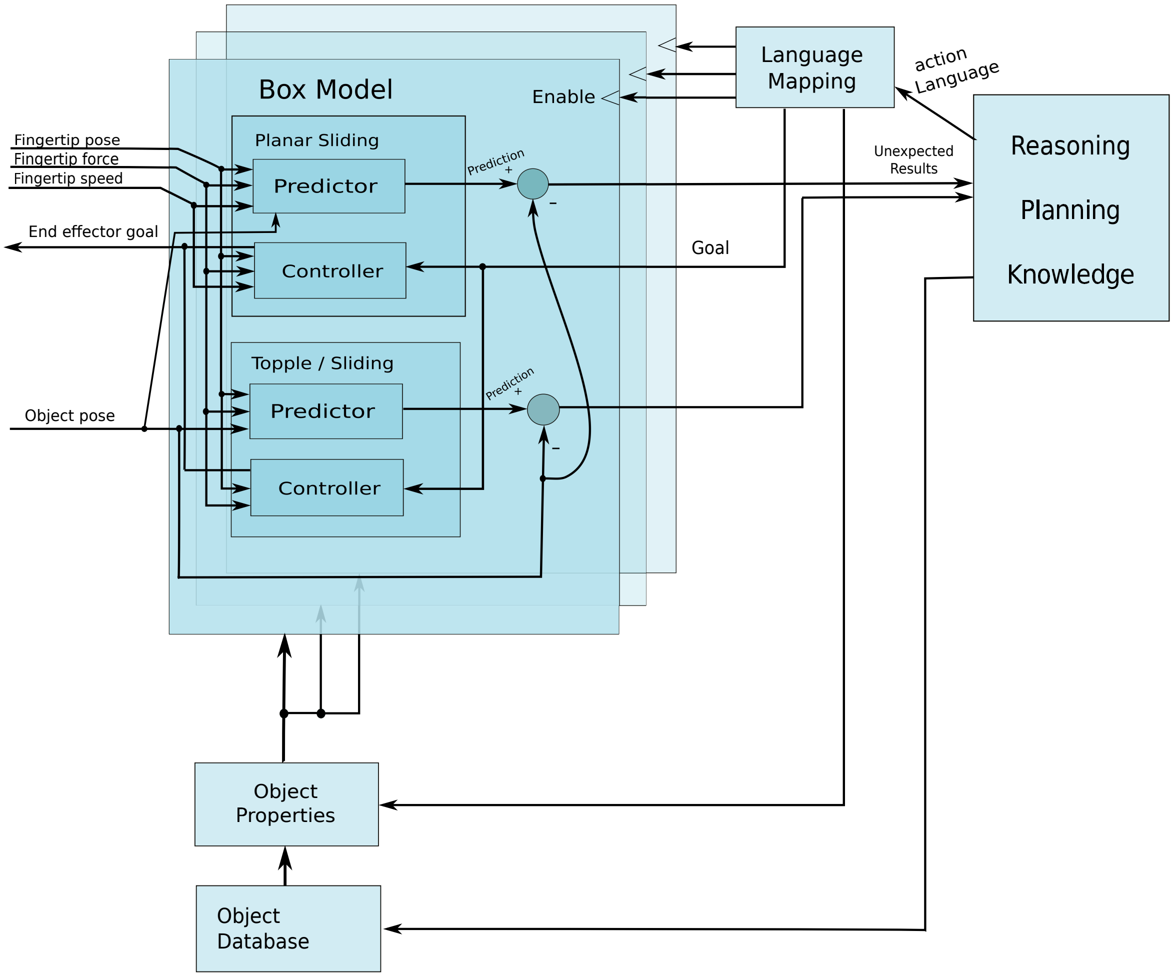


FIGURE 4.10. Object model system for our box pushing example.

Together with the software architecture presented in figure A.7 in section A.2 our complete system is fully specified.

In the following subsections we will explain in detail the submodules of our box pushing Object Model System (OMS).

4.2.1 Toppling and Sliding Model

This model is designed to predict if a box will topple or slide. To illustrate, we will use our most popular instance: an ice tea box shown in figure 4.11.

For simplification, we make to following assumptions: The box is always in contact with another object with some associated friction coefficient (box-table) and with the robot hand. The object is not deformable or if it is, the deformation is negligible. We are constraining the robot to *pushing actions*, but some other actions may be possible with the current model. We assume the object is not moving (static case), we can predict if the box will slide or rotate and how strong depending on the applied force, but this prediction is only the instantaneous value, i.e. before any movement, this is fine for slow movements where inertia doesn't play such an important role. The box model for toppling/sliding consists of three parts: friction, contact and static equilibrium. In this model, a coulomb model will be used to describe friction; we assume there will be no rotational friction (this is covered during sliding motion in our planar sliding model).

4.2.1.1 Friction

Between the object and the supporting object (table) we apply the coulomb friction model (Eq. 4.3). The relevant parameter for friction is the static friction coefficient between the object and the supporting object. Fig. 4.11 shows the interacting forces during friction.

$$F_f \leq \mu F_n \tag{4.3}$$

For our box, F_n is the combined force (external force plus weight) projected on the normal vector of the table plane and μ is found out by robot exploration. F_f lies always in the contact plane and with direction against the applied force. We are considering only the static case (no sliding yet), therefore we use the static friction coefficient for the calculations (μ).

4.2.1.2 Contact

If the object rotates (topples) we need to find out the axis of rotation of this movement. The important parameter here is the contact surface shape.

The axis where the object will rotate depends on the shape of the supporting base of the object and on the applied torques. What needs to be found is in which vertices of the base of the object there can be a rotation which involves no vertical movement (because the movement would be in the direction to the table and not away from the table).

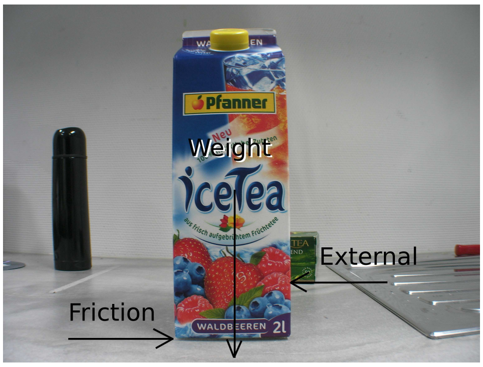


FIGURE 4.11. Friction forces.

For each vertex of the supporting base of the object a range of rotations, that will produce vertical translation, is computed using the other vertices. If a torque that is applied around the current vertex, produces a counteracting force from another point (which is part of the supporting base of the object), because this point is in contact with the table, then it is said that this torque will not produce a vertical movement on this vertex, and then we can add this torque to the set of torques that do not produce rotation with vertical movement. There is a continuous range of torques because there is a continuous set of points along the object that can cause counteracting forces. To find this continuous range, the normalized torques produced by arbitrary forces at every other vertex of the base of the object is calculated, then the biggest angle smaller than 180° between all the possible combinations of pair of angles gives the required range. Then this range is calculated for every vertex of the supporting base plane of the object and stored as part of the object model. This algorithm is shown in pseudocode 4.1 (see figure 4.12). Chapter 27 of [Siciliano and Khatib 2008] contains more elaborate information on contact forces.

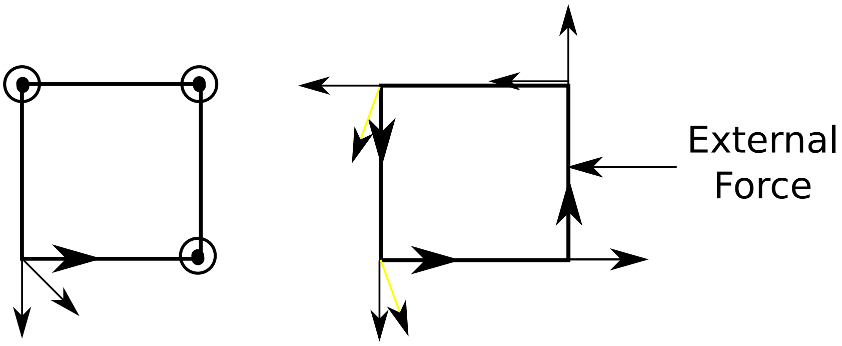


FIGURE 4.12. Rotation axis. Left: Calculating ranges. Right: Evaluating an example.

```
torque_ranges=[]
for pivot_vertex in box_vertices:
  torques=[]
  for vertex!=pivot_vertex in box_vertices:
    torque=vertex_to_pivot_vertex_torque(vertex,pivot_vertex)
    torques.append(torque)
  range=max_angle_smaller_than_180(torques)
  torque_ranges.append(range)
return(torque_ranges)
```

ALGORITHM 4.1 Calculation of range of torques for each vertex of the base of the box that produce rotations with vertical translation.

4.2.1.3 Static Equilibrium

Equations 4.4 states that all forces and torques should sum up to zero. This helps to determine if the object will slide and/or rotate depending on the applied forces, the friction model and the contact model. The relevant parameter here is the weight of the object.

$$\begin{aligned} \sum \overline{F} &= \overline{0} \\ \sum \overline{p} \times \overline{F} &= \overline{0} \end{aligned} \tag{4.4}$$

Fig. 4.13 shows the interacting forces, contact points and possible axis of rotation in the simplified 2D case (implementation is in 3D), A and B are the contact points of the box with the supporting object (table). A or/and B may be a possible axis of rotation.

4.2.2 Planar Sliding Model

Using the planar sliding model from [Goyal, Ruina, and Papadopoulos 1989] one can construct a limit surface (LS) with respect to the center of pressure (COP). This surface relates forces/torques $\boldsymbol{m}$ (from now on we will refer to this simply as forces) to a velocity direction $\boldsymbol{n}$. A typical limit surface (LS) is shown in blue in Figure 4.14.

The following assumptions will be made for this model:

1. Friction surfaces have a uniform friction (same friction coefficient for all points of contact on the friction surface) for the object-table contact, as well as, for the finger-object contact.

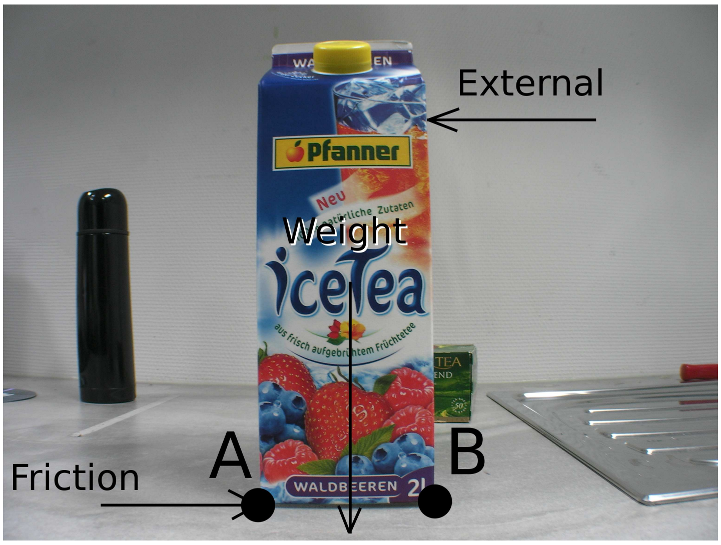


FIGURE 4.13. Contact Model.

2. The finger tip is assumed to be a point (the robot continuously corrects the finger tip orientation to maintain a fixed pose with respect the the touching surface).

3. Object-table weight distribution is uniform.

4. We use objects with box shapes and therefore the object-table surface shape is rectangular.

5. The object's centroid and center of mass (COM) are the same and therefore the object's (center of pressure (COP)) is the projection of the object's centroid in the frictional plane.

6. The finger tip can not apply torques, only forces.

As it will be shown in the last sections of this chapter and in chapter 5 the system is robust enough to give correct prediction and control results, even if our objects in reality don't follow completely these assumptions.

Using the ellipsoid approximation for the limit surface (LS) as suggested by [Howe and Cutkosky 1996]:

$$\frac{F_{xo}^2}{f_{max}^2} + \frac{F_{yo}^2}{f_{max}^2} + \frac{M_o^2}{m_{max}^2} = 1 \tag{4.5}$$

Where f_{max} and m_{max} are the maximum force (as a result from pure translation motion) and torque (as a result of pure rotation motion) respectively and $\boldsymbol{F}_{oxy}$, M_o are the force and torque applied on the object's center of mass (COM).

To fit the ellipsoid with the limit surface (LS) of our current object we need to find the maximum planar friction force f_{max}, and the maximum rotation friction torque m_{max}. It happens that for a rectangular friction surface shape, it is possible to find an analytical solution, derived from the limit surface (LS) equations when there's only translational motion (to get f_{max}) and when there's only rotational motion (to get m_{max}). The resulting formulas for f_{max} and m_{max} (using mathematical symbolic solver Maxima [Joyner 2006]) are:

$$\begin{aligned} m_{max} =& \frac{\mu\,\rho}{l_2}\left(l_2^3\,arcsinh\left(\frac{l_1}{|l_2|}\right) + l_1^3\,arcsinh\left(\frac{l_2}{l_1}\right)\right.\\ &\left. + 2\,l_1\,l_2\sqrt{l_2^2+l_1^2}\right) \qquad (4.6a)\\ \rho =& \frac{f_n}{l_1\,l_2} \qquad (4.6b)\\ f_{max} =& \mu_{ot}\,f_n \qquad (4.6c) \end{aligned}$$

Where l_1, l_2 are the rectangle dimensions of the object contact surface, f_n the normal force ($f_n = weight * g$) and μ_{ot} the object-table friction coefficient. An example of an ellipsoid (green) against a limit surface (LS) (blue) is shown in Figure 4.14.

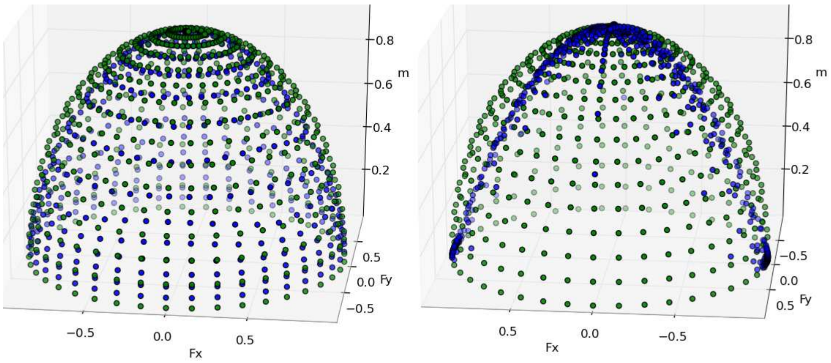


FIGURE 4.14. Limit surface. Blue: LS, green: ellipsoid approximation. For surface dimensions: Left 1x1, Right 1x10.

To get the velocity direction of the motion we calculate the normal of the ellipsoid 4.5:

$$\boldsymbol{n}_o = \nabla f(F_{xo}, F_{yo}, M_o) = \begin{bmatrix} \frac{2\,F_{xo}}{f_{max}^2} & \frac{2\,F_{yo}}{f_{max}^2} & \frac{2\,M_o}{m_{max}^2} \end{bmatrix} \qquad (4.7)$$

$\boldsymbol{n}_o$ points in the direction of the movement. In matrix form:

$$\boldsymbol{n}_o = \boldsymbol{A}\,\boldsymbol{F}_o,\; A = \begin{bmatrix} \frac{2}{f_{max}^2} & 0 & 0 \\ 0 & \frac{2}{f_{max}^2} & 0 \\ 0 & 0 & \frac{2}{m_{max}^2} \end{bmatrix} \tag{4.8a}$$

$$\hat{v}_o = \frac{n_o}{||n_o||} \tag{4.8b}$$

Since we can't control directly forces in the center of mass (COM) because we are pushing the object in its exterior surfaces, what's more useful is to derive a kinematic model that relates the motion of the object's center of mass (COM) with the motion of the pushing finger tip. The wrench relating the forces in the pushing point of contact (PPC) r_c and the center of mass (COM) is:

$$\tau_c = \tau_o - \boldsymbol{r}_{cxy} \times \boldsymbol{F}_{cxy} \tag{4.9a}$$

$$F_o = F_c \tag{4.9b}$$

$\boldsymbol{F}_{cxy}$ is the force vector in xy coordinates applied on pushing point of contact (PPC) by the finger, τ's are the torques in $\boldsymbol{r}_{cxy}$ and center of mass (COM). But since the finger tip can not apply torques:

$$\tau_c = 0,\; \tau_o = \boldsymbol{r}_{cxy} \times \boldsymbol{F}_{cxy} \tag{4.10}$$

In matrix notation:

$$\boldsymbol{F}_o = \boldsymbol{C}\,\boldsymbol{F}_{cxy},\; \boldsymbol{C} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -r_y & r_x \end{bmatrix} \tag{4.11a}$$

And since the object is sliding in the z plane:

$$\tau_o = M_o = r_x\,F_y - r_y\,F_x \tag{4.12}$$

If we combine this result with eq. (4.5):

$$M_o = m_{max}\sqrt{1-\left(\frac{F_{ox}^2}{f_{max}^2}+\frac{F_{oy}^2}{f_{max}^2}\right)} = r_x F_{cy} - r_y F_{cx} \tag{4.13}$$

But since (4.9b), grouping F terms:

$$m_{max}^2 = \left(\frac{m_{max}^2}{f_{max}^2}+r_y^2\right)F_x^2 + \left(\frac{m_{max}^2}{f_{max}^2}+r_x^2\right)F_y^2 - 2\,r_x r_y F_x F_y \tag{4.14}$$

This formula gives an analytical description of the limit surface (LS) translated to a particular pushing point ($\boldsymbol{r}$). We effectively moved the limit surface (LS) from the center of mass (COM) to the pushing point of contact (PPC).

To find the relationship between pushing point of contact (PPC) force $\boldsymbol{F}_{cxy}$ and its velocity direction $\boldsymbol{n}_{cxy}$ we calculate the normal of this limit surface (LS) with equation 4.14 to obtain:

$$n_c = \begin{bmatrix} 2(\frac{m_{max}^2}{f_{max}^2}+r_y^2)F_x - 2\,r_x r_y F_y \\ -2\,r_x r_y F_x + 2(\frac{m_{max}^2}{f_{max}^2}+r_x^2)F_y \end{bmatrix} \tag{4.15}$$

Or in matrix notation:

$$\boldsymbol{n}_{cxy} = \boldsymbol{B}\,\boldsymbol{F}_{cxy},\ \boldsymbol{B} = \begin{bmatrix} 2(\frac{m_{max}^2}{f_{max}^2}+r_y^2) & -2\,r_x r_y \\ -2\,r_x r_y & 2(\frac{m_{max}^2}{f_{max}^2}+r_x^2) \end{bmatrix} \tag{4.16a}$$

$$\hat{v}_c = \frac{\boldsymbol{n}_{cxy}}{||\boldsymbol{n}_{cxy}||} \tag{4.16b}$$

If we put (4.8), (4.16) and (4.11) together we get:

$$\boldsymbol{n_o} = \boldsymbol{A}\,\boldsymbol{C}\,\boldsymbol{B}^{-1}\,\boldsymbol{n}_{cxy} \tag{4.17a}$$

$$\hat{v}_o = \frac{\boldsymbol{n}_o}{||\boldsymbol{n}_o||} \tag{4.17b}$$

This only gives the velocity direction but not the magnitude. The twist between center of mass (COM) and pushing point of contact (PPC) is:

$$\boldsymbol{v}_{cxy} = \boldsymbol{v}_o + \boldsymbol{r}_{cxy} \times \boldsymbol{w} \tag{4.18}$$

Where $\boldsymbol{w}$ is the rotation velocity vector (equal in center of mass (COM) and $\boldsymbol{r}_{cxy}$). In matrix notation:

$$\boldsymbol{v}_{cxy} = \boldsymbol{D}\,\boldsymbol{v}_o,\ \boldsymbol{D} = \begin{bmatrix} 1 & 0 & -r_y \\ 0 & 1 & r_x \end{bmatrix} \tag{4.19a}$$

Now we can find the scaling factor to get a velocity magnitude using (4.19) and (4.17):

$$v_o = \alpha_1 * n_o \tag{4.20a}$$
$$v_c = \alpha_2 * n_c \tag{4.20b}$$
$$v_o = \frac{\alpha_1\, A\, C\, B^{-1} v_c}{\alpha_2} \tag{4.20c}$$
$$v_c = \frac{D\, \alpha_1\, A\, C,\, B^{-1} v_c}{\alpha_2} \tag{4.20d}$$
$$\alpha = \frac{\alpha_2}{\alpha_1} = \frac{||D\, A\, C\, B^{-1} v_c||}{||v_c||} \tag{4.20e}$$

We simplified the right hand side (using Maxima) and the result is:

$$\alpha = \frac{1}{m_{max}^2} \tag{4.21}$$

Then the kinematic model becomes:

$$\boldsymbol{v}_o = \boldsymbol{M}\,\boldsymbol{v}_{cxy},\ \boldsymbol{M} = m_{max}^2 \boldsymbol{A}\,\boldsymbol{C}\,\boldsymbol{B}^{-1} \tag{4.22a}$$

Which is a linear model if $\boldsymbol{r}_{cxy}$ doesn't change. The reference frame of this model is the object's center of mass (COM). To get this model in table's reference frame, we rotate the resulting velocity by the current object angle (ϕ):

$$\boldsymbol{v}_{o_{table}} = \boldsymbol{R}\,\boldsymbol{M}\,\boldsymbol{v}_{cxy},\ \boldsymbol{R} = \begin{bmatrix} cos\phi & -sin\phi & 0 \\ sin\phi & cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.23a}$$

This model relates the speed of the pushing point of contact (PPC) to the object's center of mass (COM) velocity. It is non-linear and since $\boldsymbol{R}\,\boldsymbol{M}$ is a 3x2 matrix and the system is linearly independent on the control inputs, the system is classified as a "drift-less" or non-holonomic system [Isidori 1995]. Such systems have proven to be difficult to point stabilize (car parking problem).

How can we know if, during pushing, the finger slides? To answer this question we use the friction cone of the finger-object contact. To simplify the analysis, we assume there's no finger movement in the z direction (table normal), then we can define the possible forces that can be applied by the finger as a range between two limit vectors.

Then using equation (4.16) we can calculate the range of velocities the finger can have without sliding. If the robot moves the finger outside of this range, the finger will slide. The amount of sliding can be calculated by subtracting the finger speed with the speed calculated with the limit force in (4.16).

$$\boldsymbol{v}_{finger_{xy}} - \boldsymbol{v}_{cxy_{sat}} = v_{slide_{xy}} \tag{4.24}$$

where $\boldsymbol{v}_{cxy_{sat}}$ is one extreme of the range if it slides to one side, or in the other extreme of the range if it slides to the other side.

Finally, if the finger slides the model becomes:

$$\boldsymbol{v}_{o_{table}} = \boldsymbol{R}\,\boldsymbol{M}\,\boldsymbol{v}_{cxy_{sat}} \tag{4.25}$$

Eqs. (4.25), (4.23) and (4.24) together are used to predict object behavior.

A simulation environment was programmed to test the model. A sequence of a pushing motion is shown in Fig. 4.15.

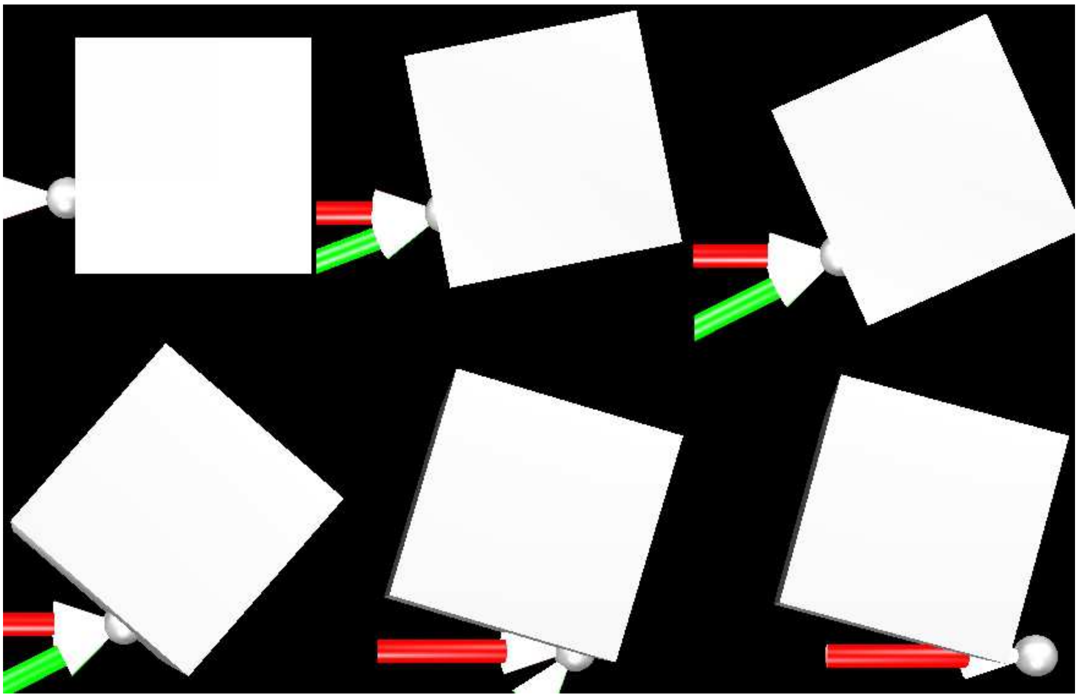


FIGURE 4.15. Sliding sequence simulation. Red: finger speed, green: finger force.

4.3 Possible Extensions to the Model

- Taking into account inertial effects can give the robot three different capabilities: 1) to predict object motion when we throw the objects. 2) the possibility to increase operating speed of the pushing action. 3) to predict free planar sliding motion.
- Modelling containers with liquid inside.
- Modelling chains of objects, i.e, objects touching other objects being pushed.
- Irregular surfaces (different slopes during motion, non-uniform friction coefficients).
- Irregular weight distribution.

Increasing complexity of the model may result as more of these effects are taken into account. Thus, it is important to use or activate only the necessary modelling sections for the particular task at hand (prediction- and control-wise).

4.4 Exploring for the Parameters

The Object Model System (OMS) requires to have the Object Model (OM) parameters determined to be able to predict and control object behavior. Because these parameters represent physical object properties, there may be several ways to obtain these parameters. A particular item may always have the same properties and the robot could fill them up just by looking into a database. Also, these parameters could be reloaded from previous use of a particular object. Unfortunately, there are many objects which properties usually change constantly or very often and measuring them before each use may become mandatory. In this case, it would be most reasonable that the robot itself could measure such parameters. This has two advantages: 1) immediate availability of the data and 2) data is "fit" to the robot sensors and therefore, its body. The second advantage is also important because it "grounds" object information representation to a robot body and to a real object. This is something that has been identified as necessary to fuse high level AI with low level motor control [Harnad 1990] [Coradeschi, Loutfi, and Wrede 2013] [Duffy and Joue 2000] [Steels 2008].

If we want the robot to find itself the Object Model (OM) parameters, the robot must do some exploration of the object, that is, the robot has to "play" with the object.

The fastest way to determine these parameters, is to have a formula that solves for a parameter directly taken from the object model, and then give the necessary inputs (for example, pushing forces) to the object and measure the corresponding outputs to calculate the parameter using such formula. One important detail is, that when we solve for this formula, we must try as much as possible not to depend on other unknown parameters. If we can solve the formula and still only depend on one unknown parameter, then the experiment will be a simplified usage of the object which shows only a part of its behavior.

It is important to find out the right order of experiments to find all the parameters one by one.

This method may have some limitations (which will be discussed in the future work), but it presents an important advantage: simplicity. It works, and it is extremely simple to execute.

In the following subsections we will show how to explore for the Object Model (OM) parameters for our "pushing a box" example. The last subsection will show the results of using this method on a real object.

4.4.1 Box Dimensions and Contact Points (Base Shape).

Instead of letting the robot explore the shape of the object (which would be a more complicated procedure out of the scope of this thesis, for an example: [Maldonado, Alvarez-Heredia, and

Beetz 2012]), the robot could get the shape of the object from a knowledge database, which is also used by the perception system [Klank et al. 2009] to detect the object and the position of it.

The center of mass (COM) is calculated from the dimensions of the box assuming that the box is resting in one of the sides and has uniform weight distribution. Since most objects will have most often a uniform horizontal weight distribution (because the contents will tend to move to the bottom of the box because of gravity), then this assumption is acceptable. It is common to find objects with non-uniform vertical weight distribution (e.g. half full liquid container), but this doesn't affect the performance of our system much because for the box object model it is more important that the horizontal component of the center of mass is as centered as possible.

The most important part of the box dimensions is the shape of the base of the box; it determines the friction forces the box experiences. The rest of the dimensions (lateral surfaces) are important because the robot needs to calculate if it is already touching the object and to calculate the effective force applied to the center of friction of the object.

4.4.2 Weight

Weight is necessary for the friction and force/torque balance part of the model. There are two ways to find the weight of the object. A practical approach is to 1) grab the object then lift it and then measure the force with a force sensing robotic hand/arm. Another way to find the weight is to 2) touch the object with the robot hand at a point that will most likely tilt the box but not slide it and then measure the necessary force to start tilting the box (detect the starting of tilting with movement detection from the hand/arm and/or with vision), then use the toppling/sliding model to calculate the weight. If there is liquid in the box, the box must be lying in one of its sides to let us use a simple calculation for the center of mass. Also, there can't be any sudden movement which would cause liquid oscillations. Method 2 is used because our robotic hand was not strong enough to lift one of our examples (the ice tea box) with 1 liter of water inside. One of the fingers of our robot is used as the force sensing device.

$$weight = \frac{2F_{ext}h}{lg} \tag{4.26}$$

Supposing the robot is pushing a box through a normal table vector that passes through the center of mass (COM), equation 4.26 calculates the weight of the object if the object is being slightly tilted. F_{ext} is the force applied by the robot while tilting the object, h is the height of the pushing point, l is the length of the side of the object and g is the gravity of the earth. This formula is the result of applying the equilibrium equations of the topple/slide model (equations 4.4).

4.4.3 Friction Coefficient.

From equation 4.3 one can easily solve for the friction coefficient:

$$\mu = \frac{F_{ext}}{F_n} \tag{4.27}$$

Once the weight is known we can find out the static friction coefficient by applying an increasing force until the object starts sliding. The value, just before it starts sliding, is the maximum static friction force. In order to detect this precisely we can use the hand/arm to measure when the object starts to move. It is important that the object has low deformation to forces and that no tilting occurs. To assure that the object will only slide and not tilt, the robot must push the object on a very low point of the object height. Also, the object must be pushed in such a way that no sliding rotation happens.

4.4.4 Experiments

In this section we explain how the robot explores for the parameters with our robot TUM-Rosie.

Weight

For estimating the weight we tilt the object to a particular angle (small but that guarantees that the bottom surface is not touching the table). With our objects ≈ 5 degrees of inclination was enough. The robot pushes the object until the angle is reached, then several force measurements are taken to calculate a mean force. The tilting angle is taken into consideration to calculate the corresponding torques generated on the rotation axis.

Friction coefficient

For the topple/sliding model we need to find one friction coefficient: the static object-table friction coefficient. For the planar sliding model we need to find four friction coefficients: the static and kinetic friction coefficients for the finger-object contact and for the object-table contact.

For the object-table friction coefficients the method consists on:

1. Approach with the finger a lateral object surface (at a low object height) at a constant speed. Meanwhile, constantly monitor the force sensed on the finger tip. The force profile should look similar to the one shown in figure 4.3(green).

2. Store the highest force value. Use it to calculate the static friction coefficient with equation 4.27.

3. Keep pushing, and wait some time (1 second to let the friction force stabilize), then measure again, but this time take several measurements and calculate the mean. Use this to calculate the kinetic friction coefficient again using equation 4.27.

4. Stop movement and move the finger away.

For the finger-object friction coefficients, we will help the robot a bit by putting the object with the lateral surface (the surface we are interested to find the friction coefficient) looking up. Then proceed as follows:

1. Approach with the finger from above the lateral surface of interest. Continue until the finger touches the object with a certain force and then stop the movement. The finger will stop but will keep applying this force against the surface of the object. This force will be the normal force in the coulomb friction model.

2. Move the finger horizontally (parallel to the surface) at a constant speed. Meanwhile, constantly monitor the force sensed on the finger tip. The force profile should look similar to the one shown in figure 4.3(green).

3. Store the highest force value. Use it to calculate the static friction coefficient with equation 4.27.

4. Keep the finger in motion, and wait some time (1 second to let the friction force stabilize), then measure again, but this time take several measures and calculate the mean. Use this to calculate the kinetic friction coefficient again using equation 4.27.

5. Stop movement and move the finger away.

The object-table friction coefficients are calculated using the friction forces and the weight. The finger-object friction coefficients are calculated using the friction forces and the normal force applied by the finger. See Fig. 4.16.

We executed two different experiments, one for the toppling/sliding model and another for the planar sliding model.

For the toppling/sliding model we use an ice tea box filled with approximately one liter of water. The results for this experiment are shown in table 4.1 (for 10 repetitions).

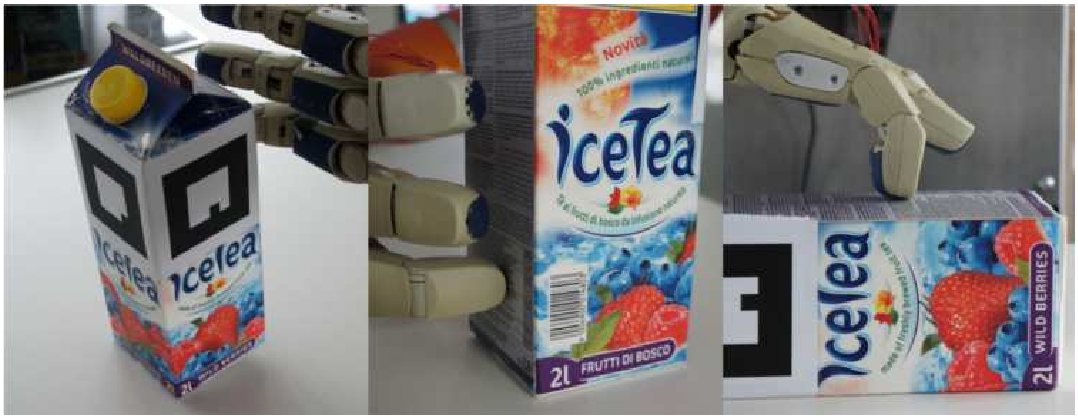


FIGURE 4.16. Robot finding the model parameters

	Weight	μ_s
Mean	0.9952 kg	0.3780
Std. Dev.	0.0375 $kg^{1/2}$	0.0689
Std. Dev. %	3.7721%	18.219%
Min	0.9367 kg	0.3127
Max	1.0583 kg	0.5533

TABLE 4.1. Properties of Ice tea box used for the toppling/sliding model.

The estimated weight value matches very well the filled water content weight. The standard deviation is low, which indicates that the procedure to find this parameter is very stable. The same can not be said about the friction coefficient. It shows significant variability. The reason for this may be, because of not modeled complicated behavior of the friction surface (deformation), also the surface of the table is not completely regular and it may have regions with different friction coefficients. For this experiment we use a wooden table.

For the planar sliding model we used three different objects with different dimensions: an ice tea box, a coffee package, and a cereal box. For each, the experiments were repeated 10 times, and then the mean of the values calculated. The resulting parameters are shown in table 4.2.

These results will be used together with the box object model to predict and control. This will be discussed in the following chapters.

	Ice tea	Coffee	Cereal
weight (kg)	0.419	0.510	0.241
$\mu_{s_{object-table}}$	0.298	0.313	0.343
$\mu_{k_{object-table}}$	0.269	0.240	0.292
$\mu_{s_{finger-object}}$	0.693	0.885	0.938
$\mu_{k_{finger-object}}$	0.567	0.733	0.660

TABLE 4.2. Objects properties for the planar sliding experiments.

4.5 Action Prediction

Up to this point we have developed an Object Model (OM) and a method to obtain its parameters. In the rest of this chapter we will show how to use this model to predict object behavior. We will first look at the toppling and sliding and then at the planar sliding model. A simulator was developed to evaluate the prediction of the Object Model (OM) before testing it on the real robot. Also, test runs on a real robot are shown and discussed.

4.5.1 Toppling and Sliding

Using equation 4.3 and the force balance part of equation 4.4, the robot can determine if the box will slide or not. If the combined force (external force and weight) projected on the table plane, is bigger than the maximum friction force $\mu_s F_n$, then the object will slide. On the other hand, if this projected force is smaller than the maximum friction force, then the object will not slide.

Apart from sliding, the object can also topple (or rotate in the vertical plane). This rotation can be 1) around an axis defined by two of the four box base vertices of the support polygon, 2) around an axis defined by a single vertex from the base and the produced total torque (external torque plus weight induced torque) around this point or 3) no rotation at all.

To find this out, first the robot must compute the total applied torque (using the torque part of equation 4.4) projected on the table plane on each of the supporting vertices, then it must compare these torques against the torque ranges calculated using algorithm 4.1. If for one of the vertices, the applied torque lies outside of the torque range of such vertex, then this vertex will translate vertically. For the vertices where the applied torque lies inside of the torque range, then these vertices will define the axis of rotation on the table where the object will rotate. If only one vertex has the applied torque inside, then that torque is going to define the rotation axis alone. Rotation will not occur if the applied torque lie outside of the torque range for all the vertices.

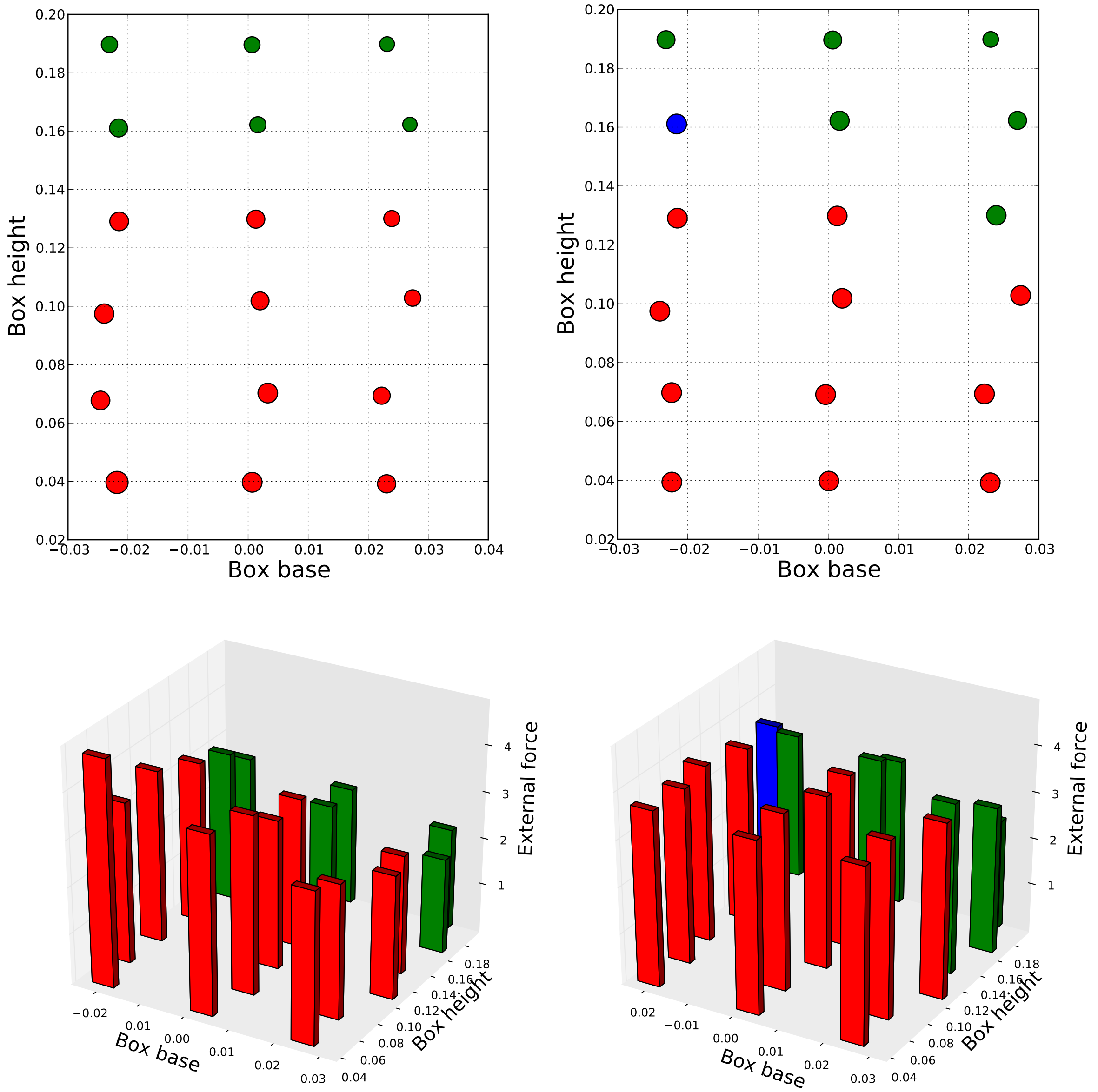


FIGURE 4.17. Points of contact where the box slides or rotates if a force high enough is applied. Right: prediction model, left: experimentation. In the 2D pictures, the circle area indicate the force magnitude, while in the 3D pictures this force is indicated by the third coordinate. Red, green and blue indicate slide, rotation and sliding/rotation respectively.

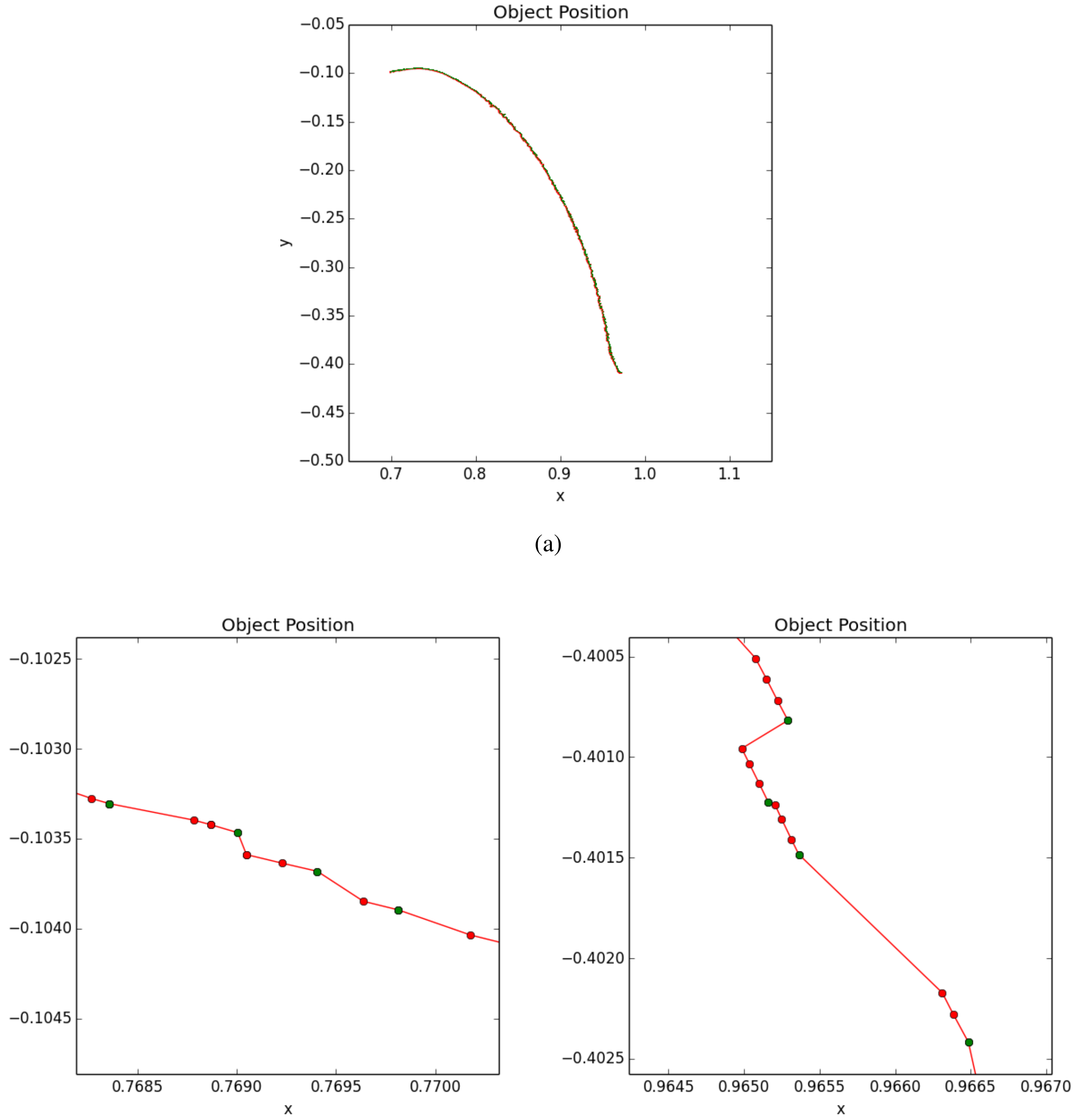
A comparison between the model predictions and actual action execution is shown in figure 4.17. The graphs show in colors the different outcomes when the box is pushed hard enough to produce a movement. As the figures show the object model is very good in predicting correct outcomes, so it's easy for the robot to determine which regions are highly possible to get the desired outcome, but it's not so accurate about the minimum forces needed in order to slide it. Most probably this is related to the fact that the static friction coefficient showed great variability, possibly because of not modeled complicated phenomena. Nevertheless, this prediction is still useful if the robot wants to avoid to use values near the threshold zone that divides sliding and not sliding, which for most of the applications it is the case.

4.5.2 Planar sliding

The planar sliding prediction will consist of calculating the next (in the next control cycle) object position according to current pushing sensor information (finger force, and finger velocity) and current object position. The current position is measured with a camera at 15 fps, and the pushing sensor information is taken at 60 to 120 samples per second (depending on the computer processing load).

Initially, the predicted object position is set to the camera object estimated position (see green circles in figure 4.18(b)). Because the camera position measurements are slower than the finger measurements (15 fps against 60 to 120 samples per second), at the second finger measurement (and the ones after that until a new camera measurement comes) the predictor only uses the finger tip measurements (force and position) to estimate object positions while there are no camera estimates available (See red circles in figure 4.18(b)). In this way, the predictor it's believing the camera completely about the global position of the object when measurements are available, but when no camera estimate is available the predictor uses the finger forces to integrate object positions. This can be better explained with figures 4.18(b) and 4.18(c) (the movement of the object goes from down-up). For example, in figure 4.18(c) at position x=0.9655 y=-0.4015 a new camera object pose estimate is available (green circle), the next three red circles are predicted object positions. The last of those three red circles is near to the next green circle, indicating that the expected object pose is quite near to the reality, meaning that the predictor is modelling the object pretty well and no external events affected the object. In the same figure, other green circles are somewhat away from the last previous red circle, meaning that the prediction failed more. Although at this scale some of these prediction errors seem big, at the complete scale they are actually still very small (see figure 4.18(a)).

Internally, the system calculates the object's velocity direction using equation 4.16 and the current finger forces. The actual velocity magnitude is calculated by scaling n_c by a factor that makes the n_c component that is normal to the finger-object surface equal to the same speed of the same component of the finger tip velocity.



(a)

(b)

(c)

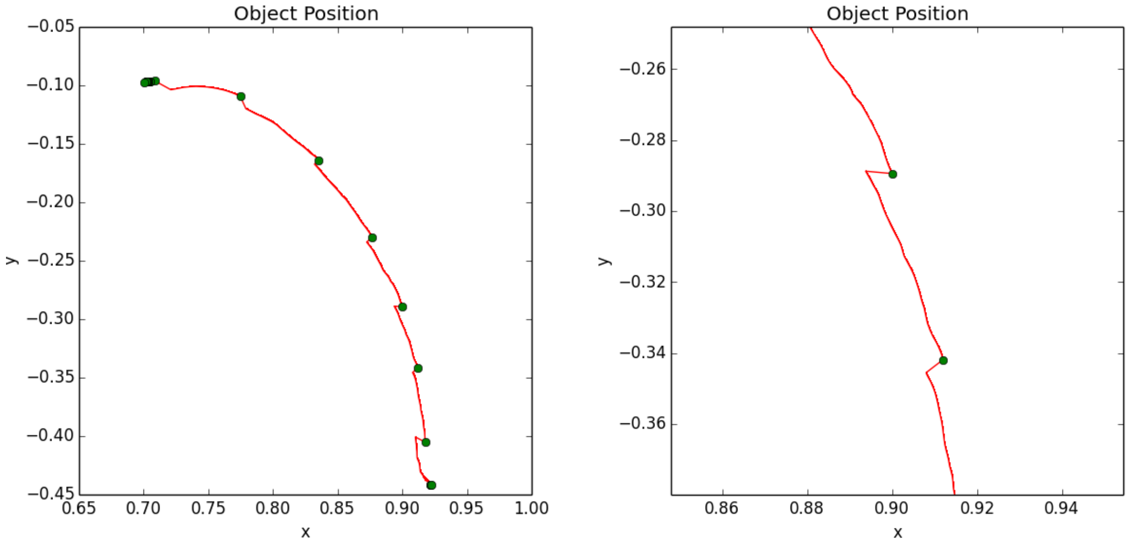
FIGURE 4.18. Prediction. Normal camera object pose update rate. (a) Complete pushing movement. (b) Close up when the object is moving mostly in the x direction. (c) Close up when the object is moving mostly in the y direction. Green: Camera object pose updates, Red: Predicted object pose based on the finger pose and finger tip force.

$$v_{finger\perp_{surface}} = \alpha\, n_{c\perp_{surface}} \tag{4.28a}$$

$$v_c = \alpha\, n_c \tag{4.28b}$$

Then equations 4.23, 4.24 and 4.25 are used to calculate the object velocity and this is then integrated to get the next object position. Additionally, the sliding velocity of the finger with respect to the object can be calculated using equation 4.24. The current finger tip velocity is obtained by derivating the finger tip position measurements.

To decide when the model should predict or not (because the finger may or may not be currently pushing the object), we use a finger force threshold based on equation 4.14, and also detect how near to the object lateral surfaces the finger is (using a plücker polygon test [Shevtsov, Soupikov, and Kapustin 2007] [Yamaguchi and Niizeki 1997]).



(a) (b)

FIGURE 4.19. Prediction. A forced very slow camera object pose update rate. (a) Complete pushing movement. (b) Close up. Green: Camera object pose updates, Red: Predicted object pose based on the finger pose and finger tip force.

Figure 4.19 shows in red the object predicted pose and in green the camera object pose updates. In this figure we artificially forced a slower camera update rate to show more clearly the effect of a longer object prediction. The "jumps" in the pose indicate that the new camera-based object pose updates differ significantly from the last expected object pose. This is expected since we artificially force a more delayed camera object pose update, therefore increasing the time the predictor works, thus accumulating more prediction errors; the prediction estimates start to drift away if we don't get global synchronizations (camera object poses) often enough. Real camera update rates are shown in figure 4.18.

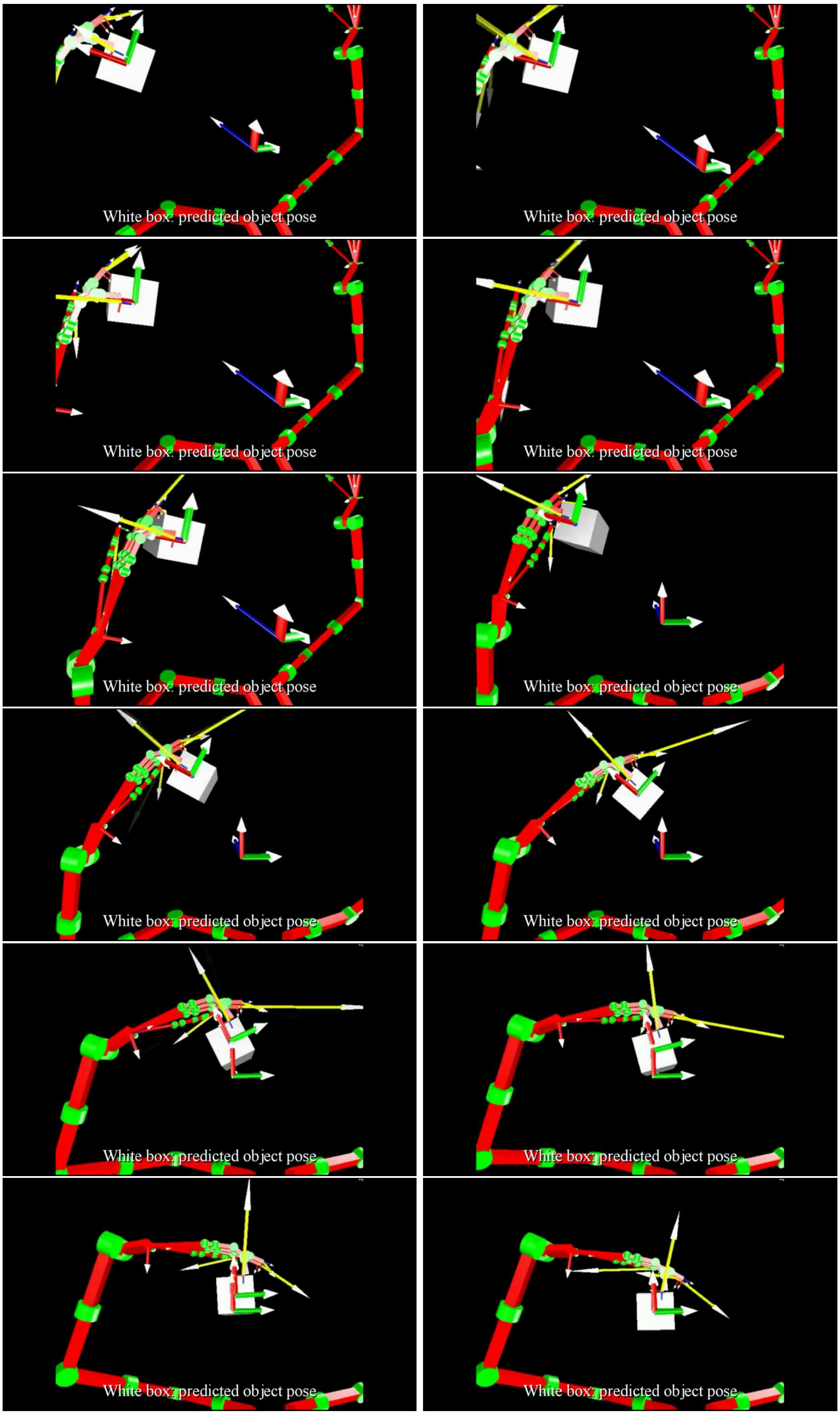


FIGURE 4.20. Visualization of a pushing sequence. White box: predicted object pose and camera measured object pose mixed. Yellow arrows: Current finger tip measured force.

During some of the experiments, we noticed prediction accuracy was specially sensitive to offset (calibration) errors in the camera estimated poses. This is expected since the planar sliding motion is very sensitive around pushing directions passing through the center of pressure (COP). A pushing vector that is passing slightly away of the center of pressure (COP) causes a strong turning (rotation) reaction. We are considering to extend our system to explore the center of pressure (COP) offset and automatically calibrate for it. This can be done by trying to push in different positions looking for a position that minimizes object rotation.

Figure 4.20 shows a pushing sequence in our visualizer (pyrovito). The white box represents the ice tea box. The current pose of the white box corresponds either to the camera measured object pose or to the current predicted object pose. This sequence is the result of running the planar sliding controller explained in the next chapter. The resulting goal precision is clearly visible.

4.6 Discussion

This chapter presents a test case example of an Object Model (OM): *pushing a box*. In this chapter only prediction is considered, controlling the object is covered in chapter 5.

Two types of motions are analyzed: toppling and sliding. The robot can predict if an object will topple or slide and it will also know the axis of rotation in case of toppling. If sliding occurs the robot will predict the sliding motion by calculating the instantaneous velocity vector of the object. Future object position can be estimated by integrating this velocity. During sliding motion, two important situations are considered: the object-table contact surface will slide and experience friction and also the finger-object contact surface could slide and experience friction. Considering a sliding motion of the finger-object surface generalizes our model, improving prediction, but also turns the dynamical system into a *time-variant* type, complicating control analysis and design.

To obtain the Object Model (OM) parameters a direct parameter exploration is used for the weight and the friction coefficients. Because the robot is doing the exploration itself, the parameters are *adjusted* to that robot body and therefore *grounded*. The object dimensions are simply extracted from a database (given previously). A series of experiments were performed on several objects and the resulting parameter values are used during action prediction.

Test results of object behavior prediction are shown at the end of the chapter. The experiments to predict when or where the box would topple or slide produced a correct height indicating the point where the box would topple instead of sliding. Also the magnitude of the force to slide it or topple it were similar for the prediction and what actually happened. This indicates that the toppling/sliding model works properly for prediction purposes.

The experiments for only sliding motion show excellent results, indicating where the box would be in the following instances given current force measurements and previous position estimates. To see the effect of a strongly delayed camera object position update, prediction results are shown when the camera update rate is strongly reduced. Even under these circumstances the object predicted position is quite acceptable. All this indicates that the model for predicting sliding motion is properly working with a satisfying performance.

Given the experimental results, we conclude this chapter stating that our Object Model (OM) for toppling and sliding of a box is properly predicting object behavior and therefore providing preliminary proof of the correctness of our methods for creating an Object Model (OM). Given these results, we believe that our methods have the potential to be used to create more Object Models (OMs) for prediction for other applications and/or objects.

For a discussion on possible methods to improve even more prediction, the reader is advised to look at the general conclusions in chapter 6.

CHAPTER 5

Pushing a Box (motion control)

Pushing a box is a complicated action that requires to generate very specific motor commands to achieve a successful final result: reach the desired final object position and orientation (pose) (see figure 5.1). When the robot uses only one finger to push a box, the resulting box movement is difficult to control. For example, if the robot pushes the box at the center of one side of the box, one can expect a straight-line motion, but in reality the box will start to rotate more and more. This happens because the robot never pushes the box exactly at the center, and the table/box interface surface doesn't have a perfectly uniform friction distribution. The result is that a very small deviation where the robot pushes from the center can cause a extremely different result on the box movement. This usually indicates that a feedback controller must be used in order to stabilize the system to generate the desired object movement.

From the previous chapter, one can notice that the box sliding on a table system is a non-holonomic, time-variant and non-linear dynamical system, and therefore one of the most difficult systems to stabilize. Problems that are similar (albeit a bit more simple) are for example the problem of parking a car; a non-holonomic, non-linear dynamical system (car) must be moved from one place to another. Many solutions to parking a car exist [Murray and Sastry 1991] [De Luca, Oriolo, and Samson 1998] [Murray and Sastry 1993] [Kolmanovsky and McClamroch 1995] [Wen 1996] [Matsuno and Tsurusaki 1999], but they usually require that the dynamical system is represented in a chained form. No method up to now was found to transform our dynamical system to a chained form, and therefore, these solutions could not be used directly. Also, another problem is that we are introducing a special requirement: to achieve the final goal in **one single shot**. That is, we would prefer a control system that requires the robot to push the box from one direction only, and therefore, the robot must push the box in such a way that it approaches the goal directly as best as possible and no necessary improvements to the final position are necessary that would require pushing from another side of the box. This requirement is important because our robot arm may run into a very limited space and it may be impossible to relocate the arm of the robot to push the box from another location.

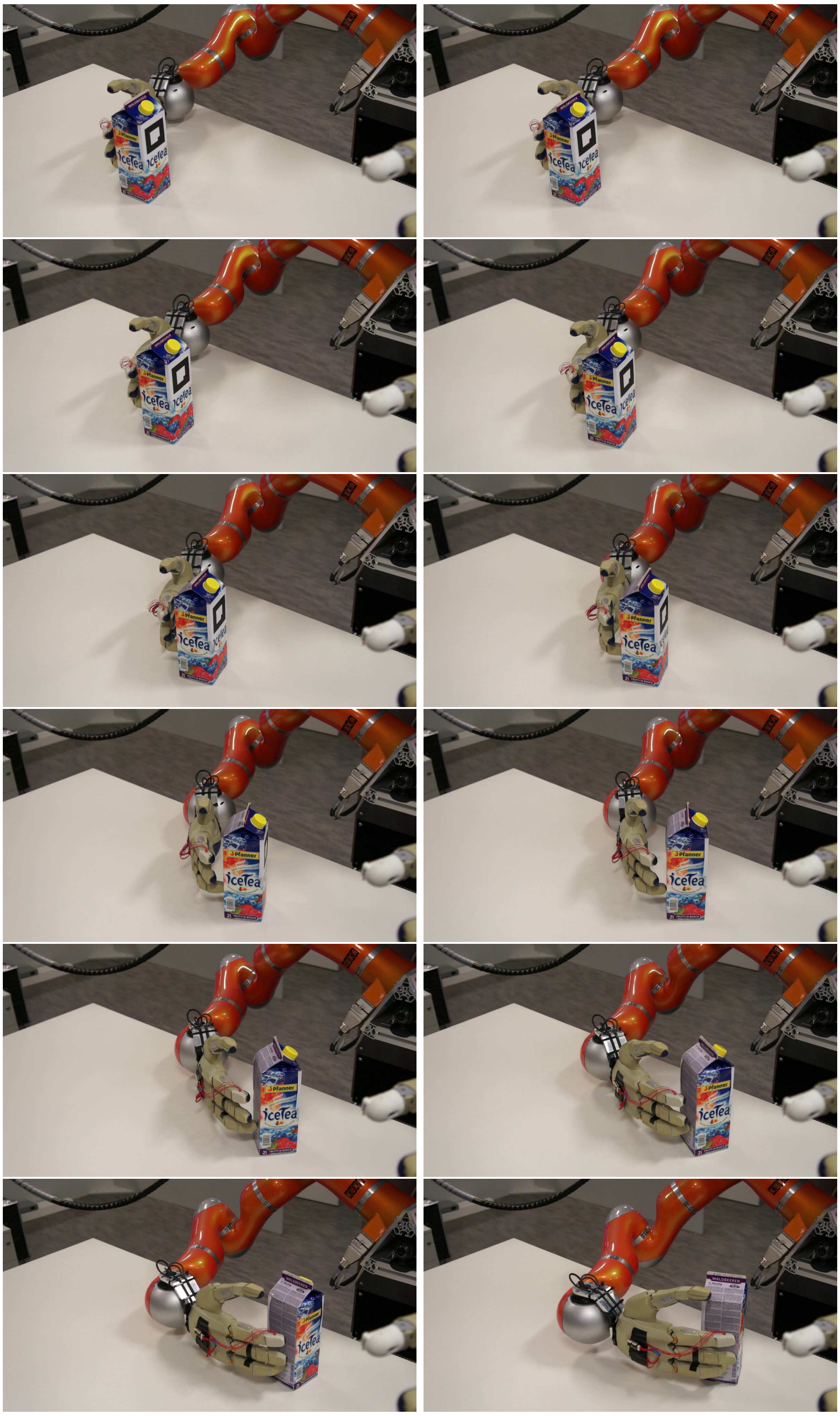


FIGURE 5.1. Pushing sequence.

In this chapter, we will first analyze the topple and slide control decision problem: where to push the object in order to topple or slide it. Then, we will explain the planar sliding control problem and how to solve it. Results from tests on our robot TUM-Rosie are shown together with the resulting performance of the system under strong perturbances.

The reader is advised to first review chapter 4 and to have some previous understanding of control theory before continuing.

5.1 Topple or Slide

The object may slide or topple depending on where (at which height) the robot pushes the object. There are objects that may never topple (when limited to push on one lateral side only) because of the shape, weight and maximum height they have.

To find out the height at which the object will topple instead of sliding we use all the equations from section 4.2.1. To reduce the space of possible solutions we also limit the movement of the robot finger to only horizontal movements (forces only in x and y coordinates: parallel to the table).

To guarantee that no sliding will happen (using equation 4.3) the applied force must be between (0.9 is a safe margin):

$$0 \leq F_{not\_slide} \leq 0.9 w g \mu_s \tag{5.1}$$

With w the weight of the object, g the earth gravity and μ_s the table-object static friction coefficient.

Sliding will happen if (1.1 is a safe margin):

$$F_{slide} \geq 1.1 w g \mu_s \tag{5.2}$$

To guarantee rotation, the applied force must be (using equation 4.4):

$$F_{rotate} > \frac{wgl}{2h} \tag{5.3}$$

Were l is the length of the side of the box parallel to the pushing direction and h is the pushing height.

And for no rotation:

$$F_{not\_rotate} \leq \frac{wgl}{2h} \tag{5.4}$$

With valid pushing heights (to avoid touching the table with the finger or missing the object at all) between (1.2 is a safety margin):

$$\frac{h_{finger}}{2} * 1.2 \leq h \leq h_{object} - \frac{h_{finger}}{2} * 1.2 \tag{5.5}$$

Where h_{object} is the height of the object and h_{finger} is the diameter of the finger along the table normal direction.

If the robot wants to only slide the object (not rotate), it first calculates $F_{slide_{min}} = 1.1wg\mu_s$ (equation 5.2), then it calculates $F_{not\_rotate_{h_{min}}}$ with equation 5.4 using the minimum pushing height ($h_{min} = \frac{h_{finger}}{2} * 1.2$). Then the pushing force must be bigger than $F_{slide_{min}}$ and smaller than $F_{not\_rotate_{h_{min}}}$, if this is not possible then only sliding (with no rotation) is not possible. If this force range exists then it selects the force in the middle ($\frac{F_{not\_rotate_{h_{min}}} - F_{slide_{min}}}{2}$) and use it in $h = \frac{wgl}{2F}$ to get the pushing height.

If the robot wants to only rotate (topple) (but not slide) the object, it first calculates $F_{not\_slide_{max}} = 0.9wg\mu_s$ (equation 5.1), then it calculates $F_{rotate_{h_{max}}}$ with equation 5.3 using the maximum possible pushing height ($h_{max} = h_{object} - \frac{h_{finger}}{2} * 1.2$). Then the pushing force must be smaller than $F_{not\_slide_{max}}$ and bigger than $F_{rotate_{h_{max}}}$, if this is not possible then only rotation (with no sliding) is not possible. If this force range exists then it selects the force in the middle ($\frac{F_{not\_rotate_{h_{max}}} - F_{not\_slide_{max}}}{2}$) and use it in $h = \frac{wgl}{2F}$ to get the pushing height.

In figure 5.2 first are shown the situations where only sliding is possible because the object is tall enough, and second, where we want to only slide the object but $F_{not\_rotate_{h_{min}}}$ is smaller than $F_{slide_{min}}$ because the object is too small, therefore sliding without toppling is not possible.

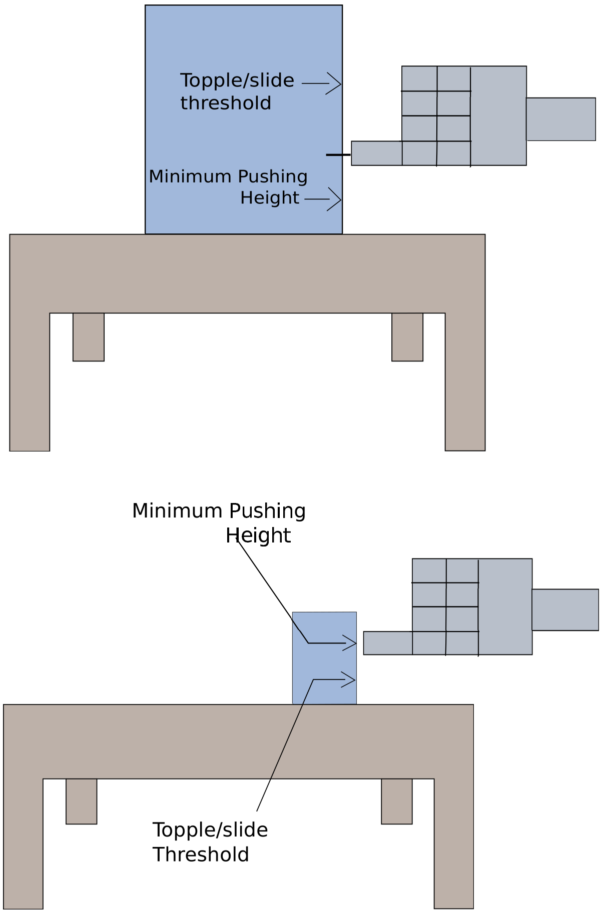


FIGURE 5.2. Pushing height. Above: Threshold high enough to be able to slide correctly. Below: Threshold smaller than minimum pushing height, sliding not possible.

No experimental data has been taken with respect to toppling/sliding control. Although we may run some experiments in this respect in the future, the equations presented in this section are straight forward and expected to work specially if we look at the prediction results obtained experimentally in section 4.5.1.

A better approach to calculate the force and height for sliding or rotating is to use an optimal solver that would take into account multiple constraints and a cost function to calculate an optimal value.

5.2 Planar Sliding

One can experimentally observe the unstable nature of pushing an object using a single point of contact: if you intend to push the object in a straight line you need to push it through the center of pressure (COP) in a straight line (figure 5.3), but you will find that no matter how precisely you position the finger through the center of pressure (COP), the object will start to rotate to one side or the other after a while. A small "bump" can suddenly rotate the object while the motion was somehow straight. Thus, the system is very sensitive to initial conditions and perturbations, making it experimentally unstable (not possible to maintain desired motions (e.g. straight lines)) in open loop control, and possibly chaotic in nature [1]. Therefore, prediction results are only valid for a limited period of time, and this period is highly dependent on the accuracy and precision of the sensory information. If we want to slide, in a controlled way, the object from one position to another, we need to solve this problem. A way to improve predictability and remove the unstable behavior of the complete system is by using a feedback control loop.

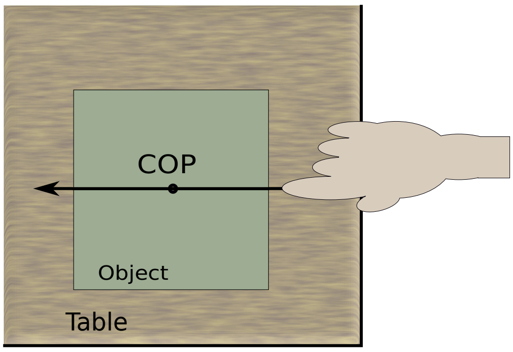


FIGURE 5.3. Pushing through the center of pressure (COP).

Another question arises: Where do we push the object? We need to select an initial pushing point of contact (PPC). There are several things to consider here. We could push at 1) a point that the finger will not slide initially, or 2) at a position where the finger has maximal control (like a perpendicular surface where the pushing point passes through the center of pressure (COP)), or 3) at a point that would generate the most straight motion in the direction of the final position goal. More complicated methods (but maybe better) may be possible, for example, 4) if we take into account the kinematic constraints of the robot (joint limits, reachable area, flexibility around that area).

[1] Proving the that system is chaotic or not is out of the scope of our work. For a way to test if the system is chaotic or not, look at [Gottwald and Melbourne 2004].

We selected the third option because it gave us a solution that produced a successful execution and it was simple to implement in the time frame we had. We would like to try other methods in the future.

The complete control strategy consists of 1) deciding the initial pushing point of contact (PPC), and 2) pushing the object towards the goal. Now we explain these two in detail.

5.2.1 Initial Pushing Point

The strategy to find an initial pushing point of contact (PPC) is based on trying to reduce control effort during pushing. We draw a line from the goal through the object's center of pressure (COP) until it touches the most far away (from the goal) lateral surface, this is the point selected to start pushing. If the finger-object friction is high enough it may even be possible to just push through the center of pressure (COP) towards the goal avoiding having to turn the box too much to reach it (figure 5.4).

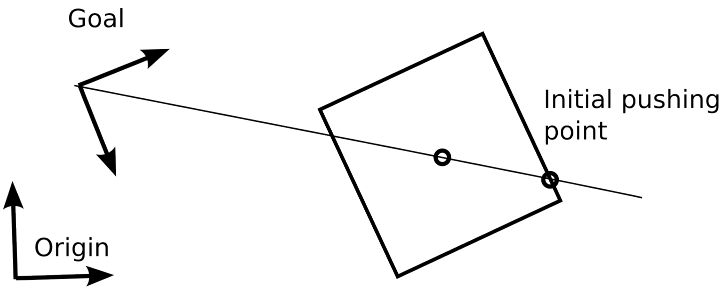


FIGURE 5.4. Initial pushing point

This method is implemented using a plücker polygon test [Yamaguchi and Niizeki 1997] [Shevtsov, Soupikov, and Kapustin 2007].

A practical detail is here taken into account: pushing on the object corners may not give good results because it may not touch the corner correctly or exactly or not at all because of accuracy errors on the object position estimations from the camera. In this case, if the pushing point of contact (PPC) happens to be too near to the corners of the object, then we move the pushing point of contact (PPC) away from the corner to a minimum predefined distance that depends mainly on the object position accuracy.

5.2.2 Pushing Towards the Goal

Since we are working with a non-linear system, and we want to have a global controller as much as possible, typical linear control design technics don't apply. In non-linear systems, there are no general ways to design a feedback controller. Some systems are well identified and have associated design technics [Matsuno and Tsurusaki 1999] [Murray and Sastry 1991] [Murray and Sastry 1993] [De Luca, Oriolo, and Samson 1998] [Wen 1996].

Our system is a so-called *drift-less* or non-holonomic system[2], a particular type of difficult to control dynamical system. They can be usually identified by having less control inputs than states, non-linear in general but linear with respect to the control inputs. A bicycle and a car are common examples of such systems. There exists a subset of these systems that can be converted to a so-called chained-form [Matsuno and Tsurusaki 1999], but no general methods to do this conversion have been devised. For the chained-form systems there exists a good list of different controller designs, which have different behaviors [Kolmanovsky and McClamroch 1995]. In our case we tried to derived a chained-form but no way was found to separate the variables to obtain a clean chained-form system. So we opted for a more intuitive design approach to the control rule.

Our particular desire was to slide the object from any pose to another one in **one single shot**. Usually non-holonomic controllers do multiple iterations near the origin to converge, but they don't necessarily make their best effort to get to the goal in the first approach [De Luca, Oriolo, and Samson 1998]. This type of solution is undesired for us because ideally we want to only push and not pull (the successive approximations would require pull and push, or push from both sides possibly using another hand). [De Wit and Sordalen 1992] use a different technic to reach the goal with pieces of circle circumferences. The problem with the circle approach is that the robot may not have enough workspace or kinematic freedom to execute the movements in case of a robot arm.

Our intuition and practice told us that by pushing the object to a position before the goal with a near-to-final orientation, then just pushing forward at the end would be enough to get to the goal. This is shown in figure 5.5(left). An even better solution is explained by [Pourboghrat 2002]. Instead of choosing a particular intermediate position, a direction angle towards this "point" is selected and, as the object gets near the goal, both this angle and the final desired angle, "naturally" converge. This gives a smooth continuous behavior. This is shown in figure 5.5(right). Since the control rule explained by [Pourboghrat 2002] depends on having a particular type of non-holonomic system which we couldn't transform ours to, then we had to derive our own control rule to follow the same behavior.

[2] Our system is also non-linear, and time variant: possible one of the most difficult systems to control.

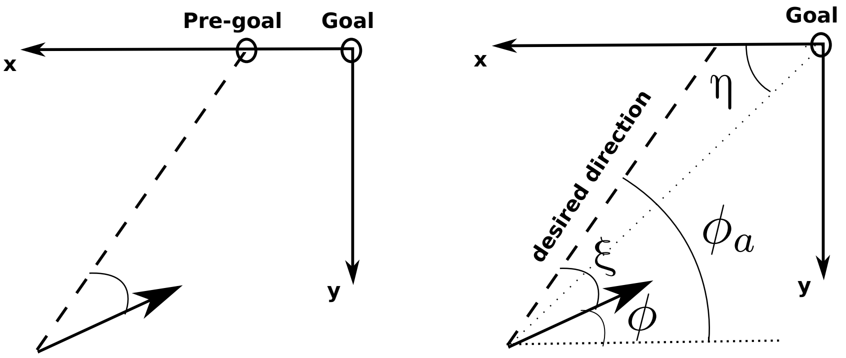


FIGURE 5.5. Pushing strategy. Left: Intuitive control: Push in the direction of the small circle, then turn the object and push towards the final goal. Right: control method from [Pourboghrat 2002]

If we rotate our model reference frame in a way that the finger pushing point is always in a new local x axis, then our local model equations are simplified to (using equations 4.22):

$$v_o = M\, v_c \tag{5.6a}$$

$$M = \begin{bmatrix} 1 & 0 \\ 0 & l_1 \\ 0 & l_2 \end{bmatrix}, l_1 = \frac{1}{1 + \frac{f_{max}^2 r_x^2}{m_{max}^2}}, l_2 = \frac{r_x}{\frac{m_{max}^2}{f_{max}^2} + r_x^2} \tag{5.6b}$$

This particular configuration is important for control, because it allows us to divide the control problem in one of pushing ($v_{forward}$) though the center of pressure (COP) to get straight motion and another ($v_{lateral}$) one to turn the object (with an undesired but unavoidable additional translational lateral movement) (see figure 5.6).

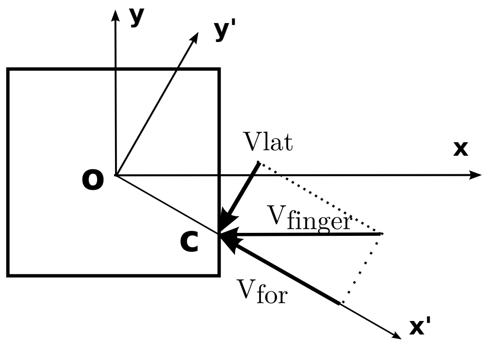


FIGURE 5.6. New reference frame for the controller

Calculating the variables d, ξ, ϕ, ϕ_a, η, (shown in Fig. 5.5) in a similar way as described by [Pourboghrat 2002]:

$$
\begin{aligned}
d =& \sqrt{x^2+y^2} && (5.7a)\\
\xi =& \phi_a - phi && (5.7b)\\
\eta =& s_x tan^{-1}\left(\frac{y}{|x|}\right) && (5.7c)\\
s_x =& sign(x) = \begin{cases} 1, & x \geq 0 \\ -1, & x < 0 \end{cases} && (5.7d)\\
\phi_a =& \left(\frac{2}{b}\right)\eta, 1 < b < 2, -\frac{\pi}{2} \leq \eta \leq \frac{pi}{2} && (5.7e)
\end{aligned}
$$

And using $b = 1.5$, we can compute our control values with the following equations:

$$
\begin{aligned}
v_{forward} =& v_{ref}\alpha && (5.8a)\\
v_{lateral} =& c_\xi + c_\phi && (5.8b)\\
c_\xi =& s_1\xi\, v_{forward}\, d_{extra}^2 && (5.8c)\\
c_\phi =& \frac{s_2\phi\, v_{forward}}{\exp\frac{l_2 * d}{l_1}} && (5.8d)
\end{aligned}
$$

In our case $s_1 = 830$ and $s_2 = 45$ gave an appropriate robust behavior. The values are high because the control system is working in a similar way as a sliding mode controller [Slotine, Li, et al. 1991] [Young, Utkin, and Ozguner 1996]. This control vector has to be rotated to the global reference frame and sent to the finger. d_{extra} is d (distance to goal) saturated to a particular value. α is a modulation value that slows down all the movements when we approach the goal.

Equation 5.8a makes the object always go forward to maintain movement. The reason of this is because if we generate something negative or zero here, the object will either not move or the finger will separate from the object.

The lateral speed depends on the control signals c_ξ and c_ϕ. The first one tries to correct the orientation of the object to go aligned to the auxiliary directional angle ϕ_a. This is to prepare the box to be near the final orientation before reaching the goal. Alone, this control signal appears to be enough, but the system easily behaves incorrectly near the goal. The reason is because near the goal the ϕ_a angle can change abruptly from small to high values, making the control rule change the orientation too much near the goal (figure 5.7(right)). But near the goal we don't want the object to follow this auxiliary direction anymore, because if we follow it this means worsening the goal orientation (the system gets unstable because ϕ_a changes too fast).

With d^2_{extra} and $\exp \frac{l_2 * d}{l_1}$ we reduce the effect of c_ξ and increase the effect of c_ϕ near the goal. c_ϕ tries to correct the object orientation to the final desired orientation. Since the finger-object vector should already be pointing towards the goal, this should give a proper behavior. We tried to make the control rule depend correctly on the objects parameters as much as possible (through l_1 and l_2). Typical control trajectories are shown in figures 5.8, 5.9 and 5.10.

Pictures during experimentation are shown in figure 5.1.

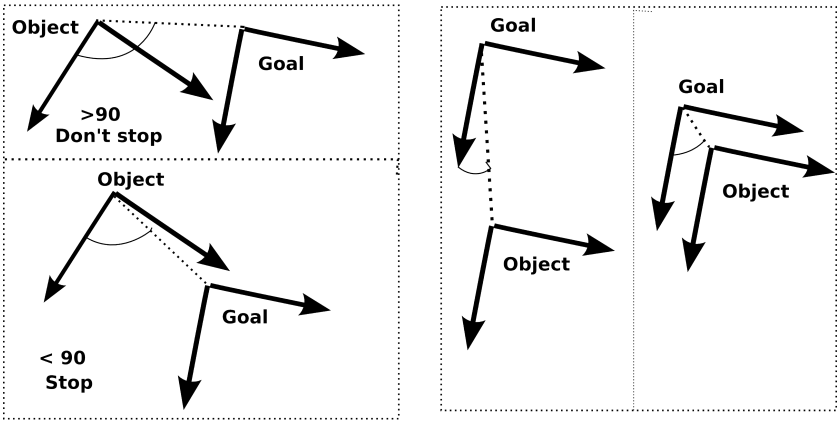
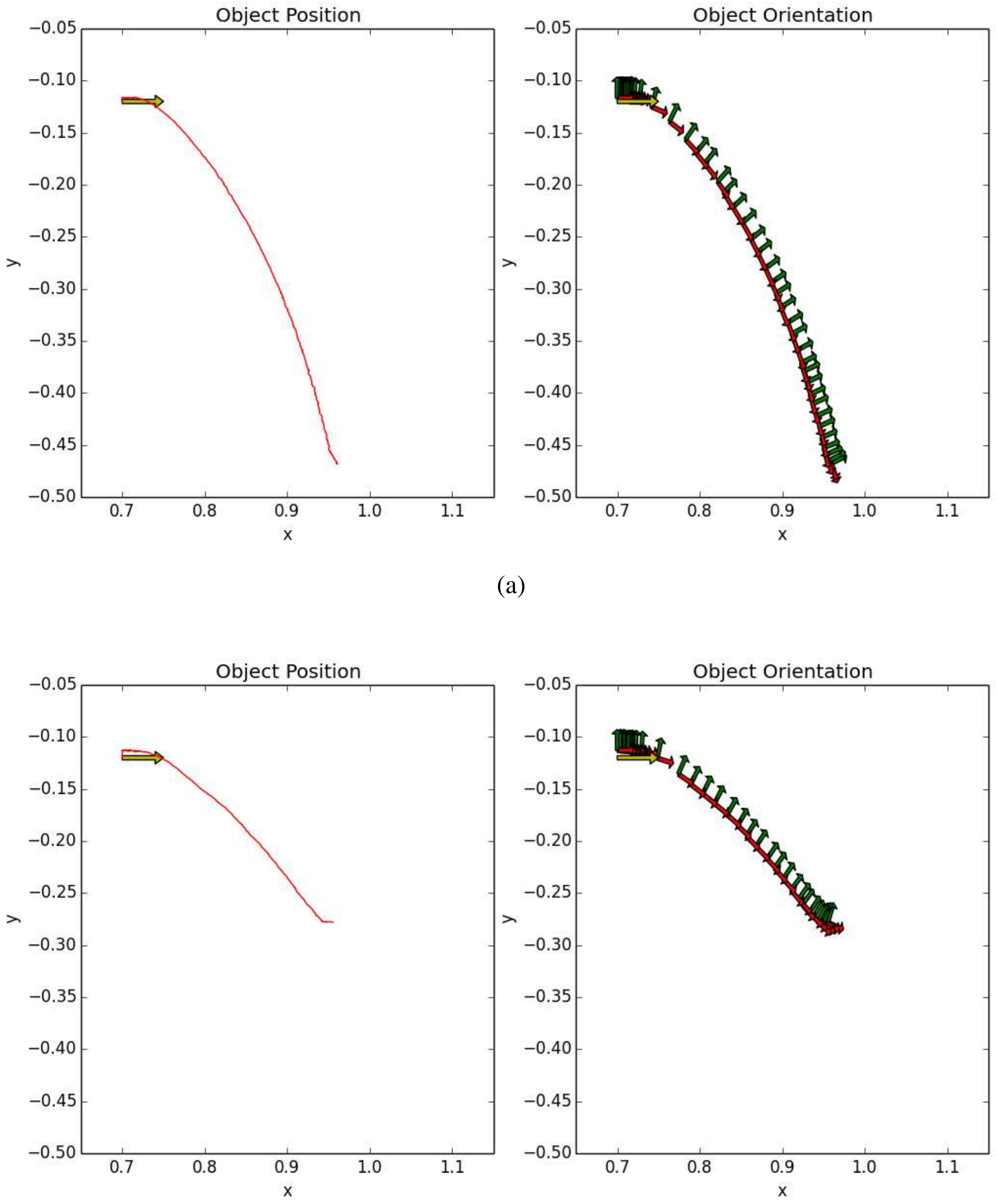


FIGURE 5.7. Situation near goal. Left: when to stop movement. Right: η and ϕ_a unstable near goal

The control correction that c_ξ provides it's imperative. And the ξ error must be maintained very near to zero to have a good final goal precision. To achieve this the s_1 constant must be very high. This creates a form of sliding mode control, where the system slides along this condition ($\xi = 0$) until it is near the goal (when c_ϕ becomes more important). Making c constant very high also proved to be important for robustness to object parameter errors and to some degree to camera object detection offsets.

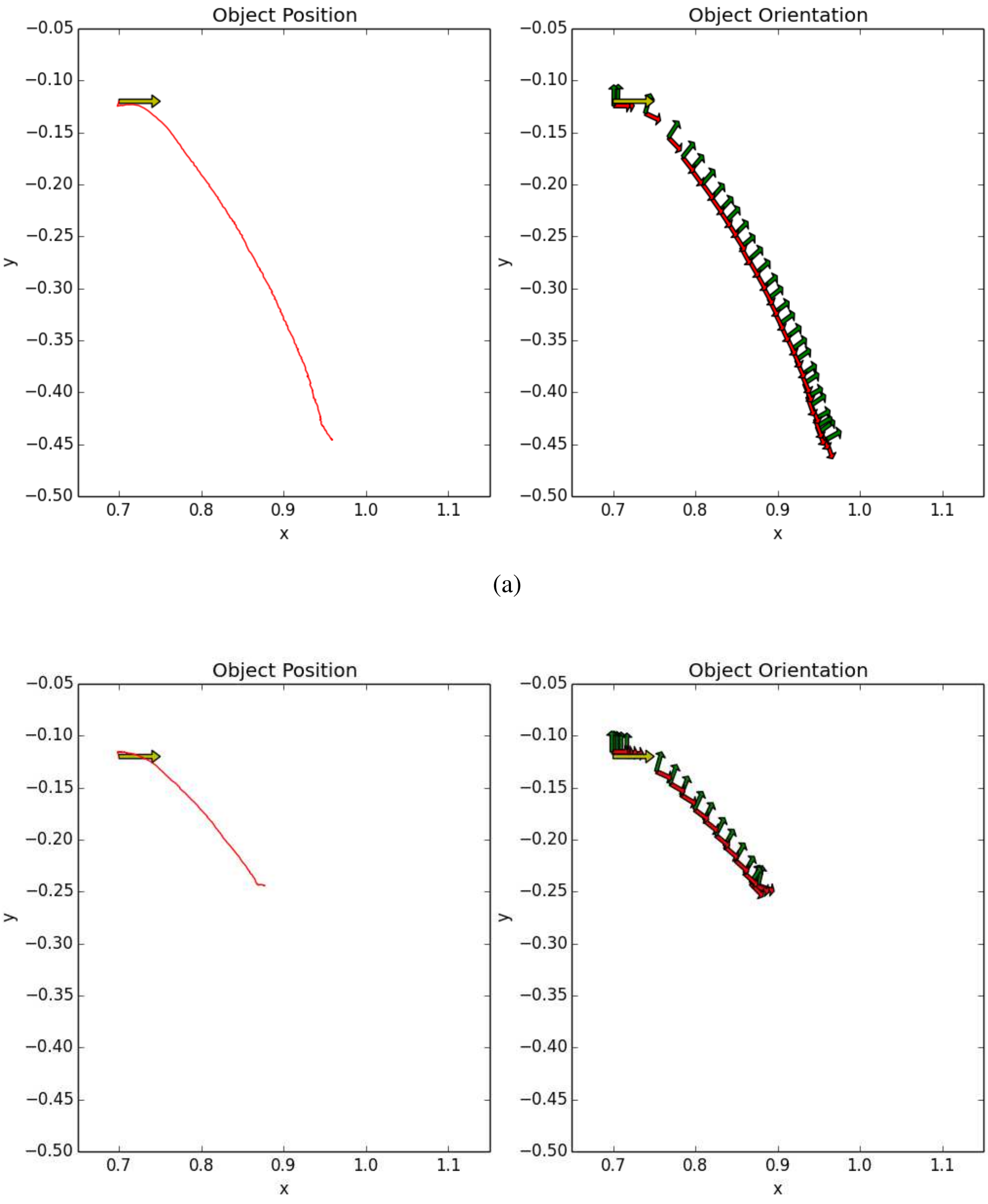
$v_{forward}$ and $v_{lateral}$ are further processed to take into account the finger friction cone, and they are saturated to be sure the finger doesn't get disconnected from the object; the lateral speed is decreased near the corners (before getting there) to avoid the fingers to keep sliding more and going away from the object. Again, we use equations 4.16 and 4.24 and some maximum safe slide velocity to limit the lateral speed. Both equations are modulated to slow down the movement as the object approaches the goal. For all this α is used.



(a)

(b)

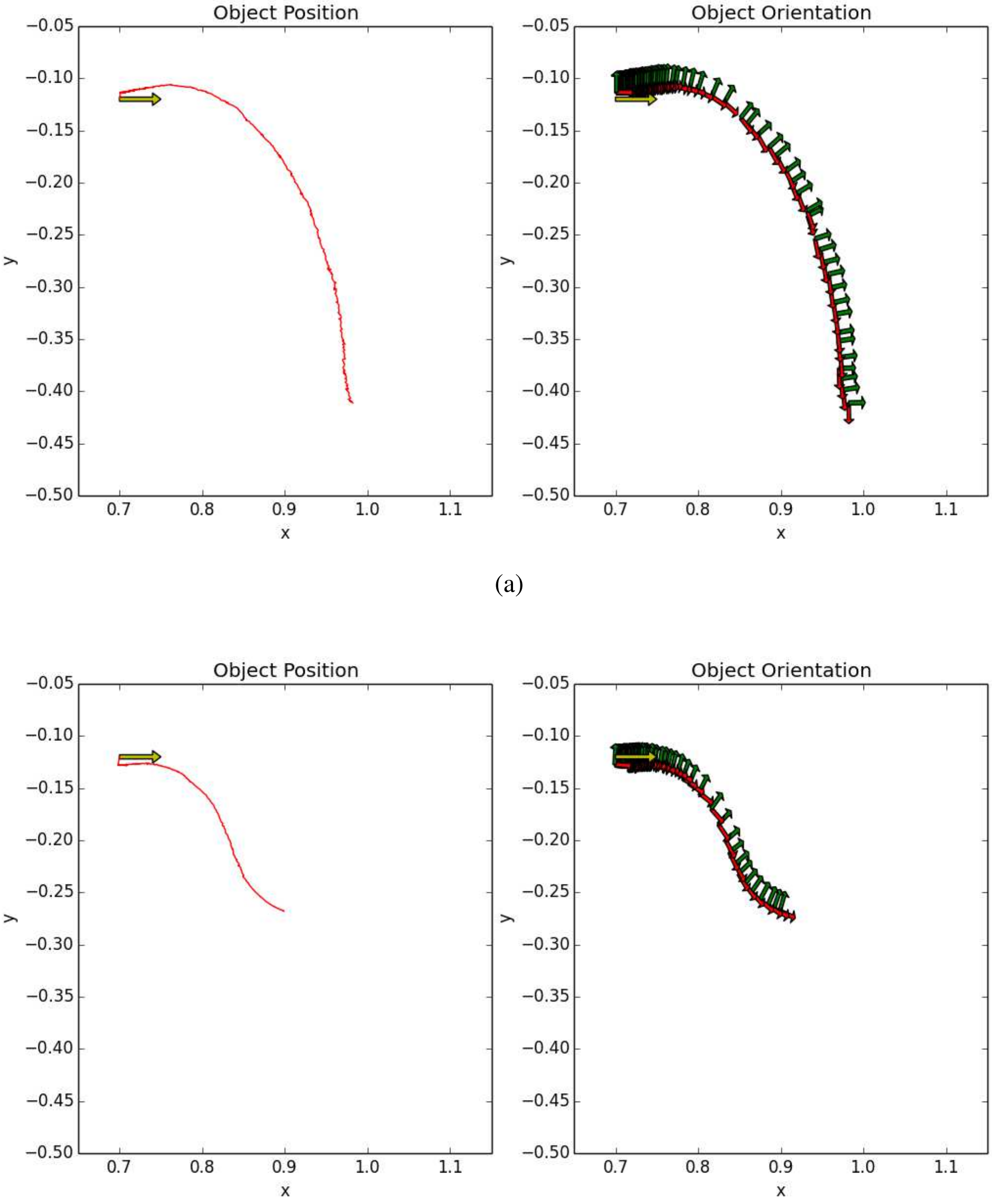
FIGURE 5.8. Pushing the Ice Tea box. Goal: 0.7,-0.12 . Yellow: goal. (a) Start position 0.96,-0.47. (b) Start position 0.96, -0.28. Left: Position trajectory. Right: Orientation trajectory



(a)

(b)

FIGURE 5.9. Pushing the Coffee box. Goal: 0.7,-0.12 . Yellow: goal. (a) Start position 0.96,-0.45. (b) Start position 0.9, -0.25. Left: Position trajectory. Right: Orientation trajectory



(a)

(b)

FIGURE 5.10. Pushing the Cereal box. Goal: 0.7,-0.12 . Yellow: goal. (a) Start position 0.99,-0.42. (b) Start position 0.9, -0.27. Left: Position trajectory. Right: Orientation trajectory

To stop the controlling action (near the goal), we calculate the angle between the object's x axis and the vector position from the goal, if this angle becomes bigger than 90 degrees, the program stops (figure 5.7).

5.2.2.1 Results

The trajectories for the ice tea and the coffee box are very similar (figures 5.8 and 5.9) because the shape of the base of both objects is also very similar. The trajectories for the cereal box (figure 5.10) are more curved indicating that turning the cereal box is “harder”. This is expected because for more elongated objects a higher torque value for the rotation friction is the result from the infinitesimal points of the cereal box that are further away from the cereal’s center of pressure (COP). There are many points in the cereal box farther away than for the ice tea or the coffee box.

With the three objects the precision that the goal was reached was quite impressive. In the case of the cereal, the error with which the object reached the goal was a bit bigger. This is due to the shape of the cereal box base: it is very elongated. Again, this elongation produces a higher rotation friction torque and therefore, this object is harder to “turn”, thus more difficult for the controller to minimize ξ.

The behavior of the robot with the object during sliding looked quite real and natural, showing very smooth movements with the arms, even when the object experienced perturbations (figure 5.12). A video showing this behavior is available at: http://www.youtube.com/watch?v=zMlQpj99bJE

There were situations when the goal was not reached. This usually happened if the position of the object was too near to the goal, but the orientation of the object was very different from the goal orientation. This can be explained with the turning capacity of each object. If the object was more elongated it meant slower turning speed, and therefore less space to correct orientation and execute the control rule correctly (ξ very near to zero). Along the control trajectory, ξ must get very near to zero at least once, if this doesn’t happen, this can be used to detect a failed trajectory because the object is too near to the goal for the desired goal orientation. A more elongated object needs a longer distance to the goal to correctly approach the goal.

5.3 Pushing with Perturbations

The figure 5.11 shows the reaction of the system to perturbations. (Human poking with the object while robot pushes the box towards the goal). Multiple disturbances happened at points x=0.9 y=-0.45 and x=0.85 y=-0.33 for the ice tea and at x=0.95 y=-0.42, x=0.91 y=-0.39, x=0.9 y=-0.35, x=0.88 y=-0.32 and x=0.8 y=-0.20 for the coffee box. As can be seen from the graphs, the robot was able to rapidly correct the ϕ_a angle minimizing very fast the error ξ angle every time a new disturbance happened. The goal was still correctly reached.

5.3.1 Perturbations and Prediction

When perturbations are present, the robot is expected to give bad prediction values from the Object Model System (OMS). These bad prediction values can be used to alert the high-level systems about something potentially dangerous or inadequate. The Object Model (OM) parameters may be wrong or other external factors are affecting object behavior. In either case, the high-level systems are responsible for any possible corrective action. In our case, the same low level controller is able to handle this particular situation by correcting motion after perturbations are removed, but if the perturbations are too strong, the control action may fail completely and the high level systems may have to plan another course of action (observe the new situation and use another type of controller for example picking the object up, instead of pushing).

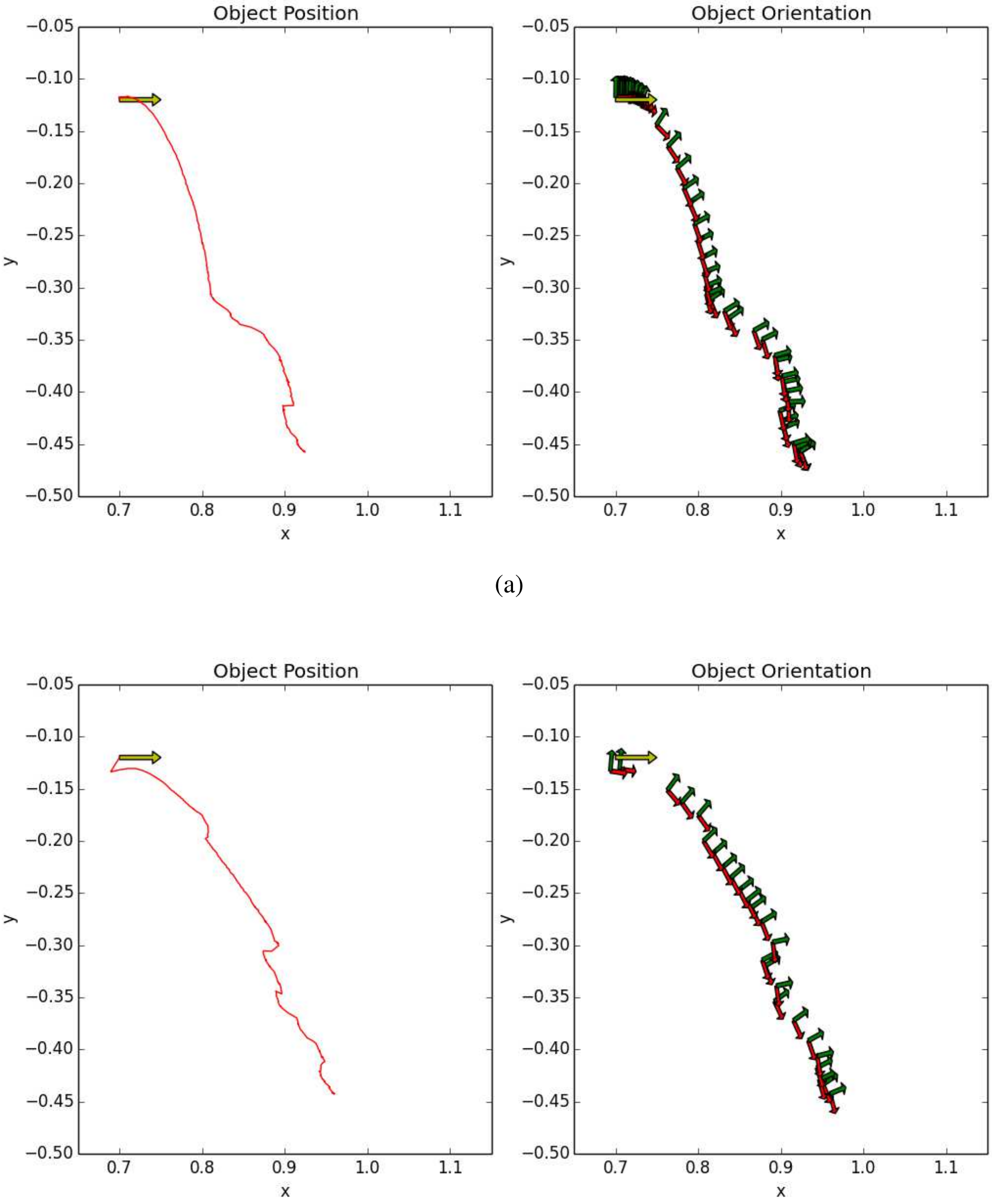
In figure 5.13(b) a close up of such unexpected behavior is presented. Normally the red circles and the green circles are somewhat along the same trajectory, indicating a continuous smooth trajectory towards the goal. But, for example, at point x=0.898 y=-0.412 one of these perturbations happened. At this point, the predicted (red circle) object pose strongly differs from the next camera measured object pose. Measuring the distance between each last red circle to the next green circle and with an appropriate threshold value these type of unexpected behavior can be detected. In our case we used a threshold distance of 0.0025m which gave good experimental results (detected all strong perturbations, and around 80% of soft perturbations). This threshold value could be learnt using a machine learning algorithm.

5.4 Discussion

If the robot wants to topple or slide the object, the decision is quite simple as stated in this chapter according to section 5.1. But, to be precise, there are infinite solutions to this problem and therefore, an optimal solver could give an improved answer that could consider additional optimization constraints.

The rest of the chapter concentrates on the planar sliding problem: *What movements must the robot execute in order to move the object from one position to a desired goal pose?* The problem is divided into two problems: At which point should the robot start pushing? How should the robot keep pushing to guide the object to the desired goal?

The initial pushing point problem is solved with a simple yet successful strategy: draw a line from the goal through the object and the farthest point from the goal that crosses the object is the initial pushing point. Again, this problem could have infinite solutions and other strategies that take into account additional constraints could be considered to generate an optimal solution.



(a)

(b)

FIGURE 5.11. Robot reaction to perturbations. (a) Ice Tea. (b) Coffee box. Left: Position trajectories. Right: Orientation trajectories.

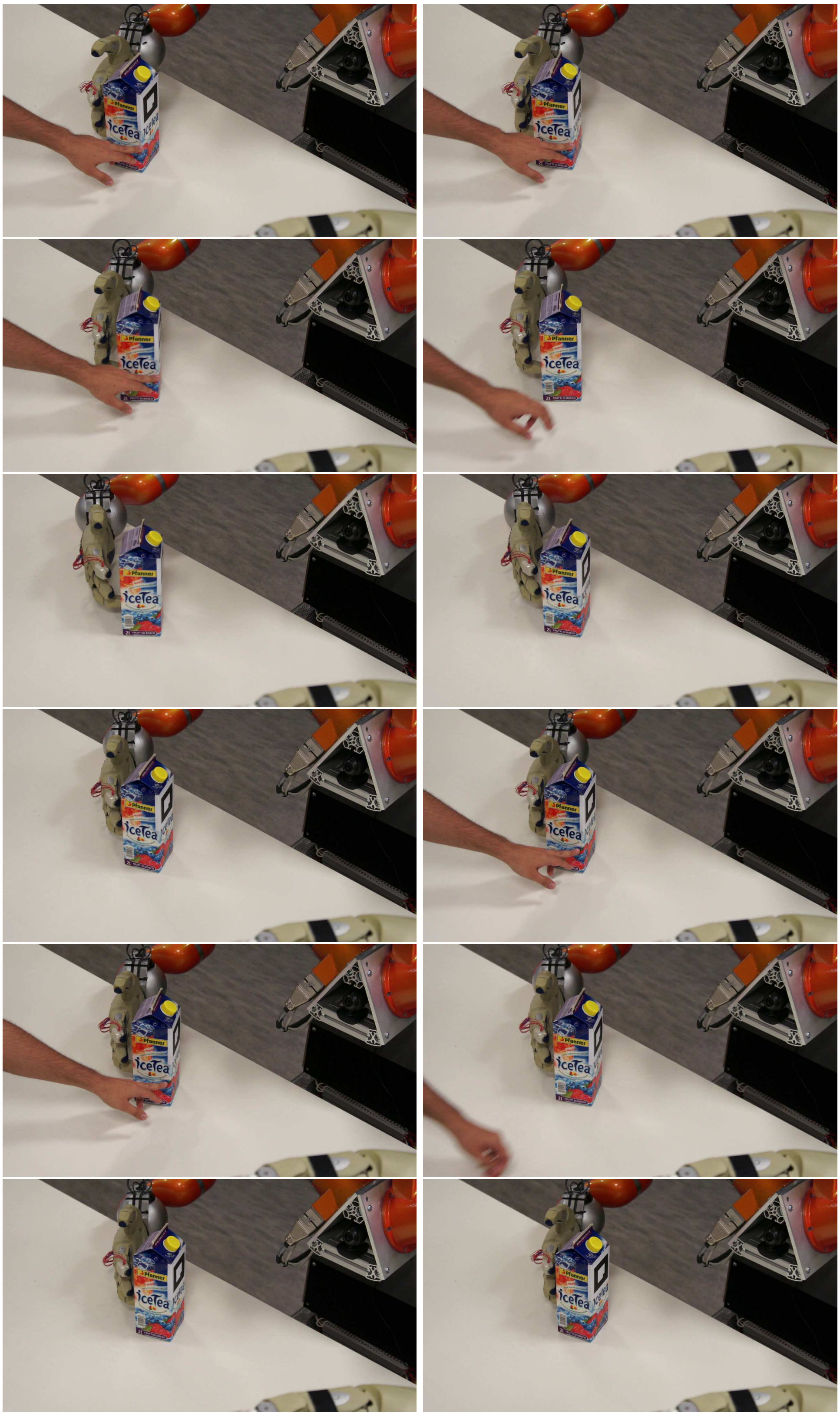
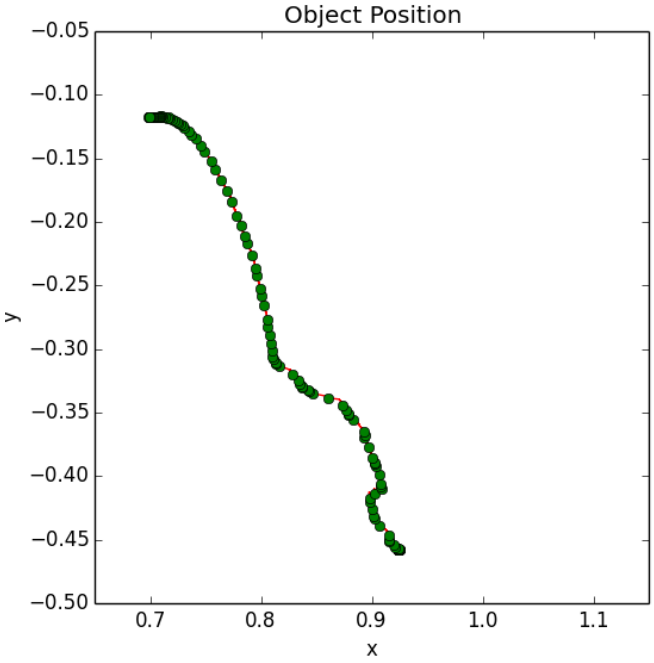
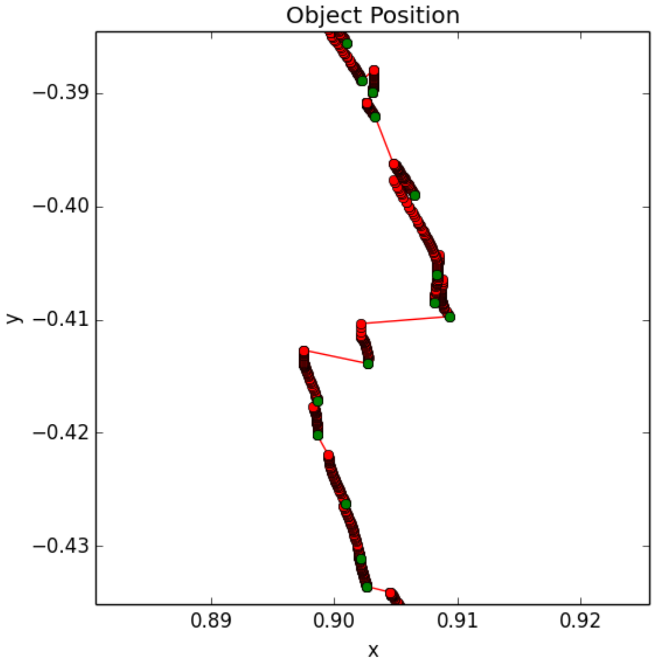


FIGURE 5.12. Pushing sequence with perturbations from figure 5.11(a).



(a)



(b)

FIGURE 5.13. Detecting unexpected behavior (perturbations). (a) Complete trajectory. (b) Close up at a perturbation. Green: Camera object pose estimate. Red: Predicted object pose.

Pushing towards the goal is a more complicated problem: we want to reach the goal with a particular orientation in one single movement using a feedback controller. We want to avoid a reciprocating movement near to goal to reach it. Because the object dynamical system is non-holonomic, it is impossible to reach the desired pose with a direct independent control action in each degree of freedom. The usual control strategies to solve this problem require a reciprocating movement at the end near the goal. We want to avoid such movement because of the reasons explained at the beginning of the chapter. The solution explained in this chapter makes use of a simple yet very clever pushing strategy (explained by [Pourboghrat 2002]): follow a line that pushes the object just slightly to the side of the goal (using an angle slightly bigger that the angle needed to reach the goal directly). If this line is re-drawn at every control step using the same angle the object will reach the goal with the correct desired final orientation without any reciprocating movement. Based on this strategy a feedback controller (that resembles a sliding mode controller) is derived (empirically).

Test results of the sliding box controller show that the performance is quite impressive, and the object approaches the goal in one single shot with the correct orientation. Because the controller resembles a sliding mode controller, the correcting force is very high and the results when perturbances are present are also very good. At this point, we subjectively noticed that the robot produced movements that looked very natural, and these movements definitely depended exclusively on the Object Model (OM), its properties and any external perturbances. As we mentioned before: *the object dictates how the robot must move*. We recommend the reader to download the video from the link mentioned in section 5.2.2.1 to be able to appreciate the results more clearly.

CHAPTER 6

Conclusion and Future Directions

To perform manipulation activities skillfully a robot must be able to predict how an object will react to its actions, and how the robot must act on the object to obtain a certain result. It is also desirable that the robot quickly adapts to objects with different properties and still manipulate them skillfully. And finally, if a robot is able to characterize the objects in a way that is easy to represent in a symbolic way, it can facilitate to reason and plan with them in bigger contexts, therefore, enabling the robot to perform more complex tasks. The Object Model System (OMS) (chapter 3) enables our robot to predict and control objects in a flexible way; it allows the robot to select a corresponding Object Model (OM) for a particular object type and use this Object Model (OM) to predict and control object behavior. Using object exploration (section 4.4) our robot can adapt the Object Model (OM) to a new object instance almost immediately. The Object Model (OM) used for the *pushing a box* example (section 4.2) uses parameters (friction coefficients, object shape/dimensions, weight) that express knowledge and meaning related to the object and the action, therefore it is possible to reason about them. Because these object parameters are explored by the same robot that manipulates the object, this knowledge is *grounded* and a proper behavior is expected when connecting this system to a reasoning and planning system. The very natural movement and reaction of the robot manipulator observed during the experiments (section 5.2.2.1) demonstrate a skillful handling of the object.

Because the control rules depend on the Object Model (OM) and its parameters we conclude that *the object commands the robot manipulator* along the control trajectory and not the other way around. The user can only select/modify the final desired goal and the rest is determined by the particular object. Subjectively, this could explain the *natural* aspect of the motion during pushing and even more visible when perturbations are present.

In this thesis, we present an integrated framework (Object Model System) that gives to a robot the capability to manipulate objects by knowing how the objects work. The Object Model System (OMS) concentrates on the object and not so much on the robot; the object dictates the way the robot must manipulate the object in order to get it to a desired goal.

Also, since we used mathematical formulas to describe the object's behavior, and these formulas have parameters (object properties) that are relevant to the object and that a human can easily and readily comprehend, the system has a strong potential to be connected to a high level reasoning and planning system. An exemplification of how to achieve this interconnection (using an Action Language) is presented in sections 3.3 and 3.1.

The Object Model (OM) prediction capabilities were correctly demonstrated and based on current sensory information the next object position was estimated.

While the robot is capable of predicting a particular object behavior, even then, there are situations where object behavior may still differ (external perturbations, wrong or changed object properties). In sections 4.2 and 5.3, we present a particular situation (external perturbations) and how the robot can detect this out-of-the-normal situation to inform the high level systems.

The Object Model System (OMS) (explained in chapter 3) is applied in chapter 4 to a particular experimental example: pushing a box. Chapters 4 and 5 further explain the methods used to predict and control the object. This Object Model (OM) example was performed with our TUM-Rosie robot in our intelligent kitchen scenario. Results were extremely successful with the robot being able to push different everyday objects around a kitchen table with an extremely good goal precision. Even when the object was under the influence of strong external perturbations, the control system was able to automatically correct the object state to achieve the final commanded goal.

Two important concepts in robotics are the symbol grounding problem ([Steels 2008]) and the embodiment ([Moravec 1988]). We believe our Object Model System (OMS) and in particular our box model can achieve symbolic grounding by the use of the action language described in section 3.3 and embodiment is available when the robot itself acquires the current object properties, therefore adjusting the object model to the current object through its own sensors (robot body). The embodiment was exemplified in our TUM-Rosie robot as shown in section 4.4. The results of this were the object properties obtained (see tables 4.1 4.2), and the good Object Model (OM) predictions.

The planar sliding sub-model proved to be a very challenging model to design. A kinematic model relating the current finger pushing position to the center of the object was obtained. Many control methods for this system were tried before finding the solution presented in chapter 5. The *one finger planar sliding pushing problem* corresponded to control a *non-linear, non-holonomic (and different than the car-like system), time-variant, naturally unsta-*

ble[1], *multivariable system*, may be one of the most difficult types of systems to control. Yet, results show the impressive control capabilities of the controller and the adequate object pose prediction of the system.

Although, modelling our box object behavior proved to be challenging (in particular the planar sliding sub-model), in general, we believe that this task has to be done only once for each type of object. This model can be used in any robot that has the same type of sensory and actuation capabilities (velocity control in an actuator, finger tip force sensing (or equivalent), and global object pose estimates). After the modelling is finished, these Object Models (OMs) can be shared (and also the properties of particular instances of objects), thus, facilitating the use of these Object Models (OMs) by other robots. In a futuristic scenario where a person purchases an object, the object could come with an Object Model (OM) included, therefore, your robot could start using that object “out-of-the-box”.

A software infrastructure is presented in appendix A. Different parts of this infrastructure were tested on different robots and finally the complete system was integrated in our TUM-Rosie robot using YARP and ROS middlewares. The source code (written in python) is freely available for download at `https://github.com/arcoslab/` and the instructions for installing and testing are available at `https://wiki.arcoslab.eie.ucr.ac.cr/doku.php?id=object_manipulation_robot_simulator`. The source code is released under the GPLv3 free software license.

6.0.1 Future Work

Current work is underway in constructing new models and expanding and improving the existing ones. The idea is to build a library of Object Models (OMs). Such library is expected to give robots the possibility to more often know what’s happening with the objects around and to have more options to manipulate them. With such a library, a robot could be continuously aware of what is happening around it. To build such a library, initially we are looking into objects like cylinders, drawers, cups, chairs, tables, windows, kitchen appliances, kitchen utensils, food (deformable, transformable or shape-shifting objects), liquids. We also want to model free-moving behavior (objects been thrown, of freely sliding).

Through the *push a box on a table* example we believed we have provided a solid proof that the idea of Object Models (OMs); that robots can manipulate objects in a more meaningful way because,

objects dictate how manipulation should be executed,

[1] Instability not proven yet.

is a powerful method for robot control. And we hope that by building an Object Model (OM) library, one day, a robot may be able to continuously work assisting humans in everyday tasks.

We are also looking to improve the knowledge of each model: that is, the predictive and control capability for each model. For example, is surface deformation important for a box object? If it is, such sub-model must be included.

Another part of the Object Model System (OMS) that can be extended is the Object Model (OM) parameter acquisition. Another way of finding the parameters is starting with a guess of all the parameters and then depending on the prediction errors adjust these parameters until all of them predict correctly the behavior of the object. This is called *on-line parameter estimation* [Kügler 2003], [Andrieu, Doucet, and Tadic 2005]. The advantage of this method is that it is not necessary to compute the closed form solution for the model and that it can find the parameters even in situations where multiple parameters are tightly coupled together and the experiments to find these parameters will not provide each parameter alone. Such system may constantly check object prediction and try to adjust continuously the model parameters to get the best possible prediction. This adaptability has to be limited to allow for out-of-the-normal situations which will be expressed as incorrectly predicted behavior. The starting parameters for such system could be obtained from a knowledge database (prior knowledge from same robot or other interconnected robots). Some parameters can be measured by other means, e.g. using vision to recognize the shape and dimensions of the object.

For each object, knowing the limits of what can be done with it could be useful. Such a system would require to simulate what would happen to the object if the maximum and minimum robot force or speed (or any other actuation variable) is applied to it. It is like a constant affordance estimator that gives the robot an idea of the control limits for a given task and object. A system that uses simulation for a similar purpose is presented by [Fedrizzi et al. 2009] and [Stulp, Fedrizzi, and Beetz 2009].

Something we are considering for improving the results of our predictors is to include multiple time-windows with a length up to a maximum point where we can trust the prediction for each Object Model (OM). This could be learnt comparing predictions with reality with and without external perturbations during several trials.

Currently, the integration of object pose estimation from different sensors is combined in a very simplified way (camera estimates and object predicted pose from the finger tip force): the robot *believes* completely the camera object pose estimates when available, and then, when not available, it believes completely to the predictions based on the finger tip forces and the Object Model (OM). A way to improve this is by using a Kalman filter to fuse this information in a more probabilistic manner.

Predictability when the robot pushes the object when the finger is at the object corners was not tested and is expected to be imprecise. This is because the camera estimated object pose accuracy and precision is not good enough for this task. Testing of this type of extreme situations is required.

Some small camera calibration errors were detected during our tests. For example, the estimated object pose (center of the object or center of pressure (COP)) could be a little bit off to one side (because when the robot pushes the object through this point the Object Model (OM) predicts that no rotation should happen to the object, but it happens). Currently, an offset to the camera estimated object pose has to be manually added. An autocalibrate procedure could improve this. Such procedure could instruct the robot to "play" (by poking) with the object and using the Object Model (OM) predictions, the robot could find out where the center of the object is: using the pushing point of contact (PPC) where the object rotation is minimized the real center of pressure (COP) of the object could be inferred, and the offset error calculated. A similar approach could be used to better estimate where the object corners are, and therefore improve predictability in such areas. Possibly a system like the one shown by [Maldonado, Alvarez-Heredia, and Beetz 2012] could be used to provide better local sensing and improve object shape and pose estimation.

Currently the controller of our object model only uses camera pose estimates to update the control command. It could be possible to use the predicted object pose using the finger tip force information together with the camera pose estimates to control with a higher update rate in a similar way as it is done with feed-forward control.

A question that remains open is: How to deal with obstacles? Should we get rid of them before manipulation? We could always have an initial situation with obstacles. How do we integrate the manipulator null space with our object model?

Several other solutions taking into account multiple points of contact for the planar sliding problem have been consider in the literature. Usually the result is a simpler but more limited controller than ours. We could integrate the possibility to use n-fingers to accomplish the pushing task and depending on current constraints different quantity of fingers would be used.

Inertial effects are not taken into account in our box model. If we take such effects into account our system could be able to predict and control situations like throwing objects or free planar sliding.

Some complicated to explain object behavior could be model as an *error* in the Object Model (OM). Such error could be learnt using a black box machine learning algorithm, therefore avoiding the need to manually model something, maybe too complicated to explain, but too small to be of general importance. Modelling such “unimportant” behavior could have a use in the case where the robot needs to *perfect* its predictions without increasing complexity in our compact models (we don’t want to add too much complexity to the models because it may complicate the symbolic mapping between the Object Model System (OMS) and the high-level systems).

Our pushing-a-box example is currently making use of ARToolKit to detect the object pose using printed markers attached to the objects. We would like to have a system that works with normal everyday objects not modified in any way. 3D perception systems like the ones described by [Marton et al. 2010] and [Klank 2012] which can detect objects without any modification could be likely integrated with our Object Model System (OMS).

In section 5.1, a simple controller for toppling or sliding is presented. Such controller is simplified in such a way that other possible solutions are not taken into account. We could state the problem in such a way that an optimal solver could take into account the complete solution space and some constraints. This will give the robot a more complete solution.

When the robot starts to slide the object, it selects a position based on method number 3 described in section 5.2. Method 4 would be a better choice because it would take into account the robot kinematics. We could also take into account the null space of the object to calculate the initial pushing point (for example, in case of a box, each of the four sides could be equally used for pushing). Such an approach is explained by [Kresse and Beetz 2012].

Currently in this thesis not much is explained of how to transport the prediction alerts back into the high-level systems. A possible solution to this could be to create a mapping from prediction alerts to an action language in a similar manner as explained in section 3.3.

We have consider modelling and controlling only objects with the Object Model System (OMS). We may need to extend this notion to model instead of objects also robots and people, and even the robot itself. For example, if we want to move an object, we first need to reach it, but the robot may be too far away from the object, then the robot must command itself to move near to a position where the object is in order to push it. This will be a separate action, therefore an action sentence together with a mapping and an Object Model will be needed. In this case the subject of the sentence is the robot itself (“position yourself at ..”). [Stulp et al. 2012] presents a possible method to accomplish such task. Also, modelling people may give the robot a better assistive behavior, because the robot is more capable of predicting what a person may need.

To detect unexpected situations section 5.3.1 explains a method to achieve this. A possible improvement to this method could employ machine learning and some training to let the system learn the threshold. Is training going to be needed with each and all types objects and with all different object properties? If the answer to this question is positive, this may be inadequate and a more analytical formula for calculating the threshold for each object model would be more appropriate.

There are some situations in which the controller of the planar sliding sub-model doesn't achieve the desired goal (e.g. object too near to goal but with a very different orientation). A previously complete simulation of such situation could prevent the robot to even try such control action before hand. [Stulp, Fedrizzi, and Beetz 2009] describe a methodology to implement such a system. Other problematic situations include: self robot collision, camera occlusion, out-of-reach object during pushing, kinematic constraints (joint limits, link lengths, capability map).

Appendix A

Hardware and Software Infrastructure

The Object Model System (OMS) and our *pushing a box* example together define the necessary hardware and software capabilities that a robot needs to properly achieve our desired goals.

The solution presented in this thesis tries to be as robot-agnostic as possible; the proof-of-concept example presented in chapters 4 and 5 has an Object Model System (OMS) that just outputs velocity instructions to an actuator that pushes an object and expects velocity and force measurements from this actuator and absolute object positions estimations (but at a slower rate) (e.g. from a camera) to properly function. Therefore a robot with force sensing and velocity movement should be enough. Out of our four robots, the B21, PR2, iCub and TUM-Rosie, the last one was capable of all this requirements. There were other considerations that made TUM-Rosie more appropriate for our system, and we will explain them in detail in the following sections.

Many software systems were developed to complete our final testing infrastructure and these systems were tested in many different robots and demonstrations. In this chapter we explain these systems in detail and the robots that were used to test them.

The last sections of this chapter describe the parts of the software architecture used to implement our Object Model System (OMS).

A.1 Robots, hardware and experimental setup

At the Intelligent Autonomous System (IAS) group, we had at our disposition a good variety of robots, each with different hardware capabilities. We will present each of our robots and which software and parts of our system architecture were tested on them. Also, we will explain the reasons to selected TUM-Rosie to test our proof-of-concept example.

A.1.1 Robots

The RWI B21 ("Bender") (Fig. A.1), was built by Real World Interface, Inc. (now iRobot). It included (after modifications):

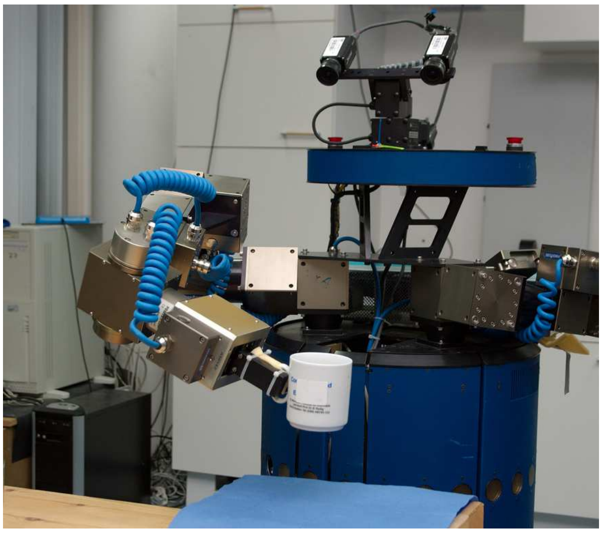


FIGURE A.1. B21 Robot

- A semi-onmidirectional mobile base
- Two 6-DOF PowerCube (Schunk) arms with two grippers.
- PTU unit, type DP-47
- A Swiss-Ranger SR3000 Time-of-flight camera.
- Two high-resolution cameras (2MP) forming stereo setup.
- A SICK LMS400 mounted on the tip of one of the PowerCube arms.
- A SICK laser scanner for localization and navigation mounted on the base.
- A Multitude of infrared and ultrasonic sensors mounted around the robot's chassis.

This robot was the first robot we used to test our first complete arm navigation software (section A.2.2) (for avoiding obstacles and reaching goals). Also, we used the Player Project [The-Player-Project 2013] with gmapping (SLAM), amcl (adaptive Monte Carlo localization) and wavefront modules for mobile localization and navigation.

The iCub was our next robot (Fig. A.2). Many were built by the IIT (Italian Institute of Technology) and one was awarded to the Intelligent Autonomous System (IAS) group. The iCub is a complete humanoid robot with the size of a 1 meter tall child. The iCub is able to crawl, and also grasp objects. Its main features are:

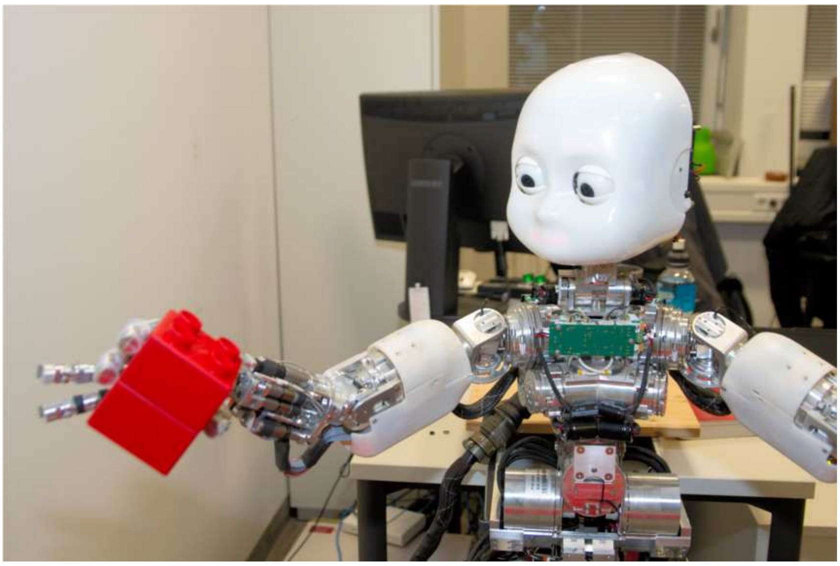


FIGURE A.2. iCub Robot

- 53 actuated degrees of freedom (DOF).
- 7 DOF arms.
- Two cameras mounted as eyes.
- Facial expression capable (eye brows, mouth, eye lids)

During one of the first iCub summer schools the first version of our arm navigation software (section A.2.2) was programmed and successfully tested.

TUM-James PR2 is a humanoid robot built by Willow Garage. Features:

- Omnidirectional mobile base.
- Two 7-DOF arms.
- Two grippers, pressure sensing capable.
- A 3D point cloud sensor.

- 5 Megapixel camera in the head.
- Narrow-angle and wide-angle stereo cameras in the head
- Cameras in the forearms
- Tilting Hokuyo UTM-30LX laser scanner
- Hokuyo UTM-30LX Laser Scanner in the base

TUM-Rosie is a humanoid robot assembled by our group [Beetz et al. 2011] (Fig. A.3). The main feature of this robot is that it is compliant with the environment; it uses impedance control in the arms and fingers. Main features:

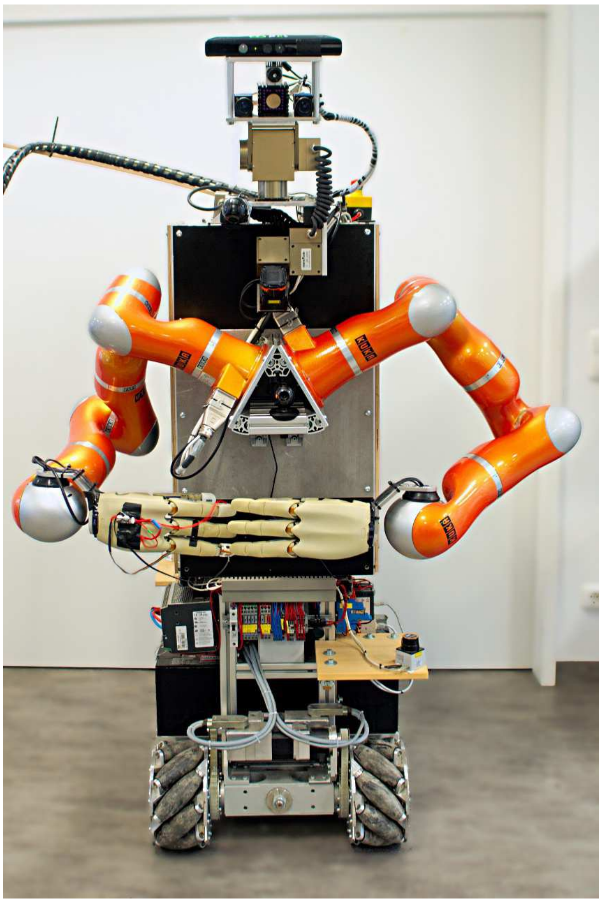


FIGURE A.3. TUM-Rosie Robot

- Omnidirectional mecanum-wheels mobile base.
- Two 7-DOF KUKA LWR arms.

- Two DLR-HIT Hands [Liu et al. 2007].
- A pan-tilt head with a 3D point cloud sensor, two high definition cameras, flir camera.
- Two Hokuyo laser range scanners on the mobile base for navigation.
- Two high performance computers.

With this robot we tested all our object model systems and performed all the experiments.

A.1.2 Experimental setup: The Intelligent Kitchen

Our experimental scenario is the assistive kitchen [Beetz et al. 2008] (Fig. A.4). In this kitchen various everyday objects are commonly used for experiments. The kitchen is an scenario that represents a normal human kitchen, with normal objects like cooking pans, glasses, plates, and so on. In the kitchen, human activity is monitored using multiple sensors and also a set of cameras [Bandouch, Jenkins, and Beetz 2012] [Albrecht et al. 2011] [Beetz et al. 2008].

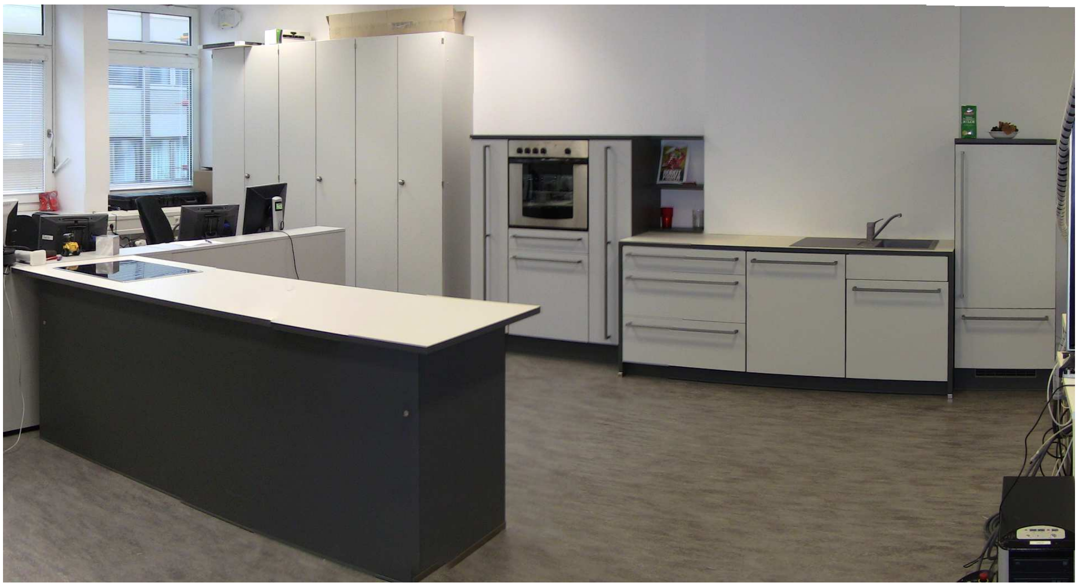


FIGURE A.4. The Intelligent Kitchen

For pushing, our type of objects are boxes of various dimensions. In particular we use an ice tea box (with liquid inside), a coffee package and a cereal box (Fig. A.5)

The place where the pushing experiments take place, is a kitchen table, where the robot is able to slide the objects freely around. The robot is pre-positioned in front of this table manually or using a localization and navigation software module.

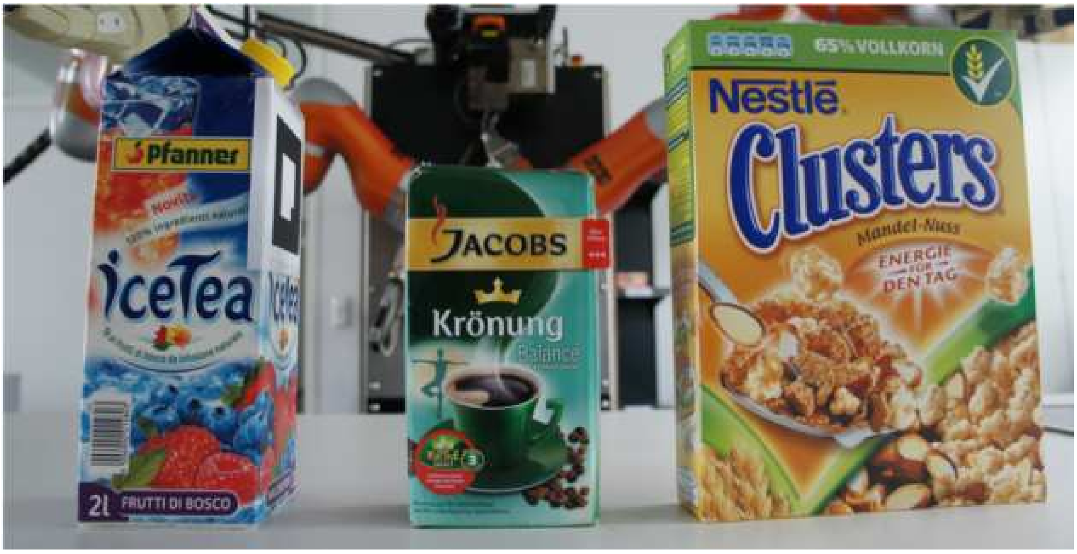


FIGURE A.5. Experiment objects

A.1.3 TUM-Rosie

Our manipulation platform (Fig. A.6) consists of a robot with an omni-directional base, two 7 DOF Kuka-DLR light-weight arms, two DLR-HIT-Schunk hands and two cameras. The hands and arms are equipped with torque sensors in all the actuated joints, and impedance control in joint space is used along all the experiments. The hands have 4 fingers each with 4 joints and 3 independent DOF and 3 torque sensors.

TUM-Rosie was our selected robot. This robot is the only one from all our robots that has torque sensors on almost all the degrees of freedom of the arms and hands. With these torque sensors, we are able to estimate the forces applied on the finger tips using the algorithms explained in section A.2.3.1. This force is used to predict object behavior using the equations explained in chapter 4.

Another important feature of this robot is impedance control. With this type of control the robot can have a soft contact with the objects, therefore, dampening oscillations on contact and not destroying or generating bad behavior on the objects: it is soft and compliant with the environment [Hirzinger et al. 2001].

We use the finger torque sensors instead of the torque sensors on the arms to calculate the finger tip forces because this sensors are more sensitive, have less offsetting, and are less noisy.

A.2 Software Architecture

Figure A.7 presents the software architecture diagram of our implementation.

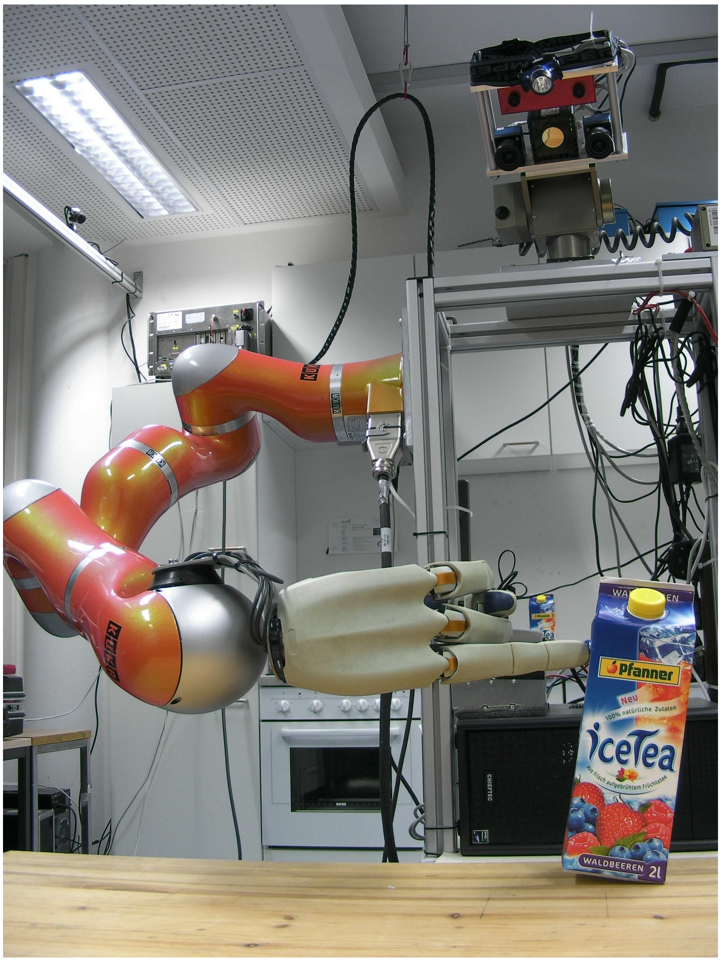


FIGURE A.6. Assistive kitchen robot

The low level hardware interface connects the hardware devices (hands, arms and camera) with our software modules. They relay to our system all the information available from the hardware like: joint positions, torque sensing, and the video signal. They also receive motor commands to move the joints or change the stiffness in case of impedance control joints.

The robot interface is responsible of transforming the commands received from the Object Model System (OMS) in object coordinates to motor coordinates. For the arms and hands, the vector field based close loop inverse kinematics (VFClik) module is used. The finger pose with respect to the hand base is used as the tool frame of the vector field based close loop inverse kinematics (VFClik) end-effector (explained more in detail in section A.2.2.1). It also provides finger tip forces (with respect to the robot base) estimated from the finger joint torques and current kinematic configuration of the arm and finger together.

All data in object coordinates is relative to the robot base, including the object pose estima-

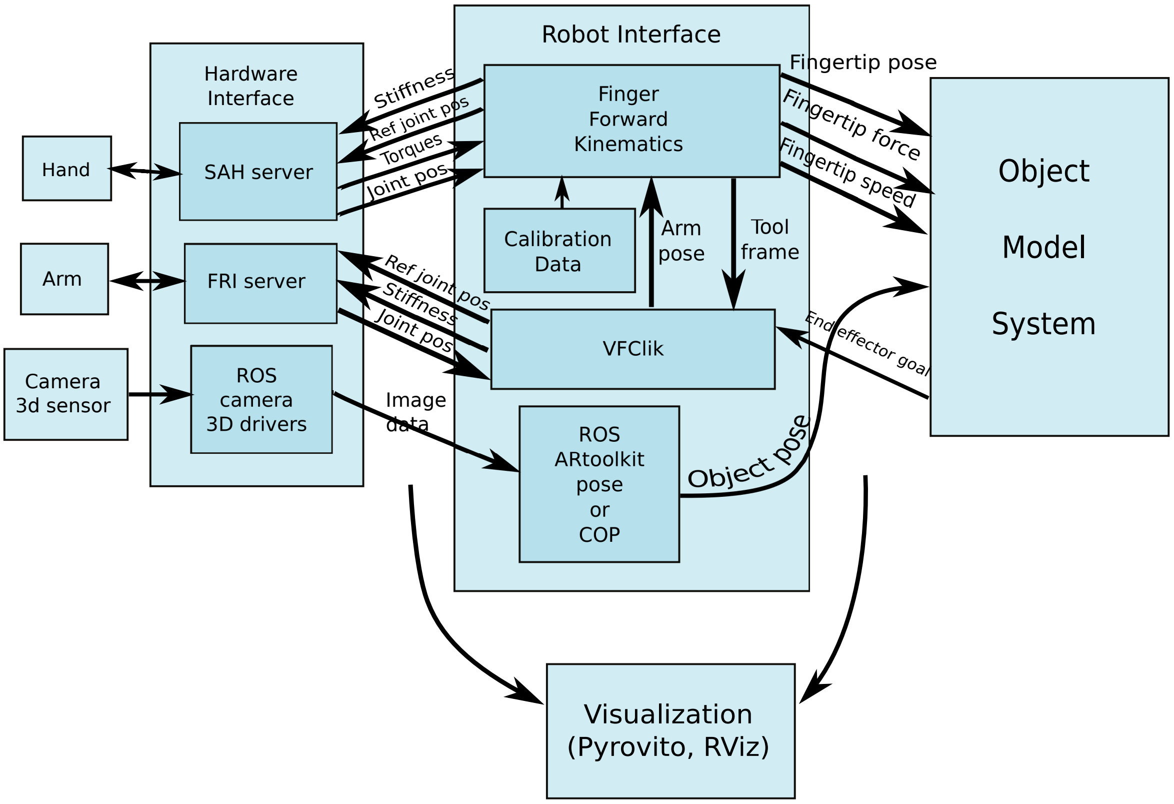


FIGURE A.7. Software architecture.

tion from the ARToolkit module.

Because our proof-of-concept example is, at this time, limited to only one type of object (boxes), the selection between models was not necessary to implement yet. As soon as we incorporate more models, the association map would select the particular model based on the object to be manipulated.

Two control loops can be used in this architecture: 1) Using the finger tip force and velocity together with joint position commands, 2) using the object pose extracted from the ARToolkit module together with joint position commands. The Object Model System (OMS) will act in these cases as the controller.

We also provide a visualization tool, PYROVITO (Python robot visualization tool) [I.A.S and ARCOS-Lab 2013], to show the current state of the robot and the object. (An example is shown in figure A.21).

A.2.1 Robotics middlewares

For the interconnection of the different modules in this work we used two different robotics middlewares: YARP [Metta, Fitzpatrick, and Natale 2006] and ROS [Quigley et al. 2009]. YARP was used during the development of the VFclik, SAH server and torque/force interface modules. For the visual perception we used the ARToolkit [Kato 2002] ROS module. The box model and association map modules used YARP for communication. We plan to transform the system to be as middleware agnostic as possible, maybe by converting all our system blocks into libraries or python modules. Then, leaving to the user the freedom to choose the middleware that better fits his needs.

A.2.2 Arm navigation

Our arm navigation system consists of a vector field based close loop inverse kinematics (VF-clik) and a joint position controller. Both systems output joint speeds to the low level interfaces. The joint position controller is mainly used to initialize the arms to known working configurations. The VFclik is used in the most interesting situations; the VFclik reaches end-effector cartesian positions which are usually manipulation goals.

A.2.2.1 Obstacle avoidance and goal attraction

Our VFclik controller makes use of the idea of vector fields to move or direct arm motion constantly. But before we continue describing how this system works we will explain why we decided to use such system.

Our impedance control arms have the advantage that they can be soft or compliant with the environment. Typically, when you have arms that don't have impedance control, a classical approach that works fine for moving a robot arm is to use trajectories; you define intermediate points in a straight line between the current position and your desired goal. For each intermediate point you find your position inverse kinematics and move the motors in a straight line to this joint position. If you put enough intermediate points, you will get a nice straight motion of the end-effector in cartesian coordinates. All this is fine except when we have obstacles and worse if they move or when the robot is compliant and the desired trajectory may not always be followed.

If we are dealing with static obstacles, the robot may be able to generate a valid trajectory by using multiple straight lines, and then, as before, define intermediate goal points in them before starting any arm motion. If the obstacles are many and you are working in a 3D world (as it is with robot manipulators) this previous calculation may take some time. Also, the motion may not be so smooth because the end-effector will experience sudden movements each time there is a change from one line to another. This last problem can be solved be smoothing the corners of the movement into soft curves, but again, this takes even more previous calculation time. This may not be a problem if you have such time before motion starts, which we usually have.

But, if the obstacle moves, we get more problems. In this case, we could generate a new trajectory, as before, every time an obstacle changes position. The problem with this is that the robot may need a moment to stop to perform this calculation. Or if the robot continues motion based on the old trajectory while calculating the new one, due to this calculation time, the robot may respond late. This can be aggravated if the obstacle moves continuously.

If we have an impedance robot arm we get yet another problem, if a person or object stops (holds) the arm during the trajectory execution, and then release the arm, it will suddenly jump very fast to get to the point where it should be at this time according to the trajectory (trajectories are position vs time). This is a very undesired behavior.

Also possible is that the human moves the robot arm suddenly to another position and then releases the arm there. In this case, should the trajectory be generated again? Probably yes, but this requires to detect a special case (human stopping and moving arm to another position) during motion, complicating the logic of the system.

Manipulator motion control using vector fields of attractors and repellers can alleviate these problems [Khatib 1986a] [Khatib 1986b]. Some advantages vector fields provide are:

1. The vector field calculates the current vector according to the current end-effector position. Therefore, if something pushes the arm or end-effector to a new position, the vector field will immediately give a new vector to follow, therefore reacting fast to new situations.

2. We can avoid obstacles, moving or not. In a vector field we add repellers; as soon as the end-effector gets near to an obstacle the repeller field pushes the end-effector away. This happens immediately even if the obstacle constantly moves.

3. Different types of attractor or repellers can be used to change motion behavior.

4. It is a close loop system that constantly monitors current position and corrects (by attraction) to get to the goal.

Therefore, it is a system that fits very well our type of robot arm.

Its main disadvantage is the local minima that usually appears when two or more obstacles are too near to each other.

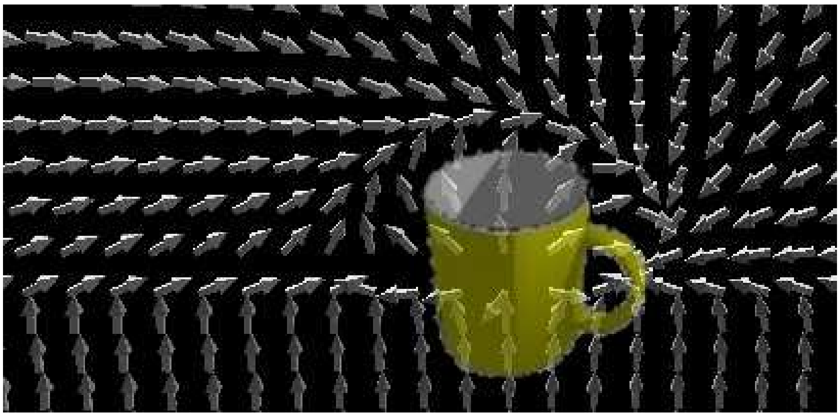


FIGURE A.8. Vector Field for grasping a cup from the handle.

In figure A.8, a vector field near a mug to be grasped with our B21 robot is shown. In the example, the system, apart from reaching the mug handle, it also avoids the obstacle that the same cup represents. It guides the gripper with a precise movement very near to the handle, giving a behavior that produces a reaching approach that avoids crashing against the handle and the cup.

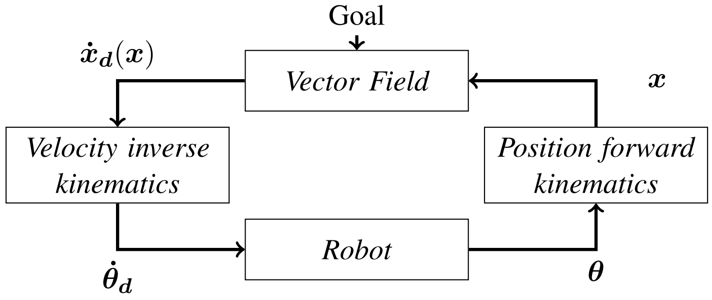

Goal
$\dot{\boldsymbol{x}}_{\boldsymbol{d}}(\boldsymbol{x})$
Vector Field
$\boldsymbol{x}$
Velocity inverse kinematics
Position forward kinematics
Robot
$\dot{\boldsymbol{\theta}}_{\boldsymbol{d}}$
$\boldsymbol{\theta}$

FIGURE A.9. Simplified block diagram of the vector field system. $\boldsymbol{x}_d$, $\dot{\boldsymbol{x}}_d(\boldsymbol{x})$ and $\boldsymbol{x}$ represent the desired intermediate point, the velocity vector and the current position in work space. $\dot{\boldsymbol{\theta}}_d$ and $\boldsymbol{\theta}$ correspond to the velocity and angular vectors in joint space.

The vector field used in the current system works as follows (refer to figure A.9). It takes the current end-effector cartesian position given by a position forward kinematics and using information about obstacles and goals in task space (object space) it calculates a task space velocity vector which is then translated to joint speeds by a velocity inverse kinematics module. Instead of calculating a vector field grid and then querying the current value from the grid, it calculates continuously (using point attractors and point repellers) the corresponding velocity vector for every control cycle. In effect a vector field is a dynamical system represented with a differential equation. For example, for a simple point attractor the differential equation describing a vector field (and therefore the feedback motion controller) is:

$$\dot{x} = (x_{ref} - x) * P \tag{A.1}$$

But this attractor grows without bound with the distance to the center of the attractor. Something better would saturate the magnitude of the vector to some maximum speed:

$$\dot{x} = sat(x_{ref} - x, d_{max}) * P \tag{A.2a}$$

$$sat(x, y) = \begin{cases} x, & \text{if } x < y \\ y, & \text{if } x \geq y \end{cases} \tag{A.2b}$$

We can continue in a similar manner to generate more vector fields depending on our application. We can also add the right hand side of several vector fields to generate a new vector field. This can be used for example, to add repellers and attractors to avoid obstacles and to move towards a goal.

The point attractor and repeller vectors are shaped depending on the role that each object has. For instance, cups that are to be grasped are composed of two repellers and two attractors, the first repeller is a sphere (located on the center of the cup) that decreases the force influence with an exponential decay as the distance to the current position increases, the second repeller is a *shadow* repeller pushing away from the back side of the cup (this gives smoother movements or more straight line motions), one attractor is a normal spherical attractor (centered on the cup handle) with no decay thus having a global effect of attraction, and the other attractor is a conical attractor with the tip of the cone in the handle of the cup and the cone increasing its diameter away from the cup, also the force of this attractor decays with the distance (Figure A.8).

Various vector fields commonly used are shown in figure A.10.

Obstacles are represented as spherical repellers with a certain radius of action and an extra radius distance for taking into account imprecisions in the arm movement tracking and joint position sensing and also in the visual object detection.

We ran this system in the B21, iCub and TUM-Rosie robots. When executing using this system with our B21 robot, Player Project (Player Gerkey, Vaughan, and Howard 2003) was used for the low-level control of the Amtec Powercube manipulator. With the iCub and TUM-Rosie we use our self-made modules to communicate to the arms. For communication between the modules we are using YARP (YARP Metta, Fitzpatrick, and Natale 2006).

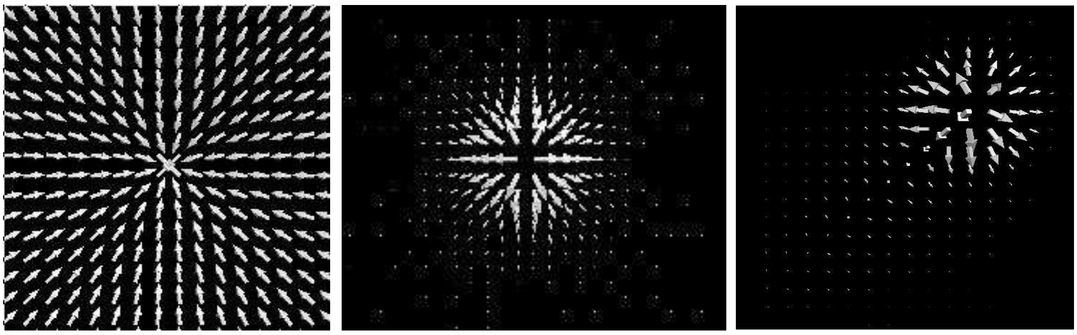


FIGURE A.10. Various Vector Fields

Since TUM-Rosie and the iCub have 7-DOF arms we can use the null space to avoid joint limits. Our VFclik system integrates a solution for avoiding the joint limits of the arms but still tries a good possible inverse kinematic solution: For all the joints, the distance of the joint position to the joint limit is measured constantly. As this distance becomes smaller (we are getting near to the joint limit), the system uses this joint less. This can be done because for the velocity inverse kinematics a weighted damped least squares inverse kinematics algorithm is used [Smits et al. 2008]. By weighting less that particular joint, the inverse kinematics uses that joint less to achieve the inverse kinematics. As the joint position gets very near to the limit, the joint is practically not used, therefore, avoiding total joint saturation and providing a joint limit aware velocity solution locally. We change the weighting factor from 1 to 0 according to the profile shown in figure A.11.

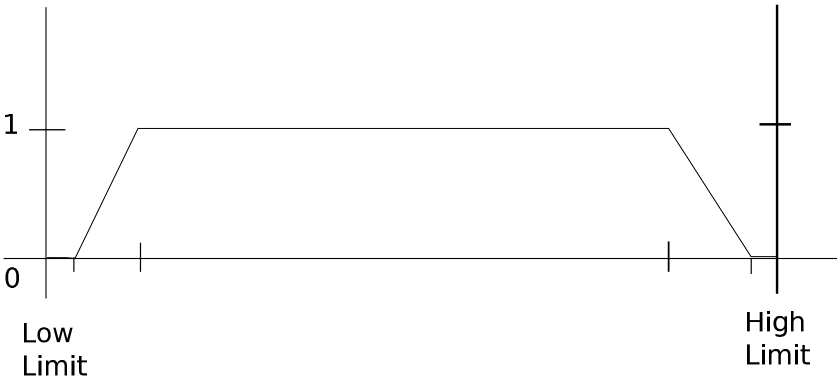


FIGURE A.11. Joint limit profile.

This system is used by our Object Model System (OMS) to approach (reach) the object and then during pushing. The VFclik system takes into account an additional homogeneous transformation that we call "the tool frame" as part of the end-effector. The Object Model System (OMS) sets the tool frame to the current finger tip position with respect to the end of the arm (Fig. A.12).

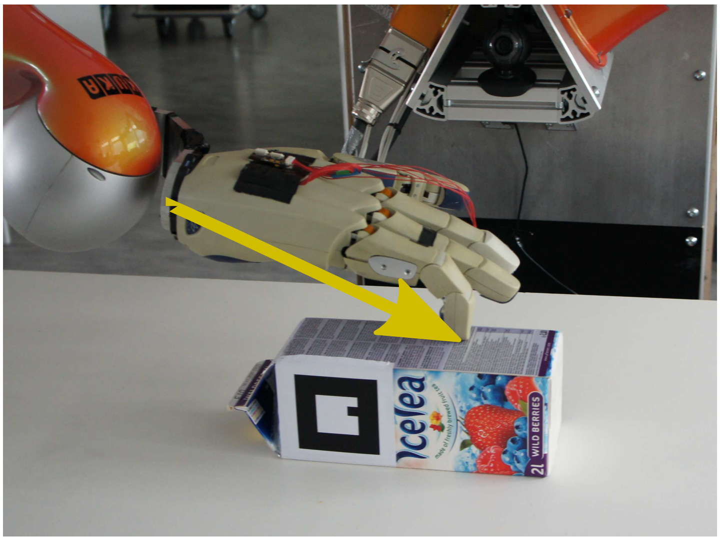


FIGURE A.12. End-effector Toolframe

A.2.2.2 Joint position controller and initialization

A joint position controller is used to position the arm to known *sane* initial configurations. The arms start in a raised position to provide the camera with a clear view of the object (Fig. A.13). Then the system changes to the VFclik controller to go in a straight line towards a point a bit far from the initial pushing point of contact (see section 5.2.1). To then approach the object in a straight line until it is touched (Fig. A.14).

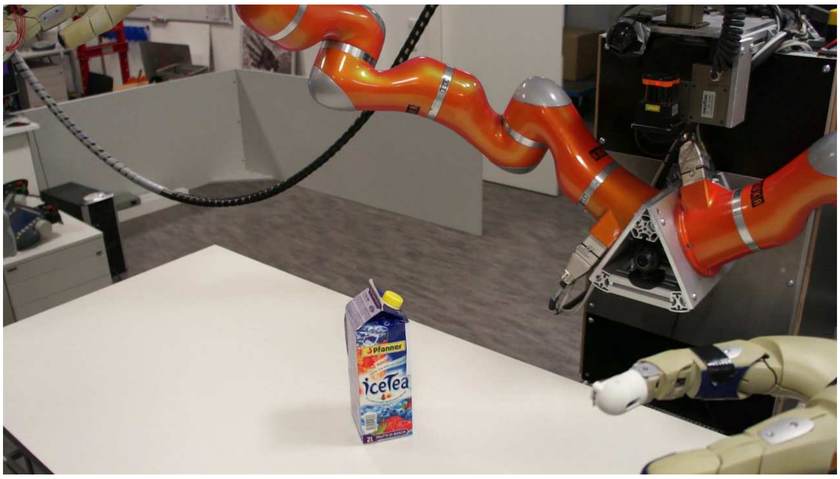


FIGURE A.13. Arms raised to have an open view of the object.

A.2.2.3 Demonstrations

Several demonstrations were ran using our VFclik system.

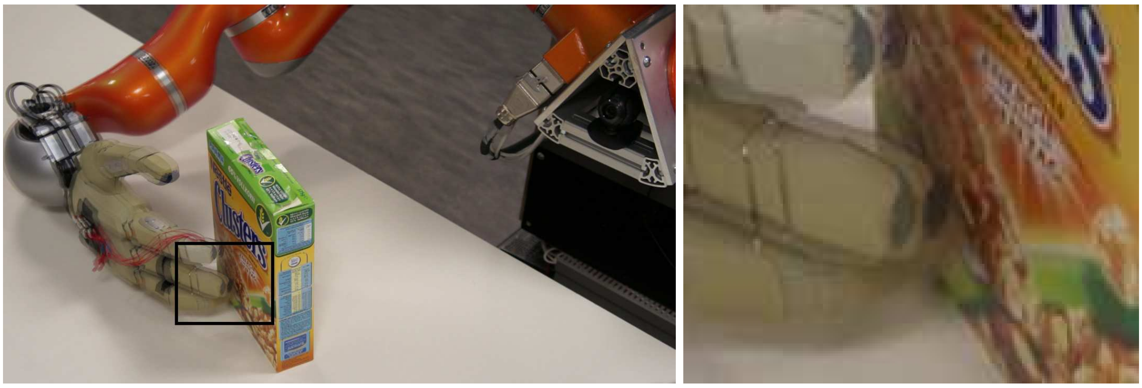


FIGURE A.14. Finger approaching.

Demonstration on B21 robot

This demonstration consisted on detecting if an object was located on a kitchen table using RFID readers under the table. When the object was detected the robot navigated using wavefront and adaptive monte carlo localization in front of the table (From the Player Project [The-Player-Project 2013]). There, the robot looked at the table and used a CAD model based perception system [Klank et al. 2009] to detect the object pose. The object in this demonstration was a coffee mug. This object pose was use in the demonstration to set the mug as an obstacle and the handle of the mug as an attractor in our VFclik system [Beetz et al. 2010]. The system was able to run the whole day during the *day of open doors* in the Technische Universität München. Some pictures of the activity and its results are shown in Fig. A.15.

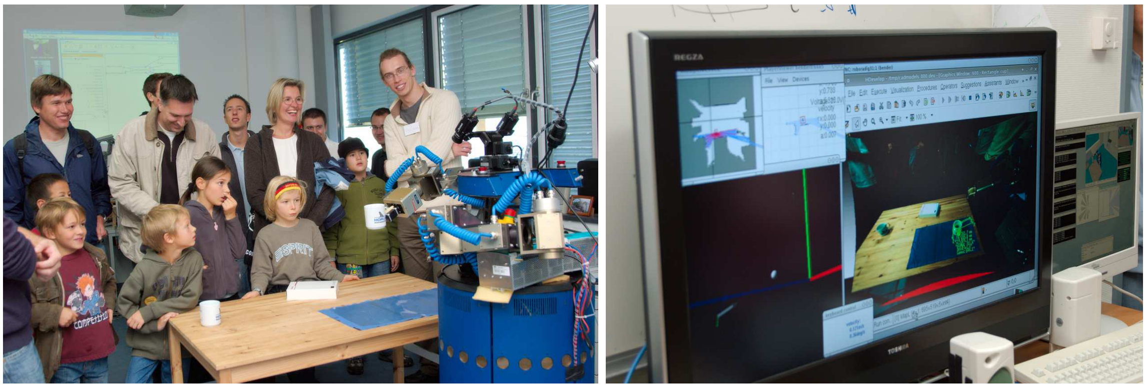


FIGURE A.15. Day of open doors at TUM.

Lego® building with the iCub

The iCub looks at a small table searching for a couple of Lego® pieces using ARToolkit markers. Once found, it picks them up. Then it looks at the Lego® pieces in the hands to see if the markers are still visible, if not, then the robot drops the pieces on the table and tries to pick them up once more. To check if the grasping of the Lego® piece is correct a Support

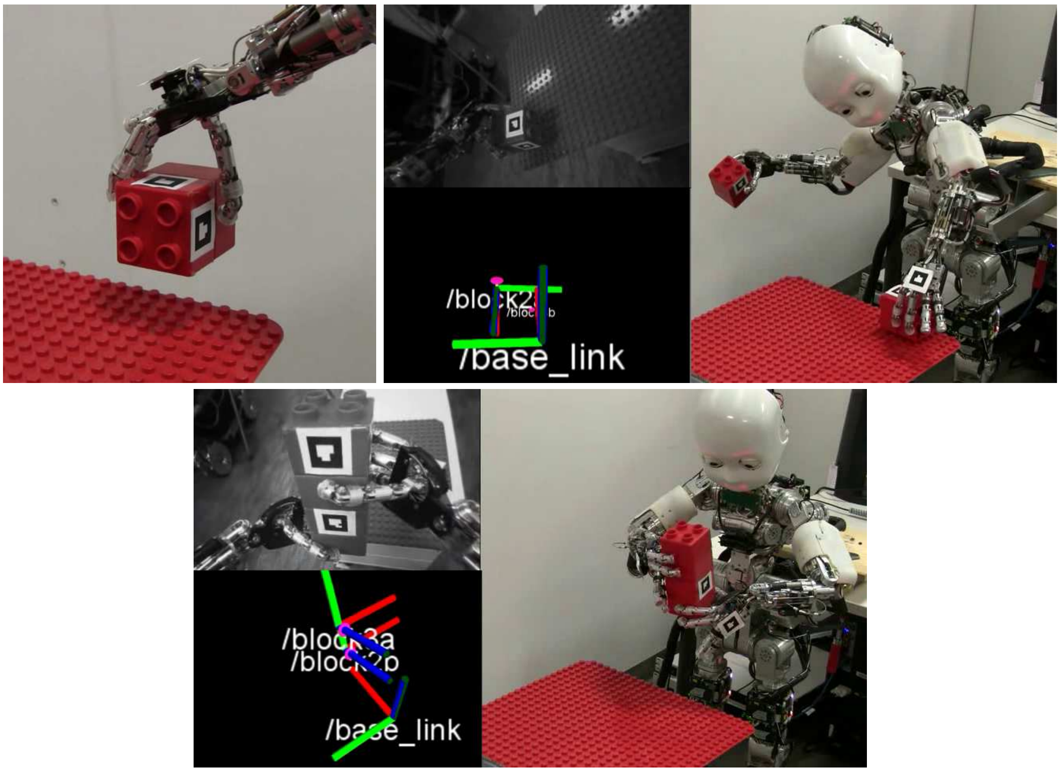


FIGURE A.16. iCub Lego® building.

Vector Machine is used to learn if the grasp worked or not and if it used two or three fingers. For this the SVM uses the PID error values from the motor controllers of the finger joints (the robot didn't have any other type of finger sensors). This proved to be very reliable. Then the robot proceeds to align and approach the two lego® pieces (while constantly recalculating the tool frame of the VFclik, which corresponds to the vector between the lego® piece and the arm end-effector). This align and approach part was guided by a human operator. The pictures in figure A.16 show the results. The reaching, and approaching motions were all done using the VFclik system.

Memoman imitation

In this system, the iCub robot executed joint position trajectories acquired from a marker-less full body motion tracker [Bandouch, Jenkins, and Beetz 2012] (Fig. A.17), using the joint position controller described in section A.2.2.2.

Human motion transfer using optimization principles

From a marker-less full body motion tracker [Bandouch, Jenkins, and Beetz 2012], human arm reaching trajectories were separated and its optimality principles extracted. Then this optimality principles were applied to an optimal controller based on the iCub kinematics and dynamics. This optimal motion trajectories were executed on the iCub using a trajectory gen-

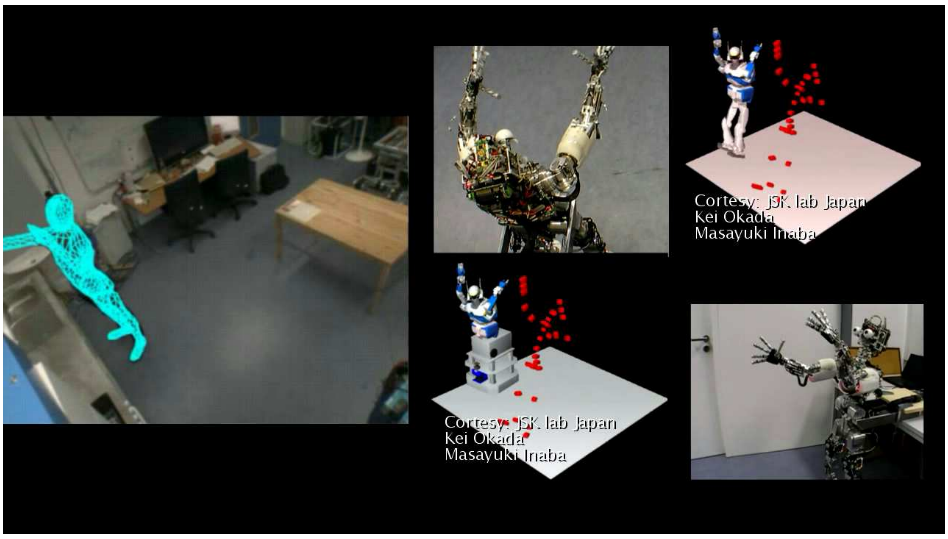


FIGURE A.17. iCub imitation.

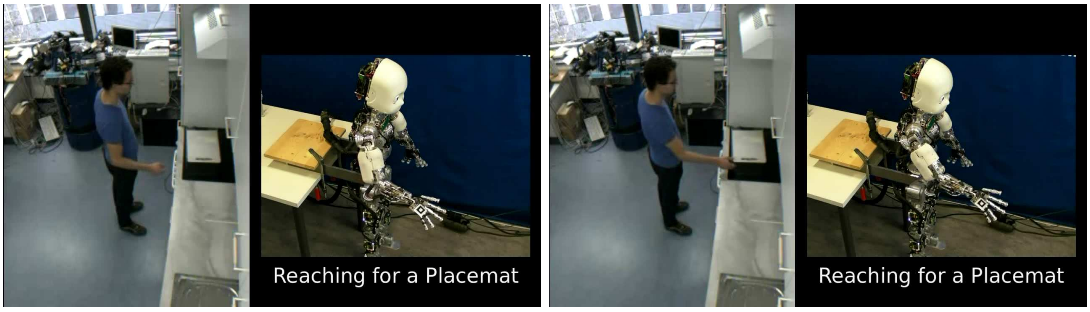


FIGURE A.18. iCub optimizations transfer.

erator connected to the VFclik system [Albrecht et al. 2011]. Some results from this system can be appreciated in figure A.18.

Demonstration on TUM-Rosie robot

Several demonstrations were done with our TUM-Rosie robot using the VFClik system. Pancake making, Bavarian breakfast and Sandwich making [Beetz et al. 2011] are some examples of demonstrations using the VFClik system (Fig. A.20). VFClik was use mainly to produce straight cartesian movements. The VFClik obstacle avoidance system was tested by [Maldonado, Klank, and Beetz 2010] (Fig. A.19).

A.2.3 Perception

The perception in our system consisted of finger tip forces, finger tip velocities and object absolute position measurements.

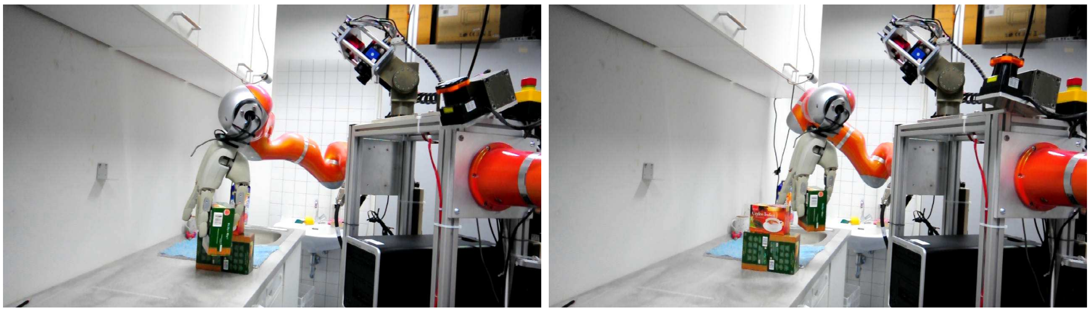


FIGURE A.19. TUM-Rosie avoiding obstacles.

FIGURE A.20. TUM-Rosie demonstrations.

A.2.3.1 Finger tip force estimation

Since our DLR/HIT I hands don't have finger tip force sensors but joint torque sensors, we had to estimate the finger tip force sensed using the torque measurements.

One important practical detail with this force estimation is that the torque sensors on fingers of the DLR-HIT hand were not really on the joints exactly, but slightly offset from the joint. To calculate the force on the finger tip the Jacobian is necessary, but this jacobian is a different one for the velocity inverse kinematics than for the force forward kinematics because of the offset. Two different kinematic chains were necessary to account for this. We use the Orocos-kdl [Smits et al. 2008] library for forward and inverse kinematics. We created a long kinematic chain containing a joint for each mechanical joint and each torque sensors, when using the kinematic chain we "locked" the joints that were not necessary: we "locked" the joints corresponding to torque sensors when doing velocity kinematics, and we "locked" the motor joints when doing force/torque kinematics. Also, orocos-kdl lacked the possibility to express coupled joints in the kinematic chain. A *coupled joint* is a joint that its movement depends on the movement of another joint. For expressing the relationship between the joints a *coupling matrix* is used. This modified version of orocos-kdl can be downloaded from https://github.com/arcoslab/orocos-kdl.

The torque sensors on the fingers were totally uncalibrated, therefore the finger tip force readings were not useful. We designed a procedure to calibrate each sensor one by one: We attach a calibrated weight to the finger tip, then we command the robot hand in three different orientations, each of them such that the estimated force only depend on that sensor. Then we find the corresponding multiplication factor for each sensor such that the finger tip force matched the calibrated weight.

A.2.3.2 Object absolute position estimation

A marker tracking system was used to detect the objects position, in particular we use the ARToolkit implementation running in ROS: ar pose. IT should be easy to interchange this system with any other system that can output object poses like the ones described by [Klank et al. 2009], [Beetz et al. 2009] and [Marton et al. 2010].

A.2.3.3 Simulator

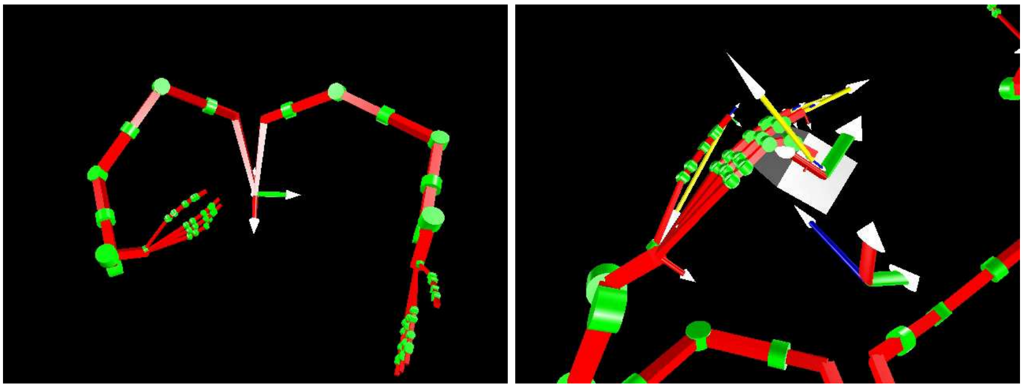


FIGURE A.21. TUM-Rosie object sliding simulator.

To facilitate development and testing we develop a robot and object simulator [Ruiz-Ugalde, IAS-TUM, and ARCOS-Lab 2013a]. The simulator is designed to be as transparent as possible to the Object Model System (OMS) and to the robot interface modules. The simulator basically exchanges the real robot (TUM-Rosie) for an emulation at the low level hardware interfaces (see figure A.7). There it emulates impedance behavior in all the joints, and it also uses the same Object Model (OM) to simulate object behavior. A visualization of the simulator running is shown in figure A.21.

The simulator proved to behave quiet similar to the real robot, and we expect to use it to add more Object Models (OMs) in the future. It is available to the general public as free/open-source software (GPLv3). Instruction for installation and usage can be found in the following webpage:

https://wiki.arcoslab.eie.ucr.ac.cr/doku.php?id=object_manipulation_robot_simulator.

Bibliography

[1] Sebastian Albrecht et al. “Imitating human reaching motions using physically inspired optimization principles”. In: *11th IEEE-RAS International Conference on Humanoid Robots*. Bled, Slovenia, 2011.

[2] Alin Albu-Schäffer et al. “The DLR lightweight robot: design and control concepts for robots in human environments”. In: *Industrial Robot: An International Journal* 34.5 (2007), pp. 376–385.

[3] Harold Alexander and Hemanshu Lakhani. “Robotic control of sliding object motion and orientation”. In: *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*. Vol. 4. IEEE. 1996, pp. 3336–3342.

[4] Christophe Andrieu, Arnaud Doucet, and Vladislav B Tadic. “On-line parameter estimation in general state-space models”. In: *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on*. IEEE. 2005, pp. 332–337.

[5] Jan Bandouch, Odest Chadwicke Jenkins, and Michael Beetz. “A Self-Training Approach for Visual Tracking and Recognition of Complex Human Activity Patterns”. In: *International Journal of Computer Vision* 99.2 (2012), pp. 166–189.

[6] Jerome Barraquand and Jean-Claude Latombe. “Robot motion planning: A distributed representation approach”. In: *The International Journal of Robotics Research* 10.6 (1991), pp. 628–649.

[7] Michael Beetz. *Plan-based Control of Robotic Agents*. Vol. LNAI 2554. Lecture Notes in Artificial Intelligence. Springer Publishers, 2002.

[8] Michael Beetz. “Towards Comprehensive Computational Models for Plan-Based Control of Autonomous Robots”. In: *Mechanizing Mathematical Reasoning: Essays in Honor of Jörg H. Siekmann on the Occasion of His 60th Birthday*. Ed. by Werner Stephan Dieter Hutter. Springer LNCS 2605, 2005, pp. 514–527.

[9] Michael Beetz, Martin Buss, and Dirk Wollherr. “Cognitive technical systems—what is the role of artificial intelligence?” In: *KI 2007: Advances in Artificial Intelligence*. Springer, 2007, pp. 19–42.

[10] Michael Beetz, Lorenz Mösenlechner, and Moritz Tenorth. “CRAM – A Cognitive Robot Abstract Machine for Everyday Manipulation in Human Environments”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. Taipei, Taiwan, 2010, pp. 1012–1017.

[11] Michael Beetz et al. “CoP-Man – Perception for Mobile Pick-and-Place in Human Living Environments”. In: *Proceedings of the 22nd IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) Workshop on Semantic Perception for Mobile Manipulation*. Invited paper. St. Louis, MO, USA, 2009.

[12] Michael Beetz et al. “Generality and Legibility in Mobile Manipulation”. In: *Autonomous Robots Journal (Special Issue on Mobile Manipulation)* 28.1 (2010), pp. 21–44.

[13] Michael Beetz et al. “Robotic Roommates Making Pancakes”. In: *11th IEEE-RAS International Conference on Humanoid Robots*. Bled, Slovenia, 2011.

[14] Michael Beetz et al. “The Assistive Kitchen – A Demonstration Scenario for Cognitive Technical Systems”. In: *IEEE 17th International Symposium on Robot and Human Interactive Communication (RO-MAN), Muenchen, Germany*. Invited paper. 2008, pp. 1–8.

[15] Richard Bellman. *Adaptive control processes: a guided tour*. Vol. 4. Princeton university press Princeton, 1961.

[16] J.D. Bernheisel and K.M. Lynch. “Stable transport of assemblies by pushing”. In: *Robotics, IEEE Transactions on* 22.4 (2006), pp. 740 –750.

[17] J.D. Bernheisel and K.M. Lynch. “Stable transport of assemblies by pushing”. In: *Robotics, IEEE Transactions on* 22.4 (2006), pp. 740–750. ISSN: 1552-3098. DOI: 10.1109/TRO.2006.875488.

[18] Martin Buss, Hideki Hashimoto, and John B Moore. “Dextrous hand grasping force optimization”. In: *Robotics and Automation, IEEE Transactions on* 12.3 (1996), pp. 406–418.

[19] Martin Buss and Thomas Schlegl. “Multi-fingered regrasping using on-line grasping force optimization”. In: *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*. Vol. 2. IEEE. 1997, pp. 998–1003.

[20] Lillian Y. Chang, Siddhartha Srinivasa, and Nancy Pollard. “Planning pre-grasp manipulation for transport tasks”. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2010.

[21] Lillian Y Chang, Garth J Zeglin, and Nancy S Pollard. “Preparatory object rotation as a human-inspired grasping strategy”. In: *Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on*. IEEE. 2008, pp. 527–534.

[22] Lillian Y. Chang et al. “Preparatory object rotation as a human-inspired grasping strategy”. In: *International Conference on Humanoid Robots*. 2008.

[23] Silvia Coradeschi, Amy Loutfi, and Britta Wrede. “A short review of symbol grounding in robotic and intelligent systems”. In: *KI-Künstliche Intelligenz* 27.2 (2013), pp. 129–136.

[24] CA Coulomb. “Mémoires de Mathématiques et de Physique de l’Académie Royale des Sciences”. In: *Mémoires de Mathématiques et de Physique de l’Académie Royale des Sciences* (1785).

[25] Alessandro De Luca, Giuseppe Oriolo, and Claude Samson. “Feedback control of a nonholonomic car-like robot”. In: *Robot motion planning and control*. Springer, 1998, pp. 171–253.

[26] C Canudas De Wit and OJ Sordalen. “Exponential stabilization of mobile robots with nonholonomic constraints”. In: *Automatic Control, IEEE Transactions on* 37.11 (1992), pp. 1791–1797.

[27] Mehmet Dogar and Siddhartha Srinivasa. “Push-Grasping with Dexterous Hands: Mechanics and a Method”. In: *Proceedings of 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010)*. 2010.

[28] Mehmet Remzi Dogar and Siddhartha S Srinivasa. “Push-grasping with dexterous hands: Mechanics and a method”. In: *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE. 2010, pp. 2123–2130.

[29] Pedro Domingos. “What’s missing in AI: The interface layer”. In: *Artificial Intelligence: The First Hundred Years. AAAI Press, to appear* (2006).

[30] Brian R Duffy and Gina Joue. “Intelligent robots: The question of embodiment”. In: *Proc. of the Brain-Machine Workshop*. 2000.

[31] Andreas Fedrizzi et al. “Transformational planning for mobile manipulation based on action-related places”. In: *Advanced Robotics, 2009. ICAR 2009. International Conference on*. IEEE. 2009, pp. 1–8.

[32] Paul Fitzpatrick et al. “Learning About Objects Through Action - Initial Steps Towards Artificial Cognition”. In: *In Proceedings of the 2003 IEEE International Conference on Robotics and Automation (ICRA*. 2003, pp. 3140–3145.

[33] Matteo Fumagalli et al. “Exploiting proximal F/T measurements for the iCub active compliance”. In: *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE. 2010, pp. 1870–1876.

[34] Brian Gerkey, Richard T. Vaughan, and Andrew Howard. “The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems”. In: *Proceedings of the 11th International Conference on Advanced Robotics (ICAR)*. 2003, pp. 317–323.

[35] James J. Gibson. *The Ecological approach to visual perception*. New edition. Lawrence Erlbaum Associates, Sept. 1986. ISBN: 0898599598.

[36] Simon F Giszter, Ferdinando A Mussa-Ivaldi, and Emilio Bizzi. “Convergent force fields organized in the frog’s spinal cord”. In: *The journal of neuroscience* 13.2 (1993), pp. 467–491.

[37] Georg A Gottwald and Ian Melbourne. “A new test for chaos in deterministic systems”. In: *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 460.2042 (2004), pp. 603–611.

[38] Suresh Goyal. “Planar sliding of a rigid body with dry friction: limit surfaces and dynamics of motion”. PhD thesis. Cornell University, 1989.

[39] Suresh Goyal, Andy Ruina, and Jim Papadopoulos. “Limit surface and moment function descriptions of planar sliding”. In: *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*. IEEE. 1989, pp. 794–799.

[40] Suresh Goyal, Andy Ruina, and Jim Papadopoulos. “Planar sliding with dry friction part 1. limit surface and moment function”. In: *Wear* 143.2 (1991), pp. 307–330.

[41] Gutemberg Guerra-Filho and Yiannis Aloimonos. “A language for human action”. In: *Computer* 40.5 (2007), pp. 42–51.

[42] Li Han and Jeffrey C Trinkle. “Dextrous manipulation by rolling and finger gaiting”. In: *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*. Vol. 1. IEEE. 1998, pp. 730–735.

[43] Li Han, Jeffrey C Trinkle, and Zexiang X Li. “Grasp analysis as linear matrix inequality problems”. In: *Robotics and Automation, IEEE Transactions on* 16.6 (2000), pp. 663–674.

[44] Li Han et al. “The planning and control of robot dextrous manipulation”. In: *Robotics and Automation, 2000. Proceedings. ICRA’00. IEEE International Conference on*. Vol. 1. IEEE. 2000, pp. 263–269.

[45] Stevan Harnad. “The symbol grounding problem”. In: *Physica D: Nonlinear Phenomena* 42.1 (1990), pp. 335–346.

[46] Masahiko Haruno, D Wolpert, and Mitsuo Kawato. “Mosaic model for sensorimotor learning and control”. In: *Neural computation* 13.10 (2001), pp. 2201–2220.

[47] Gerd Hirzinger et al. “On a new generation of torque controlled light-weight robots”. In: *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*. Vol. 4. IEEE. 2001, pp. 3356–3363.

[48] Mohammed Hjiaj et al. “On the modelling of complex anisotropic frictional contact laws”. In: *International journal of engineering science* 42.10 (2004), pp. 1013–1034.

[49] Robert D. Howe and Mark R. Cutkosky. “Practical Force-Motion Models for Sliding Manipulation”. In: *International Journal of Robotic Research* 15 (1996), pp. 557–572. DOI: 10.1177/027836499601500603.

[50] I.A.S and ARCOS-Lab. *Python robot visualization tool*. [Online; accessed 25-August-2013]. 2013. URL: https://github.com/arcoslab/pyrovito.

[51] Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. “Movement imitation with nonlinear dynamical systems in humanoid robots”. In: *Robotics and Automation, 2002. Proceedings. ICRA’02. IEEE International Conference on*. Vol. 2. IEEE. 2002, pp. 1398–1403.

[52] Tetsunari Inamura et al. “Embodied symbol emergence based on mimesis theory”. In: *The International Journal of Robotics Research* 23.4-5 (2004), pp. 363–377.

[53] Alberto Isidori. *Nonlinear Control Systems*. 3rd. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1995. ISBN: 3540199160.

[54] Naoto Iwahashi. “Robots that learn language: Developmental approach to human-machine conversations”. In: *Symbol Grounding and Beyond*. Springer, 2006, pp. 143–167.

[55] Odest Chadwicke Jenkins and Maja J Matarić. “Performance-derived behavior vocabularies: Data-driven acquisition of skills from motion”. In: *International Journal of Humanoid Robotics* 1.02 (2004), pp. 237–288.

[56] David Joyner. “OSCAS: maxima”. In: *ACM Communications in Computer Algebra* 40.3-4 (2006), pp. 108–111.

[57] Hirokazu Kato. “ARToolKit: library for Vision-Based augmented reality”. In: *IEICE, PRMU* (2002), pp. 79–86.

[58] Oussama Khatib. “Real-time obstacle avoidance for manipulators and mobile robots”. In: *The international journal of robotics research* 5.1 (1986), pp. 90–98.

[59] Oussama Khatib. “The potential field approach and operational space formulation in robot control”. In: *Adaptive and Learning Systems: Theory and Applications* (1986), pp. 367–377.

[60] Ulrich Klank. “Everyday Perception for Mobile Manipulation in Human Environments”. PhD thesis. Technische Universität München, 2012.

[61] Ulrich Klank et al. “Real-time CAD Model Matching for Mobile Manipulation and Grasping”. In: *9th IEEE-RAS International Conference on Humanoid Robots*. Paris, France, 2009, pp. 290–296.

[62] Ilya Kolmanovsky and N Harris McClamroch. “Developments in nonholonomic control problems”. In: *Control Systems, IEEE* 15.6 (1995), pp. 20–36.

[63] Ingo Kresse and Michael Beetz. “Movement-aware Action Control – Integrating Symbolic and Control-theoretic Action Execution”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. St. Paul, MN, USA, 2012.

[64] Peng Cheng Jonathan Fink Vijay Kumar. “Abstractions and algorithms for cooperative multiple robot planar manipulation”. In: *Robotics: science and systems IV* (2009), p. 143.

[65] Philipp Kügler. “An Approach to Online Parameter Estimation in Nonlinear Dynamical Systems”. In: (2003).

[66] Hemanshu Mansukhlal Lakhani. “Robotic control of sliding-object motion and orientation”. PhD thesis. Massachusetts Institute of Technology, 1990.

[67] Steven M LaValle and James J Kuffner Jr. “Rapidly-exploring random trees: Progress and prospects”. In: (2000).

[68] H Liu et al. “The modular multisensory DLR-HIT-Hand”. In: *Mechanism and Machine Theory* 42.5 (2007), pp. 612–625.

[69] Hong Liu et al. “Multisensory five-finger dexterous hand: The DLR/HIT Hand II”. In: *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*. IEEE. 2008, pp. 3692–3697.

[70] Kevin Lynch, Hitoshi Maekawa, and Kazuo Tanie. “Manipulation and active sensing by pushing using tactile feedback”. In: *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 1992, pp. 416–421.

[71] Kevin M Lynch and Matthew T Mason. “Dynamic nonprehensile manipulation: Controllability, planning, and experiments”. In: *The International Journal of Robotics Research* 18.1 (1999), pp. 64–92.

[72] Kevin M Lynch and Matthew T Mason. “Stable pushing: Mechanics, controllability, and planning”. In: *The International Journal of Robotics Research* 15.6 (1996), pp. 533–556.

[73] Jeremy Maitin-Shepard et al. “Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding”. In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE. 2010, pp. 2308–2315.

[74] Alexis Maldonado, Humberto Alvarez-Heredia, and Michael Beetz. “Improving robot manipulation through fingertip perception”. In: *IEEE International Conference on Intelligent Robots and Systems (IROS)*. Vilamoura, Algarve, Portugal, 2012.

[75] Alexis Maldonado, Ulrich Klank, and Michael Beetz. “Robotic grasping of unmodeled objects using time-of-flight range data and finger torque information”. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Taipei, Taiwan, 2010, pp. 2586–2591.

[76] Zoltan-Csaba Marton et al. “General 3D Modelling of Novel Objects from a Single View”. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Taipei, Taiwan, 2010.

[77] Matthew T Mason. *Mechanics of robotic manipulation*. MIT press, 2001.

[78] Matthew Thomas Mason. “Manipulator grasping and pushing operations”. In: (1982).

[79] F. Matsuno and J. Tsurusaki. “Chained form transformation algorithm for a class of 3-states and 2-inputs nonholonomic systems and attitude control of a space robot”. In: *Decision and Control, 1999. Proceedings of the 38th IEEE Conference on*. Vol. 3. 1999, 2126–2131 vol.3. DOI: 10.1109/CDC.1999.831234.

[80] Giorgio Metta, Paul Fitzpatrick, and Lorenzo Natale. “YARP: yet another robot platform”. In: *International Journal on Advanced Robotics Systems* 3.1 (2006), pp. 43–48.

[81] R Michalowski and Z Mroz. “Associated and non-associated sliding rules in contact friction problems”. In: *Archiwum Mechaniki Stosowanej* 30.3 (1978), pp. 259–276.

[82] Tom M Mitchell. “Machine learning. 1997”. In: *Burr Ridge, IL: McGraw Hill* 45 (1997).

[83] Hans Moravec. *Mind children*. Cambridge Univ Press, 1988.

[84] Lorenz Mösenlechner and Michael Beetz. “Parameterizing Actions to have the Appropriate Effects”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. San Francisco, CA, USA, 2011.

[85] Z Mróz and S Stupkiewicz. “An anisotropic friction and wear model”. In: *International journal of solids and structures* 31.8 (1994), pp. 1113–1131.

[86] Armin Müller, Alexandra Kirsch, and Michael Beetz. “Transformational Planning for Everyday Activity”. In: *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS’07)*. Providence, USA, 2007, pp. 248–255.

[87] Richard M Murray and S Shankar Sastry. “Steering nonholonomic systems in chained form”. In: *Decision and Control, 1991., Proceedings of the 30th IEEE Conference on*. IEEE. 1991, pp. 1121–1126.

[88] Richard M Murray and Sosale Shankara Sastry. “Nonholonomic motion planning: Steering using sinusoids”. In: *Automatic Control, IEEE Transactions on* 38.5 (1993), pp. 700–716.

[89] Donald A Norman. *The design of everyday things*. Basic books, 2002.

[90] D. Omrcen et al. “Autonomous Acquisition of Pushing Actions to Support Object Grasping with a Humanoid Robot”. In: *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*. 2009.

[91] Erhan Oztop, Mitsuo Kawato, and Michael A Arbib. “Mirror neurons: Functions, mechanisms and models”. In: *Neuroscience letters* (2012).

[92] Dae-Hyung Park et al. “Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields”. In: *Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on*. IEEE. 2008, pp. 91–98.

[93] Vijay Kumar Peng Cheng Jonathan Fink. “Abstractions and Algorithms for Cooperative Multiple Robot Planar Manipulation”. In: *Proceedings of Robotics: Science and Systems IV*. Zurich, Switzerland, 2008.

[94] Farzad Pourboghrat. “Exponential stabilization of nonholonomic mobile robots”. In: *Computers and Electrical Engineering* 28.5 (2002).

[95] Morgan Quigley et al. “ROS: an open-source Robot Operating System”. In: *ICRA workshop on open source software*. Vol. 3. 3.2. 2009.

[96] Marc Raibert et al. “Bigdog, the rough-terrain quadruped robot”. In: 2008.

[97] Federico Ruiz-Ugalde, IAS-TUM, and ARCOS-Lab. *CMOC: Compact Models for object Control*. [Online; accessed 22-August-2013]. 2013. URL: https://github.com/arcoslab/.

[98] Federico Ruiz-Ugalde, IAS-TUM, and ARCOS-Lab. *CMOC: Compact Models for object Control Simulator Installation Instructions*. [Online; accessed 22-August-2013]. 2013. URL: https://wiki.arcoslab.eie.ucr.ac.cr/doku.php?id=object_manipulation_robot_simulator.

[99] Giulio Sandini, Giorgio Metta, and David Vernon. “The icub cognitive humanoid robot: An open-system research platform for enactive cognition”. In: *50 years of artificial intelligence*. Springer, 2007, pp. 358–369.

[100] Stefan Schaal. “Dynamic movement primitives-a framework for motor control in humans and humanoid robotics”. In: *Adaptive Motion of Animals and Machines*. Springer, 2006, pp. 261–280.

[101] Stefan Schaal, Shinya Kotosaka, and Dagmar Sternad. “Nonlinear dynamical systems as movement primitives”. In: *IEEE International Conference on Humanoid Robotics*. 2000.

[102] Stefan Schaal et al. “Learning movement primitives”. In: *Robotics Research*. Springer, 2005, pp. 561–572.

[103] Maxim Shevtsov, Alexei Soupikov, and Alexander Kapustin. “Ray-triangle intersection algorithm for modern CPU architectures”. In: *Proceedings of GraphiCon*. Vol. 2007. 2007, pp. 33–39.

[104] Bruno Siciliano and Oussama Khatib, eds. *Springer Handbook of Robotics*. Berlin, Heidelberg: Springer, 2008. URL: http://dx.doi.org/10.1007/978-3-540-30301-5.

[105] Jean-Jacques E Slotine, Weiping Li, et al. *Applied nonlinear control*. Vol. 199. 1. Prentice hall New Jersey, 1991.

[106] Ruben Smits et al. "Orocos: A software framework for complex sensordriven robot tasks". In: *IEEE Robotics and Automation Magazine* 108 (2008).

[107] Luc Steels. "The symbol grounding problem has been solved. so what's next". In: *Symbols and embodiment: Debates on meaning and cognition* (2008), pp. 223–244.

[108] Freek Stulp, Andreas Fedrizzi, and Michael Beetz. "Action-related place-based mobile manipulation". In: *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*. IEEE. 2009, pp. 3115–3120.

[109] Freek Stulp et al. "Learning and Reasoning with Action-Related Places for Robust Mobile Manipulation". In: *Journal of Artificial Intelligence Research (JAIR)* 43 (2012), pp. 1–42.

[110] Komei Sugiura and Naoto Iwahashi. "Learning object-manipulation verbs for human-robot communication". In: *Proceedings of the 2007 workshop on Multimodal interfaces in semantic interaction*. ACM. 2007, pp. 32–38.

[111] Moritz Tenorth and Michael Beetz. "KnowRob – A Knowledge Processing Infrastructure for Cognition-enabled Robots. Part 1: The KnowRob System". In: *International Journal of Robotics Research (IJRR)* (2013). Accepted for publication.

[112] Moritz Tenorth and Michael Beetz. "Priming Transformational Planning with Observations of Human Activities". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Anchorage, AK, USA, 2010, pp. 1499–1504.

[113] The-Player-Project. *The Player Project Web Page*. [Online; accessed 25-August-2013]. 2013. URL: http://playerstage.sourceforge.net/.

[114] Eric Walter and Luc Pronzato. "Identification of parametric models". In: *Communications and Control Engineering* (1997).

[115] John T Wen. "Control of nonholonomic systems". In: *The Control Handbook* (1996), pp. 1359–1368.

[116] Alan M Wing and J Randall Flanagan. "Anticipating dynamic loads in handling objects". In: *Proceedings of the ASME dynamic systems and control division*. Vol. 64. American Society of Mechanical Engineers. 1998, pp. 139–143.

[117] Daniel M Wolpert and Mitsuo Kawato. "Multiple paired forward and inverse models for motor control". In: *Neural Networks* 11.7 (1998), pp. 1317–1329.

[118] Florentin Wörgötter et al. “Cognitive agents - a procedural perspective relying on the predictability of Object-Action-Complexes (OACs)”. In: *Robotics and Autonomous Systems* 57.4 (2009), pp. 420–432.

[119] Fujio Yamaguchi and Masatoshi Niizeki. “Some basic geometric test conditions in terms of Plücker coordinates and Plücker coefficients”. In: *The Visual Computer* 13.1 (1997), pp. 29–41.

[120] Anna Yershova et al. “Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain”. In: *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE. 2005, pp. 3856–3861.

[121] K David Young, Vadim I Utkin, and Umit Ozguner. “A control engineer’s guide to sliding mode control”. In: *Variable Structure Systems, 1996. VSS’96. Proceedings., 1996 IEEE International Workshop on*. IEEE. 1996, pp. 1–14.

[122] Franziska Zacharias, Christoph Borst, and Gerd Hirzinger. “Capturing robot workspace structure: representing robot capabilities”. In: *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*. IEEE. 2007, pp. 3229–3236.

[123] Tom Ziemke. “Are robots embodied”. In: *First international workshop on epigenetic robotics Modeling Cognitive Development in Robotic Systems*. Vol. 85. Citeseer. 2001.