# Chapter 5

# Contact event prediction

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

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

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

## 5.1 Introduction

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

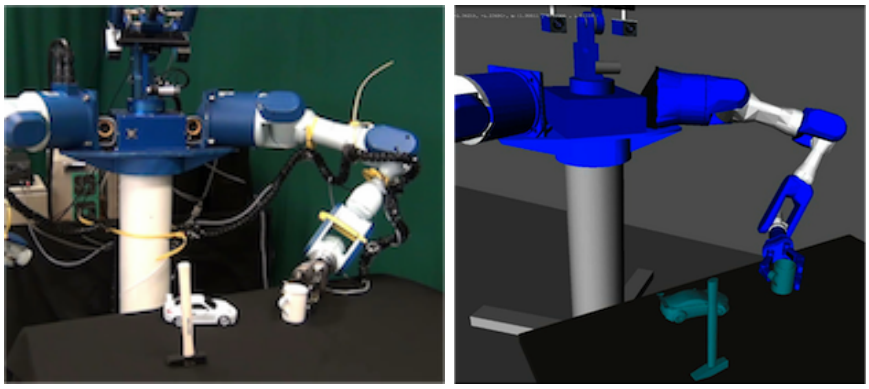


Figure 5.1: Left: Real robot. Right: Simulated robot (Tombatossals)

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

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

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

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

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

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

## 5.2 Related work

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

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

[1]Havok Physics: `http://www.havok.com/physics/`
[2]NVIDIA PhysX: `https://developer.nvidia.com/gameworks-physx-overview`

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

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

## 5.3 Components selection

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

### 5.3.1 Physics engine selection

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

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

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

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

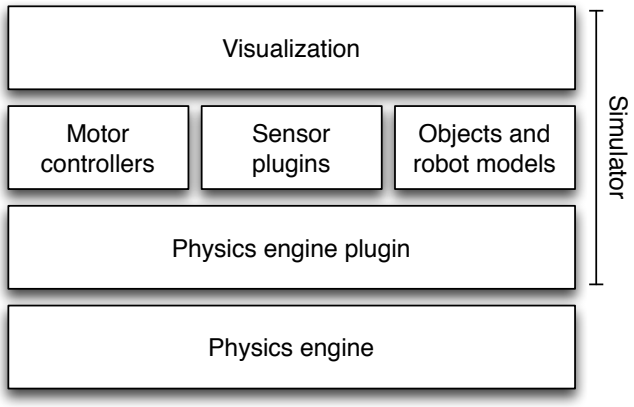



Figure 5.2: Components of the prediction engine composed mainly by a simulator and a physics engine.

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

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

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

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

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

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

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

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

### 5.3.2 Simulator selection

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

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

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

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

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

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

## 5.4 Robot implementation

### 5.4.1 Real robot setup

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

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

### 5.4.2 System architecture

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

### 5.4.3 Simulation and physics engine

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

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

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

#### Sensors

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

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

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

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

### Actuators

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

## 5.5 Experimental Setup

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

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

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

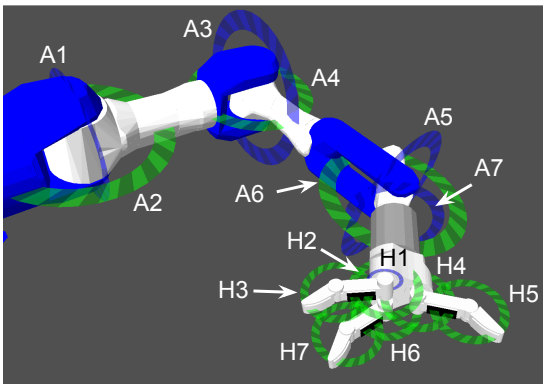



Figure 5.3: Arm (A) and hand (H) joints of the Tombatossals left arm.

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

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

### 5.5.1 Manipulation tasks

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

#### Grasp

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

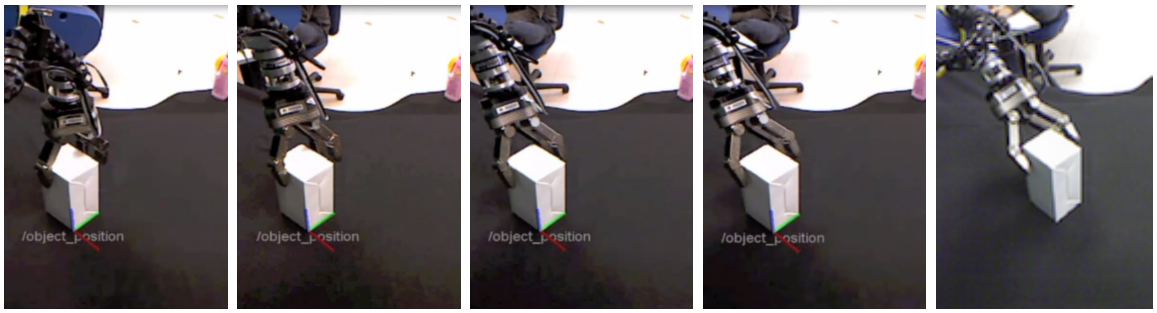



Figure 5.4: Grasping task starting positions. The object pose is acquired by a model based tracker and depicted as a reference frame.

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

## Slide

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

## In-hand manipulation

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

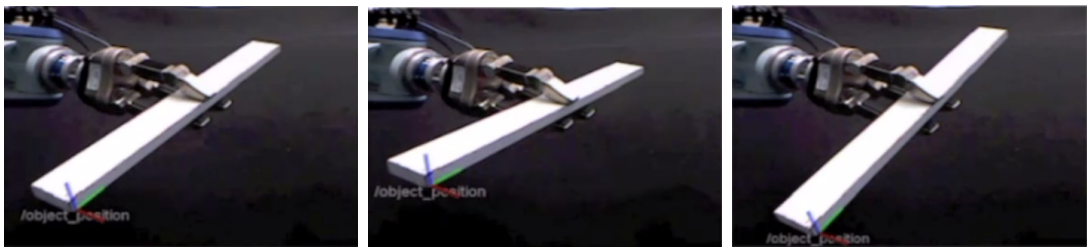


Figure 5.6: In-hand manipulation experiment. Center, swing left and swing right hand joint positions. The object pose is acquired by a model based tracker and depicted as a reference frame.

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

### 5.5.2 Metric definitions

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

#### Joint values

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

#### Tactile sensor readings

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

#### Object position

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

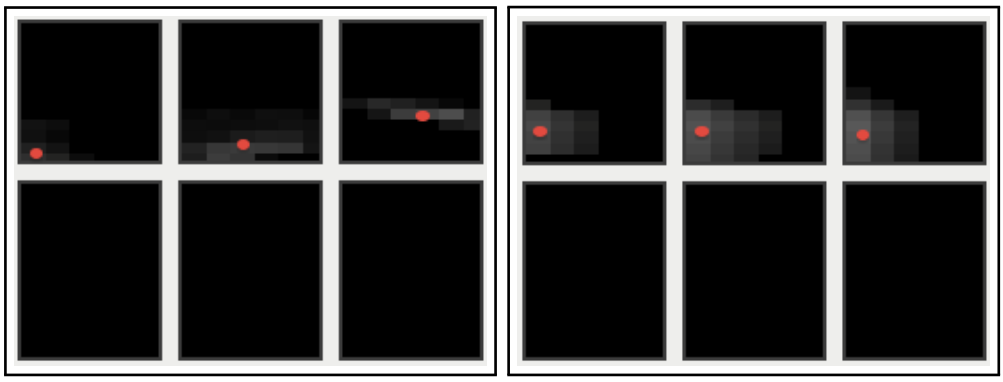


Figure 5.7: Tactile sensor data with centroids during the grasping task. Left: Real. Right: Simulated.

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

## 5.6 Results

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

### 5.6.1 Visual inspection

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

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

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

### 5.6.2 Detailed quantitative results

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

#### Grasp

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

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

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

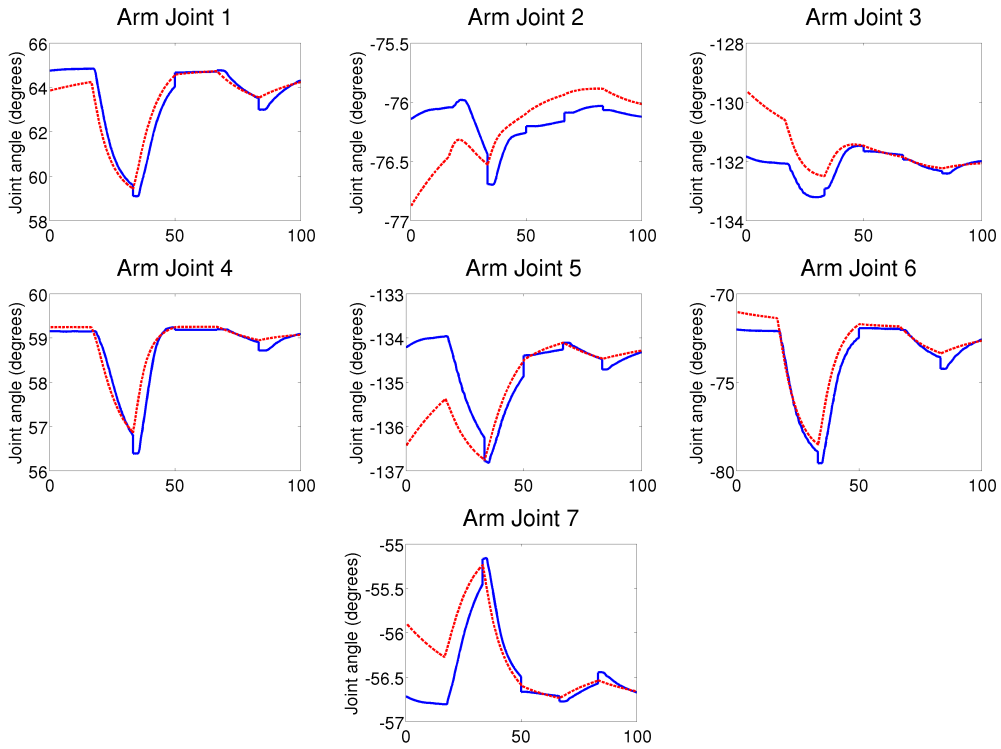



Figure 5.8: Arm joints positions for the first grasping experiment over time in percentage over the experiment duration. Blue lines indicate the readings from the robot and red dotted ones from the simulator.

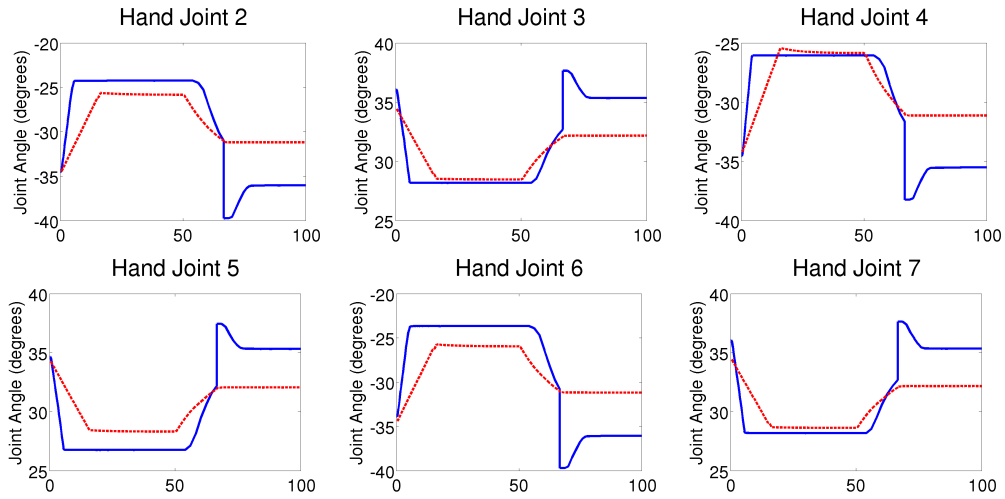



Figure 5.9: Hand joints positions for the first grasping experiment over time in percentage over the experiment duration. Blue lines indicate the readings from the robot and red dotted ones from the simulator. Hand joint 1 is omitted, it did not move and the error was constantly below 0.1 degrees.

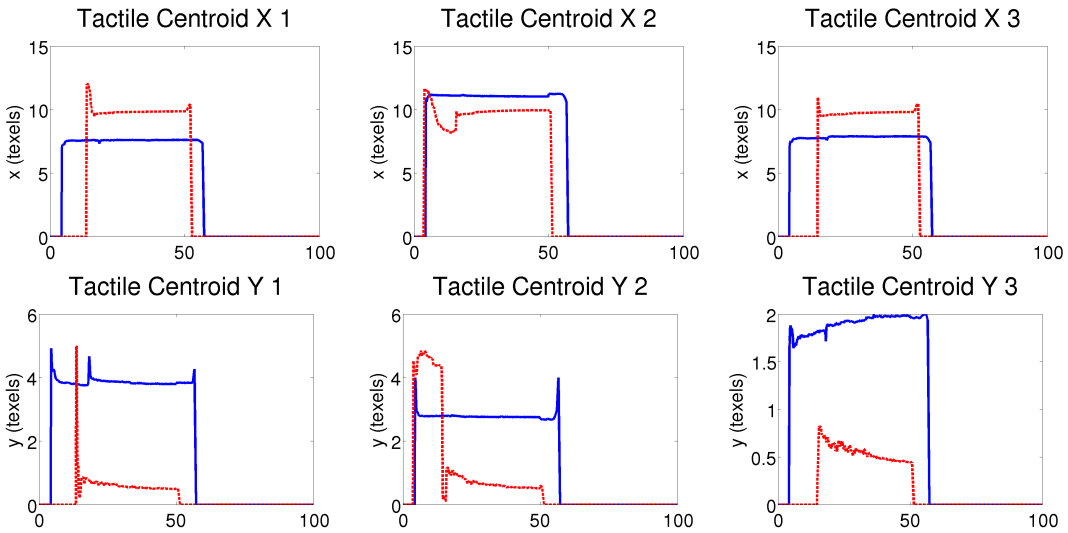



Figure 5.10: Tactile sensor results for the first grasping experiment over time in percentage over the experiment duration. Blue lines indicate the readings from the robot and red dotted ones from the simulator.

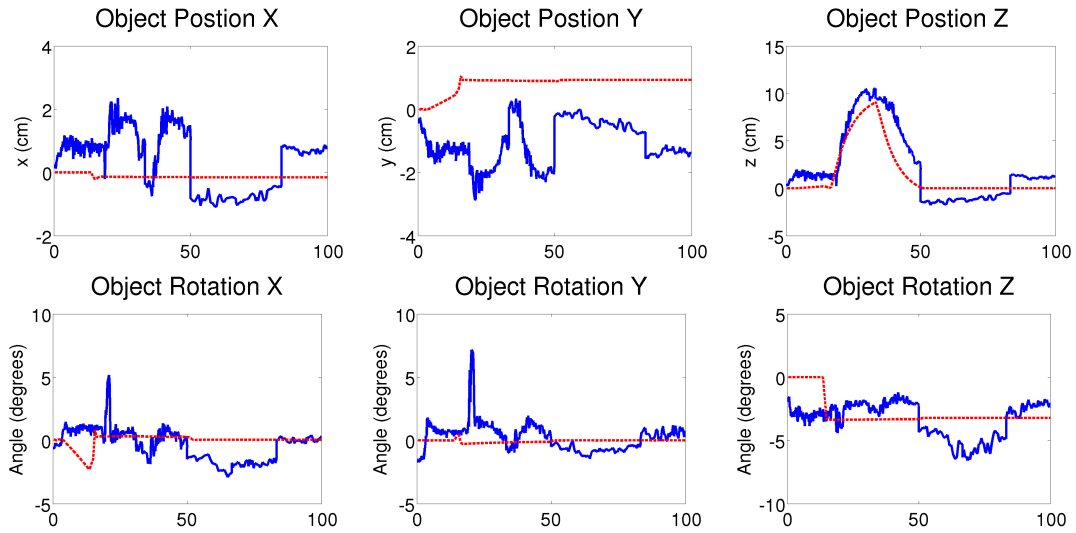



Figure 5.11: Object pose results for the first grasping experiment over time in percentage over the experiment duration. Blue lines indicate the readings from the robot and red dotted ones from the simulator.

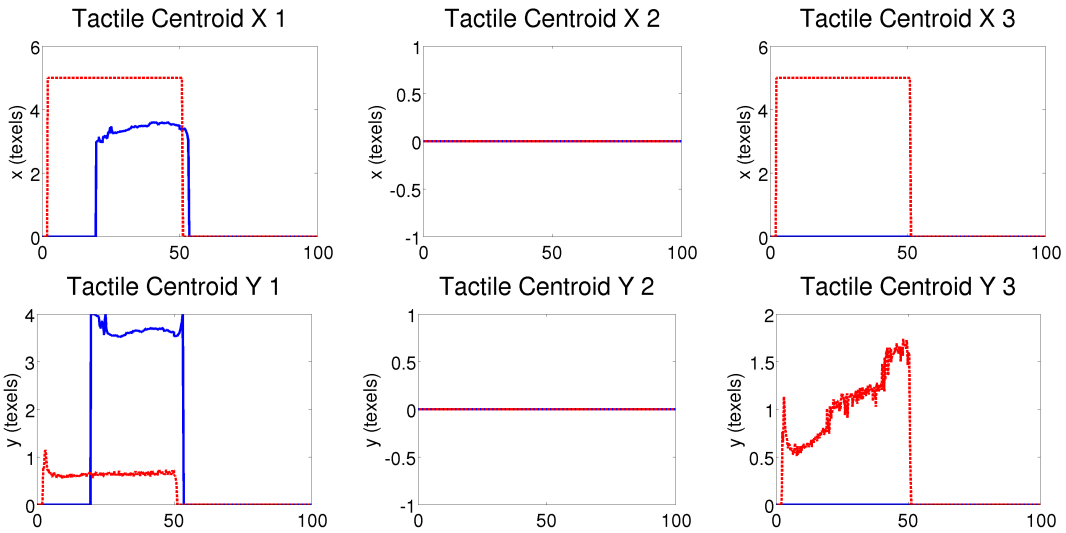



Figure 5.12: Tactile sensor results for the sliding experiment over time in percentage over the experiment duration. Blue lines indicate the readings from the robot and red dotted ones from the simulator.

## Slide

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

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

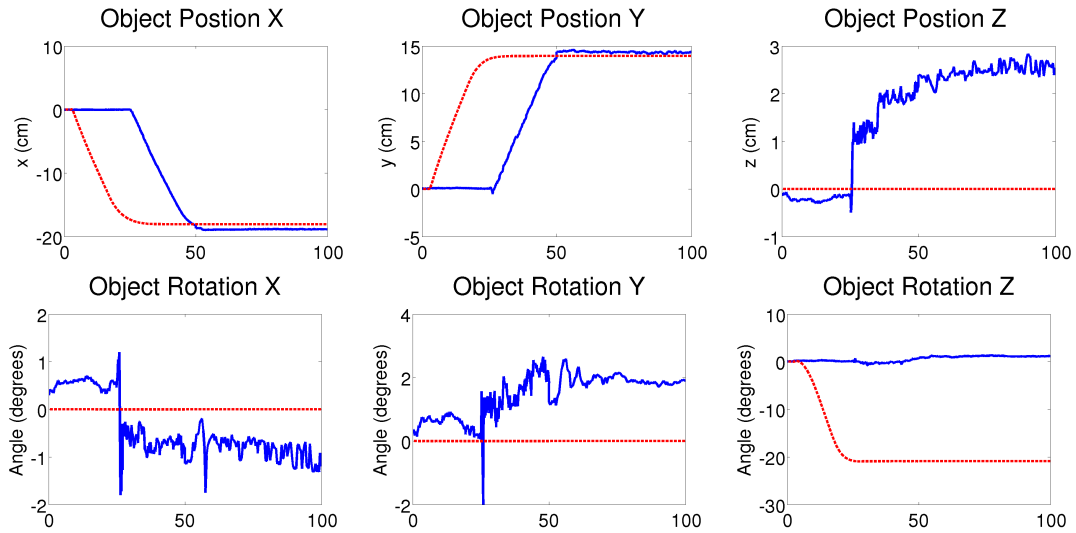



Figure 5.13: Object pose results for the sliding experiment over time in percentage over the experiment duration. Blue lines indicate the readings from the robot and red dotted ones from the simulator.

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

## In-hand manipulation

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

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

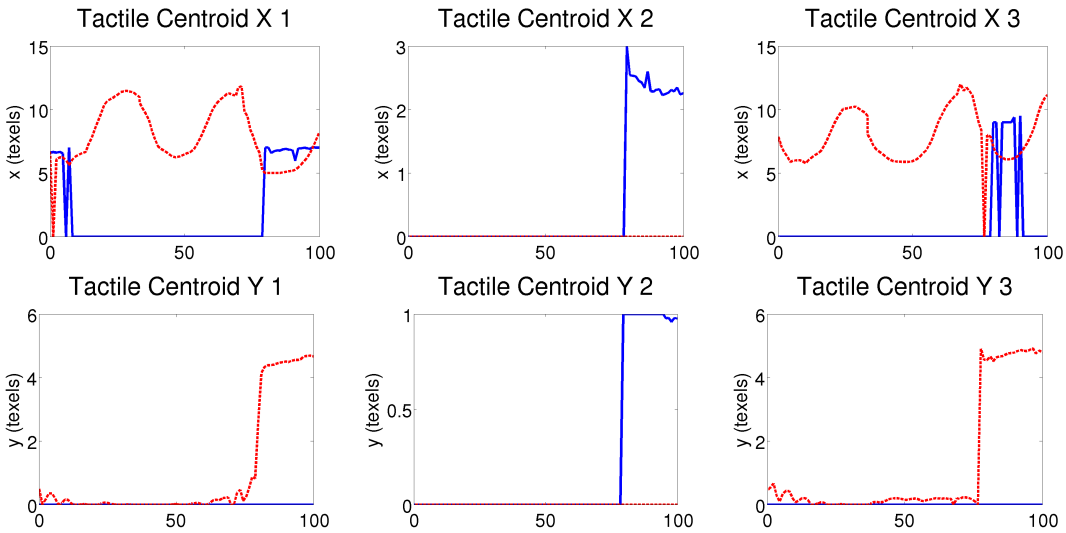



Figure 5.14: Tactile sensor results for the in-hand manipulation experiment over time in percentage over the experiment duration. Blue lines indicate the readings from the robot and red dotted ones from the simulator.

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

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

### 5.6.3 Global quantitative results

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

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

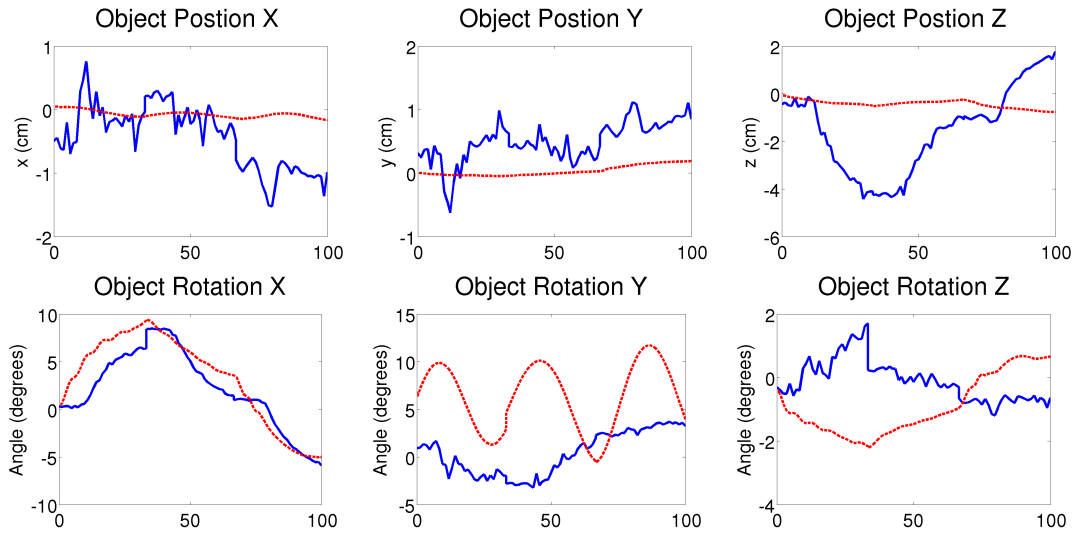



Figure 5.15: Object pose results for the in-hand manipulation experiment over time in percentage over the experiment duration. Blue lines indicate the readings from the robot and red dotted ones from the simulator.

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

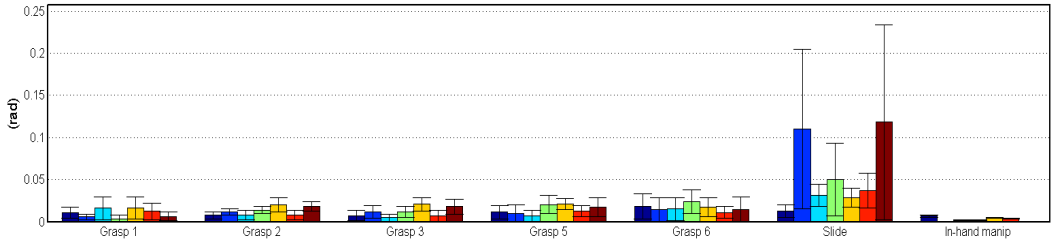
Regarding object position, the error (see dark blue column in Fig. 5.16(c)) is in general below 2.5 cm. While the rotation error, is below 0.05 radians which is very low. The special issues that rise up error, for the slide and in-hand manipulation task, are caused by the specific conditions already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

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

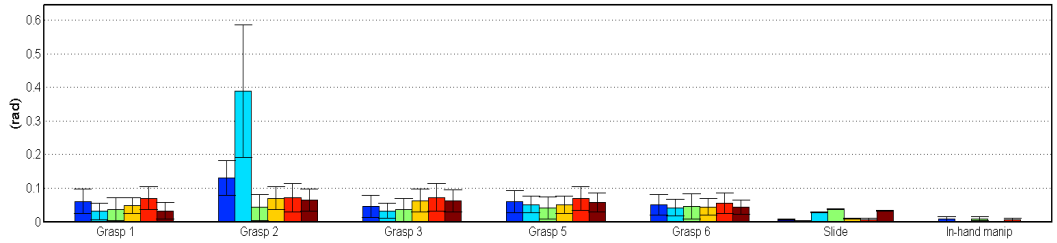
## 5.7 Simulation as a prediction engine

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

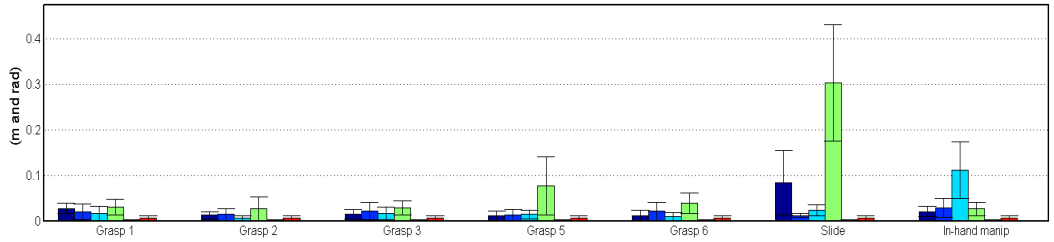
The simulated robot will run an exact copy of the controllers and tasks that are executed on the real robot. The controllers running on both robots will be independent. Thus,



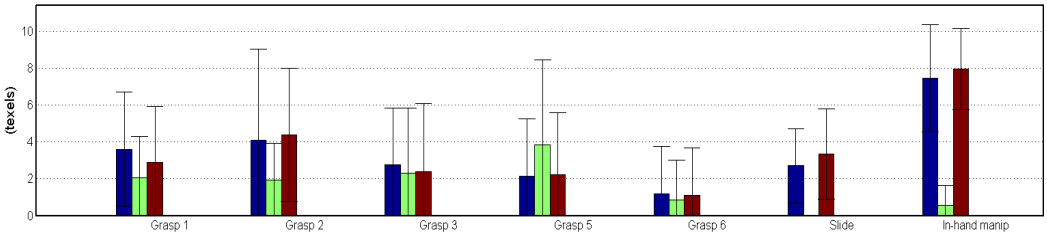

(a) Arm joint errors: mean and stdev for each joint of the arm grouped by experiment. For each group, from left to right: joint A1(dark blue) to joint A7(brown).




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




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




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

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

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

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

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

## 5.8 Conclusions

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

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

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

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

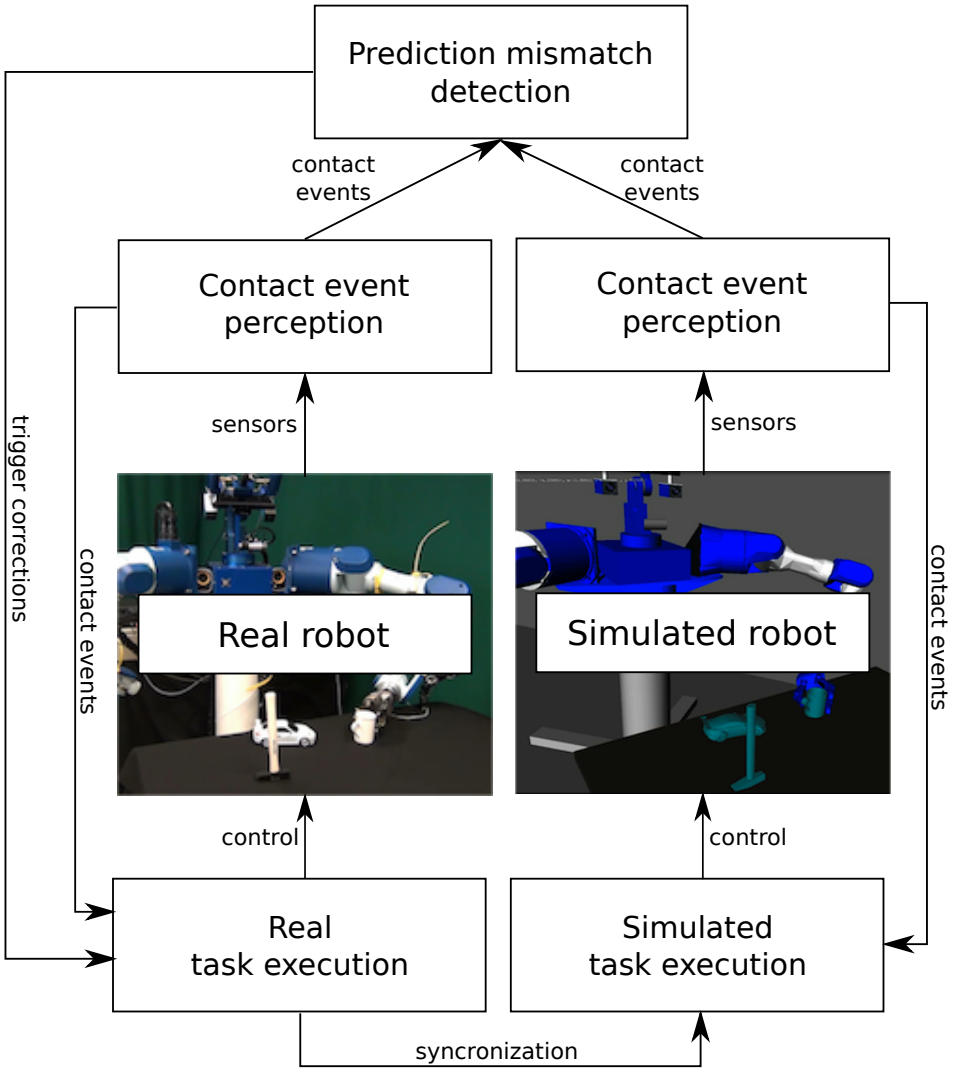



Figure 5.17: Integration of the simulator as a prediction engine into the presented framework. The real task and the simulated are running at the same time, the predicted contact events (obtained from the simulation) are compared with the contact events perceived from the real robot. The mismatches trigger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

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

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

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

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

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

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