Robot Door Opening

Erik Fahlén	Karins Allé 3
	181 45 Lidingö
	070 - 592 16 59
	efahlen@kth.se
Josef Sunesson	Beckombergavägen 3
	168 54 Bromma
	073 - 215 43 70
	josefsun@kth.se

Supervisor: Mårten Björkman

May 21, 2012

Abstract

In this thesis, the problem of a robot opening a door is simulated in a virtual environment. The environment used is OpenRAVE, with parts of the physics simulations written in Python. In order to solve the problem, a dynamic force/velocity controller is used to open doors that have trajectories that are unknown to the robot beforehand. The performance of this controller is tested with different control gains and starting guesses, though the results did not give a full understanding of how well it actually worked. For a complete understanding, more testing would be required.

Sammanfattning

I den här rapporten undersöks problemet med att en robot öppnar en dörr, i en simulerad miljö. Miljön som används är OpenRAVE, förutom en del av fysiksimuleringarna som är skrivna i Python. För att lösa problemet används en dynamisk kraft/hastighet-controller som även klarar dörrar där roboten inte känner till dörrarnas rörelsebanor i förväg. Controllerns framgång testas med olika värden och startgissningar, men resultaten gav inte någon klar bild av hur bra den faktiskt fungerade. För att få en komplett bild skulle kräva mer testning.

Contents

Chapter 1

Introduction

In recent years, more efforts have been put on creating robots that operate in domestic environments. Such environments are seldom well structured, which makes the operating of a robot in such an environment a real challenge. One of the most commonly encountered objects in such environments are doors. The task of opening a door where some of the door properties (i.e. the location of its hinges or which direction it opens in) are not known to the robot is not a trivial task in any way. None the less, it is a task into which much research and effort is being put to achieve some way to smoothly complete the task. One unpublished paper has proposed an adaptive dynamic force/velocity controller which uses measurements of force and position/velocity to deal with the task of opening a door where all the specifics are not known beforehand[5].

1.1 Statement of Collaboration

Both of the authors of this report were very interested at the prospect of learning a new API in an unknown programming language – something which was crucial for solving this problem. It is certainly a more advanced task than what the authors are used to be assigned in university courses.

There has not been a formal plan for how work was divided within the group – as we, the authors, are only two people it was not deemed necessary. When it comes to the programming, the solutions have been discussed within the group, but Erik has been doing most of the actual coding (roughly 75%). The work of writing the report has been divided equally between both parties.

Finally, the authors would like to extend a big thank you to Yiannis Karayiannidis and Christian Smith at the Centre for Autonomous Systems at KTH, for being extremely helpful with explaining the physics of this problem and basic robotics theory.

1.2 Problem Statement

The purpose of this thesis is to create and perform simulations of a robot trying to open a door where the robot does not know all the specifics of the door. The simulations evaluate different control gains in an adaptive force/velocity controller used for trying to open those doors. The goal of the simulations is to

try to find control gains that will make the robot only exert forces that will not break neither the door nor the robot while opening the door. Moreover, this thesis also evaluates whether or not a software library called OpenRAVE is a suitable tool to use for performing simulations of robot controllers such as the one used in this thesis.

Chapter 2

Background

In the first section of this chapter, the adaptive force/velocity controller under evaluation in the simulations done in this thesis will be introduced. In the subsequent sections, an introduction is given to the software library OpenRAVE and a few basic concepts in robotics.

2.1 Adaptive Force/Velocity Controller

In order for a robot to perform any kind of action, there has to be some controlling mechanism sending commands to robot telling it what to do. In the simulations performed in this thesis work, the objective of the robot has been to open a door about which not all the specifics has been known to the robot in advance. To achieve that objective, an adaptive force/velocity controller proposed in a currently unpublished paper[5] has been used. This controller is in turn designed for trying to achieve a smooth opening of an unknown door.

For the controller to achieve this smooth opening, it uses dynamically calculated force and velocity signals, which it wants the end-effector (see section 2.3.2) to use. Those signals have in turn been calculated from estimates about the kinematics of the door. A description of how those estimates and signals are calculated will follow, though before that, some notations and then a model of the robot door opening problem will be given.
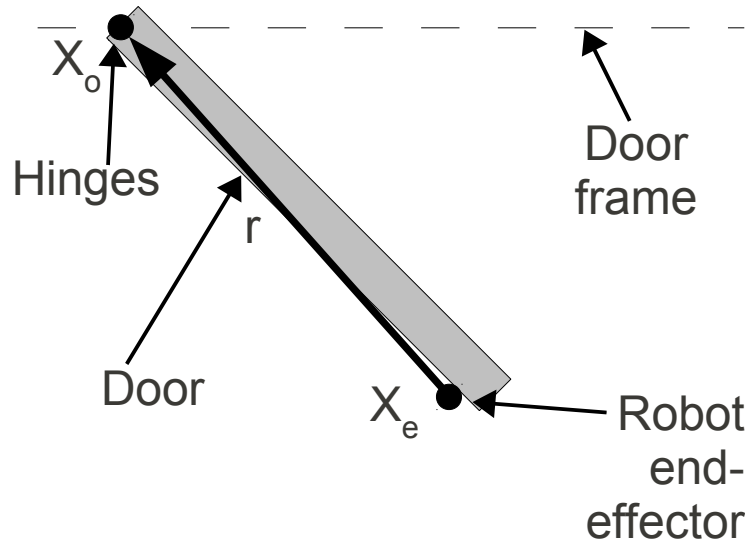
2.1.1 Notations

For convenience, these notations will be used occasionally:

$$\underline{\mathbf{z}} = \frac{\mathbf{z}}{||\mathbf{z}||}$$

$$\mathbf{s}(\mathbf{z}) = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \mathbf{z}$$

2.1.2   Kinematic Model

Figure 2.1: A basic overview of the door opening.

The kinematic model of the robot door opening problem used by the controller assumes a planar definition of the door opening, which means that it does not work with doors where also vertical movements have to made in order to open the door. It is also assumed that the robot has achieved a fixed grasp on the handle of the door. This, which is depicted in figure 2.1, provides the following:

$\mathbf{r} = \mathbf{X}_o - \mathbf{X}_e$

$\mathbf{X}_o$ is the position of the hinges of the door, though that position is not known by the robot, and $\mathbf{X}_e$ is the position of the end-effector. In the force and velocity signals, estimations of $\mathbf{r}$ and $\mathbf{X}_o$ play an extremely important part. When the end-effector is moved, forces arise between the end-effector and the kinematic mechanism of the door. Seen from the perspective of the door, these forces act in the radial direction, which is the direction of $\mathbf{r}$, and in its orthogonal direction, which is the tangential direction. The force along the radial direction is defined as follows:

$f_r = \underline{\mathbf{r}}^\top \mathbf{F}$ where $\mathbf{F}$ is the total force acting between the end-effector and the kinematic mechanism of the door.

2.1.3   Control Design

To begin with, the controller has a desired radial force, f_{rd}, and a desired velocity in the tangential direction, v_d. The objective of the controller is to achieve: $f_r \to f_{rd}$ and $\dot{\mathbf{X}}_e \to \underline{\mathbf{r}} v_d$, while not knowing the direction of $\underline{\mathbf{r}}$.[1] As previously stated, the controller uses estimates of the kinematics of the door. More specifically, it uses estimates of the center of rotation, $\hat{\mathbf{X}}_o(t)$. From this

---

[1] Newton's notation, the dot notation, will be used for differentiation in this thesis.

estimate of the center of rotation, an estimated radial direction, $\hat{\mathbf{r}}(t)$, can be calculated as follows: $\hat{\mathbf{r}}(t) = \hat{\mathbf{X}}_o(t) - X_e$.[2] The estimate of the radial direction is then used to calculate the force and velocity signals, and to update the estimate of the center of rotation. The formula for calculating the force signal will be omitted here because it is only used to control the motion of the robot's arm and that OpenRAVE itself can handle that. However, part of the force signal formula appears in the formula for updating the estimate of the center of rotation.

The formula for calculating the velocity signal looks like this:

$\mathbf{v}_{\text{ref}} = \mathbf{s}(\hat{\underline{\mathbf{r}}})v_d - \alpha\hat{\underline{\mathbf{r}}}v_f$

α is a positive control gain and v_f is a force feedback term defined as:

$$v_f = \int_0^t \tilde{\hat{f}}_r \, \mathrm{d}t$$

The term $\tilde{\hat{f}}_r$ is the error of the estimated force in the radial direction. The estimated force in the radial direction is defined as: $\hat{f}_r = \hat{\underline{\mathbf{r}}}^\top \mathbf{F}$. With that, $\tilde{\hat{f}}_r$ is defined as: $\tilde{\hat{f}}_r = \hat{f}_r - f_{rd}$.

To update the estimation of the center of rotation, the following update law is defined:

$$\dot{\hat{\mathbf{X}}}_o = \gamma ||\hat{\mathbf{r}}||^{-1} \left[(k_f + 1)\tilde{\hat{f}}_r + k_I \int_0^t \tilde{\hat{f}}_r \, \mathrm{d}t \right] \mathbf{v}$$

$\mathbf{v}$ is the current velocity of the end-effector and γ, k_f and k_I are three more control gains.

2.2   OpenRAVE

OpenRAVE (Open Robotics Automation Virtual Environment) is a virtual environment designed for testing motion planning algorithms for robotics applications[1]. It is plugin-based and quite a few different plugins with different useful functionality are offered in the standard installation package. One of those plugins provided gives the user the possibility to create a nice graphical visualization of a simulation. That functionality makes it a bit easier to understand what is happening in a simulation.

In order to perform a simulation in OpenRAVE, a model of an environment that is supposed to be simulated has to be created. When that is done, objects in that model can be manipulated using a high level programming language.

2.2.1   Models

Models in OpenRAVE are stored in XML files.[3] In its essence, a model can be a definition of an object, a robot or several objects and robots. In the XML-format used by OpenRAVE, a definition of an object has to be of the type *KinBody*. A *KinBody* in turn can consist of several rigid bodies connected by joints. A definition of a robot is of the type *Robot*, though the robot will in turn consist of at least one *KinBody* definition.

---

[2] The argument of t will onwards be omitted for notational convenience.

2.2.2 Programming

While the core OpenRAVE API is written in C++, there is also a Python implementation which uses the C++ API seamlessly. The Python API makes it easy to quickly do high level scripting in OpenRAVE. The software is distributed with several Python examples that make robots do simple things such as grabbing an object or moving a hand in a straight line. These examples is a great help to quickly get a basic environment set up when first starting to use the OpenRAVE library.

2.3 Robotics Theory

While it is not necessary to be an expert on robotics to understand the simulations in this thesis, a few concepts encountered in this thesis work and in the code behind the simulations will be explained in this section.

2.3.1 Degrees of Freedom

Degrees of freedom (DOF) is a general mechanical term describing the number of parameters that define a system's state[7]. A common example in the real world is six degrees of freedom - three parameters for position and three parameters for rotation. A robot arm consists of many different individual parts that themselves have very few degrees of freedom. While every joint individually might have only one or two DOF, the total DOF of the arm is a combination of all the joints.

2.3.2 End-effector

An end-effector is a tool of some kind that sits at the end of a robot arm. It is typically some sort of grasper. In this report there is no specific tool at the end of the arm − the end-effector simply refers to the tip of the arm.

2.3.3 Inverse Kinematics

Inverse kinematics (IK) is an important part of robot motion planning. It plans the motion path for a robot arm by determining the joint movements that give the desired position of an end-effector.[6] Motion planning with inverse kinematics can therefore transform a motion path of an end-effector into joint movements. When these joint movements are performed, the end-effector will follow the desired motion trajectory until it reaches the desired position.

Chapter 3

Methods

In this part, the methodology used to solve the problem stated in this thesis will be described. A short description of what has been done is that first, a simulation in OpenRAVE was created, and then, statistics of different setups were collected. Simultaneously with those phases, the suitability of OpenRAVE was evaluated. A more exhaustive description of all those phases follows in the subsequent sections.

3.1   Creating Simulation

To begin with, the first step was to play around with the examples provided along with the OpenRAVE library to try and make some sense as to what and how things could be done with this library. The next step was then to setup a basic environment with a robot and a door present in them. Both a robot, which is fairly similar to a real robot used at CVAP[1], a model of a door already existed in the examples provided. Therefore, those objects only had to be merged together in a single environment to have a rudimentary setup.

---

[1]Computer Vision and Active Perception Lab at KTH.

3.1.1   Physics Model

Figure 3.1: The end-effector is connected to the door handle by a spring. The spring force is depicted in this figure.
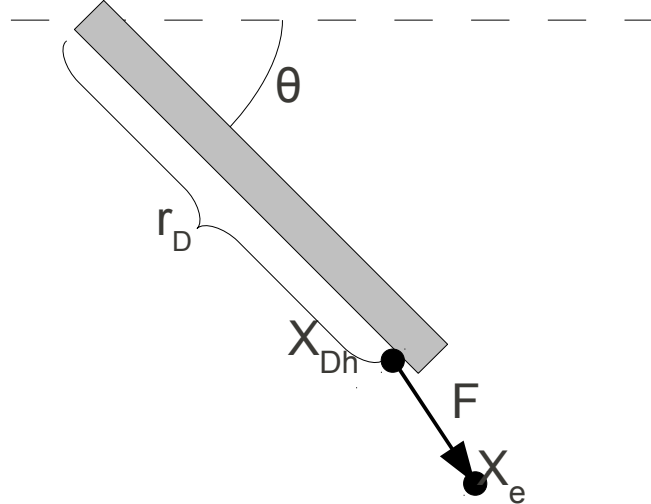
With that simple setup complete, the next objective was to implement the controller. In order to make it work, some sort of physics model had to be implemented first. The one that was used assumes that the end-effector is connected to the handle of the door by a very stiff spring. That way, a simulation of the forces acting on the end-effector and the door exist. Those forces arise through the fact that when the end-effector is moved, the spring becomes stretched out. When the spring is stretched out, a force arises to return the spring to its relaxed state. The magnitude of that force follows Hooke's law[4]: $F = -Kx$, where x is the distance the spring has been stretched out and K is a constant representing the stiffness of the spring. In this model, $\mathbf{F}$ is therefore defined as:

$\mathbf{F} = -K(X_{Dh} - X_e)$ where X_{Dh} is the position of the door handle as can be seen in figure 3.1. When $\mathbf{F}$ has been calculated, it can through a basis change give the force acting in the radial direction of the door, $\mathbf{F}_R$, and the force acting in the tangential direction of the door, $\mathbf{F}_T$. The value of $|\mathbf{F}_R|$ is a measurement of how much pressure is applied towards or away from the door hinge. If it becomes too large, either the door or the arm of the robot arm would break in a real-world situation. The tangential force, $\mathbf{F}_T$, is in turn the one that affects the rotation of the door. In the physics model used, the torque of the door is defined as the following: $\tau = \mathbf{F}_T r_D = c\dot{\theta}$, where r_D is the door radius (seen in 3.1), c is a friction constant and $\dot{\theta}$ is the angular velocity. Basic algebra then gives an equation for $\dot{\theta}$:

$$\dot{\theta} = \frac{\mathbf{F}_T r_D}{c}$$

In the formulas for $\mathbf{F}$ and $\dot{\theta}$, two constants are present. The values of those constants can be chosen a bit freely, but a rule of thumb is:

1. K should be a diagonal 2×2 matrix with the same value in the two diagonal positions. Since K represent the stiffness of an imaginary spring between the end-effector and the handle of the door, and that spring should be very stiff, this value should be large. In the simulations done in this thesis, the value 1000 has been used on the two diagonal position of K.

2. c should be a relatively small value compared to K. Different values on c results in different behaviours of a door. In the simulations done in this thesis, c has been set to 5.

3.1.2   Time Step

When the physics model was in place, the implementation of the controller could finally be done. After the controller was done, only one thing remained to have a working simulation was to introduce some form of time simulation. This was done by introducing a time step, dt. In the simulations done in this thesis, dt has been set to 0.001 seconds. That means that the system is updated 1000 times per second. With this time step, the position of the end-effector could be updated like this:

$\mathbf{X}_{t+1} = \mathbf{X}_t + \dot{\mathbf{X}}_t dt$

This formula and the implemented physics model is what enables the controller to do its work in the simulations done in this thesis. As a side note, calculations of integrals use numerical estimation, which means that they are calculated by multiplying dt with the current value of the variable to integrate and adding it to the integral sum.

3.2   Evaluating OpenRAVE

Evaluating the suitability of OpenRAVE was impossible to do in an entirely objective way. Therefore, the verdict of the suitability of OpenRAVE was a highly subjective impression based on the time spent and the experiences gained by the authors of this thesis, while trying to setup the simulations.

3.3   Measuring Results

In order for the results of the simulations to make sense and relate to each other, a way to measure how well the performance of the door opening was defined. First of all, the door were considered to have been completely opened when the angle θ, seen in figure 3.1, surpassed 60°. But more importantly, the way of measuring the performance of the controller was by calculating the force in the radial direction $f_r = \mathbf{r}^\top \mathbf{F}$. The value of f_r should converge to the desired radial force, f_{rd}. This force is also very interesting for the reason that it measures how much force the robot actually puts on the door. If this force is too large, either the robot or the door could break, so looking at this force over time could provide a good insight into how safe the door opening process would be. If the magnitude of the force exceeds the desired radial force (f_r) by a large factor, the structure of the door would probably be in danger.

Another interesting value to look at is how the planar angle of $\hat{\mathbf{r}}$ changed during the course of the simulation. This is interesting since this represents

the robots approximation of the sum of θ and the small additional angle that is generated from the protruding handle of the door. The angle of $\hat{\mathbf{r}}$ is supposed to converge towards the actual value of θ added with that small additional angle.

Chapter 4

Results

This chapter covers the results of the experiments. The first section of this chapter describes how the simulation creation in OpenRAVE progressed, while the rest covers data gathered regarding the actual simulations.

4.1 Simulation creation in OpenRAVE

The robot intended for simulation already existed in the OpenRAVE example files and there was also a model of a door in another example. This door model included a joint to simulate the door rotating on its hinges. Therefore, the first step in creating the simulation was to merge these two models into one environment, and the result can be seen in figure 4.1. The door was later modified to be slightly smaller, in order for the robot to open it further.

Figure 4.1: The robot and the door.
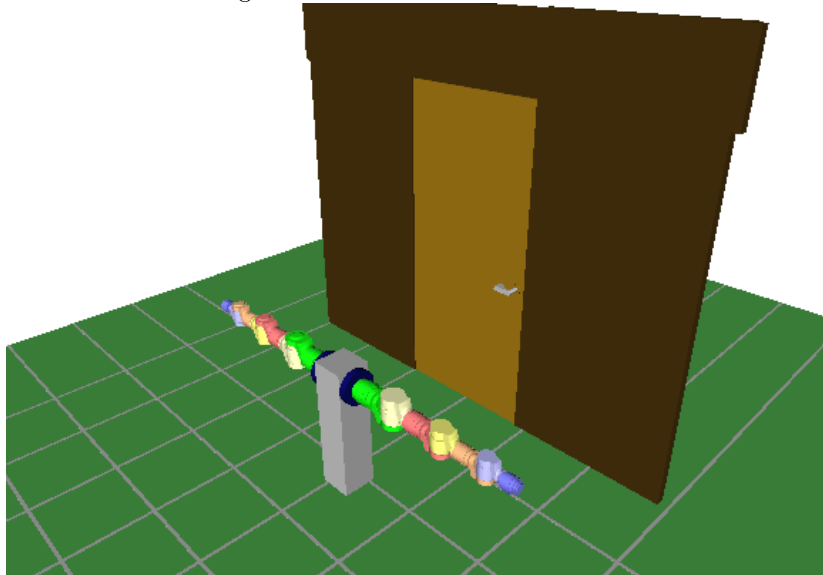
The next step was to achieve the supposed fixed grasp of the handle of the door. After that, it was time to implement the physics of the door opening process. This had to be implemented manually in Python, as OpenRAVE currently lacks the functionality to handle it[2]. After this step was completed, the door would start to move slightly if the end-effector was given some constant velocity to hold throughout the simulation. The result appears in figure 4.2.
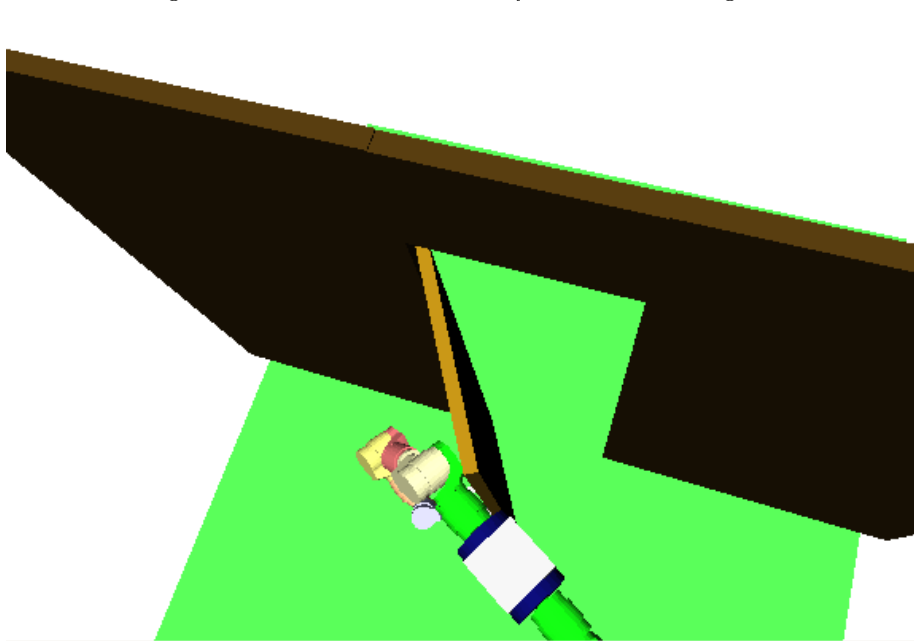
Figure 4.2: The robot and the door, now somewhat opened.
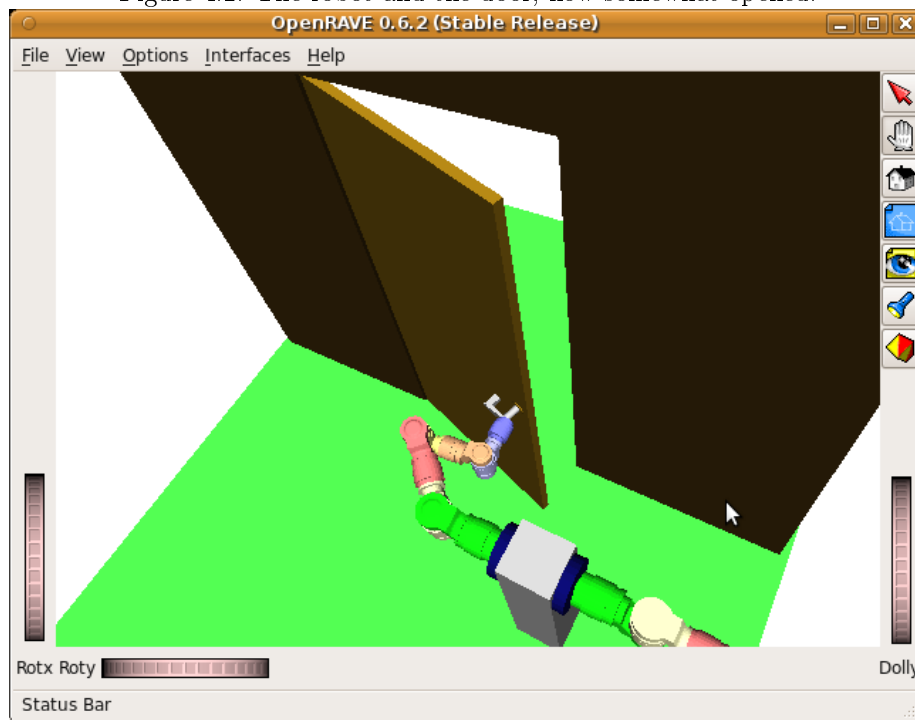
Finally, the adaptive force/velocity controller was implemented to achieve a complete door opening. After some tweaking of the different controller gains, the robot was finally able to open the door to a full 60 degrees. The result can be seen in figure 4.3.

Figure 4.3: The door successfully opened to 60 degrees.

As a side note, the location of the robot's base was found, not surprisingly, to play an important part in how far the door could be opened. Therefore, the robot's base was placed at a location where the robot had the possibility to open the door 60° in all the simulations made for this thesis work.

It can also be noted that opening of an average sized door was not possible without the used robot's base moving. Doors that are significantly smaller than the robot arm – such as a cupboard door – can work, but a large door big enough for a person to walk through would require a longer arm than the one that the robot used in the simulations, had. This was discovered while the physics model was being implemented. That is why the figures from that stage and onwards can be seen to have a smaller door than before.

4.2 Controller Gains

As expected, the performance of the door opening varied a lot depending on the starting guess of the center of rotation and on the values of the different controller gains.

The next figure, 4.4, shows the performance of the controller in a first experiment.

Figure 4.4: The first experiment. The diagram shows the radial force along the time.
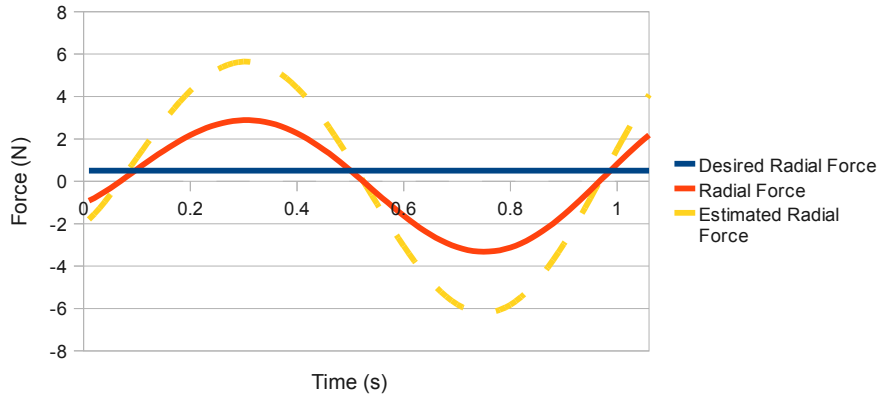
This experiment had a correct starting guess of the center of rotation, which resulted in a complete door opening. The following control objectives: $v_d = 0.5$ and $f_{rd} = 0.5$ were used, and the following controller gains were used in that experiment: $\gamma = 0.001, \alpha = 0.05, k_f = 0.002$ and $k_I = 0.0015$. Despite a correct starting guess, both the estimated radial force and the actual radial force seem to *move around* the desired radial force rather than *converging* towards it, as seen in figure 4.4. The forces involved were almost 6 times larger than the desired radial force, which means that the structure of the door could be in danger. This probably indicates that the gains used were not optimal.

Figure 4.5: The second experiment. The diagram shows the radial force along time.
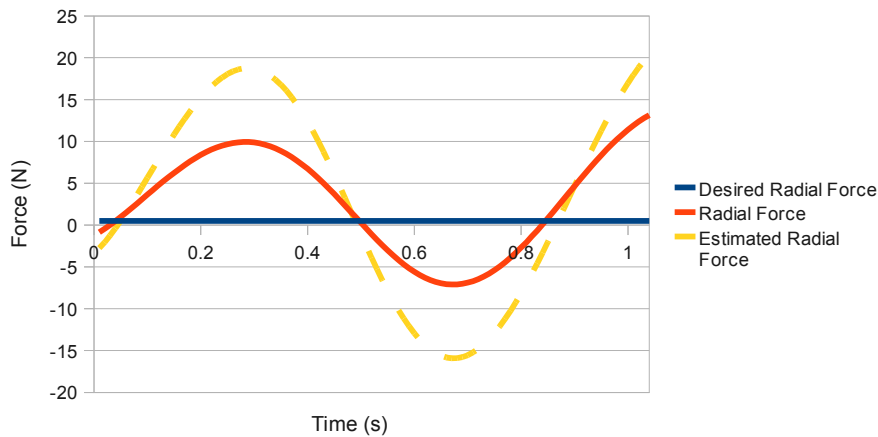

Figure 4.5 shows another test run were the starting guess was moved to 2 decimeters off the correct position along the axis of the door frame. The control objectives were still the same as in the first experiment, though the

controller gains were increased slightly in order to improve on the results in the last experiment. They were changed to $\gamma = 0.01, \alpha = 0.05, k_f = 0.02$ and $k_I = 0.015$. Despite this, the same behaviour can be observed as in the first experiment – there was no convergence going on. The behaviour was actually even worse and the forces involved were now almost 20 times the desired in the radial direction. This means that something probably would have broken in a real life scenario. In the simulation though, the door were still successfully opened.

The figure 4.6 shows the change of the planar angle of $\hat{\mathbf{r}}$ over the course of the second experiment.

Figure 4.6: How the angle of the estimated radial direction changed over time compared to how much the door had been opened.
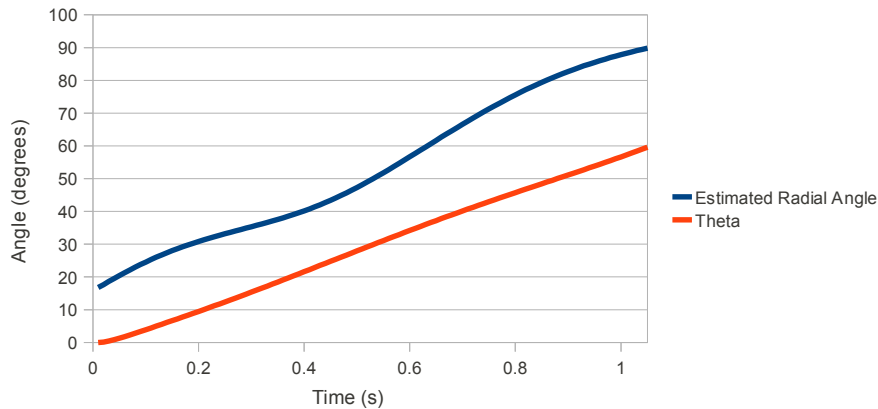
With the guess off two decimeters along the axis of the door frame, the angle should start out to be a little too big and shrink to a correct value. What figure 4.6 shows, is that it indeed starts high but that it does not converge towards the actual value of θ throughout the simulation. The behaviour of $\hat{\mathbf{r}}$ seen in the figure 4.6 indicates that the initial guess is not actually updated that much. To try to change this, one could try additional control gains, but in the limited time frame of this project, it was unfortunately not possible to do so.

A number of other experiments with too high control gains failed completely, giving insanely high forces that would definitely have broken the door or the robot. Therefore, the best solution appears to be to keep them small and increase them later if it turns out they are to weak. This minimizes the risk that something will break and also works very well with a decent starting guess. If the starting guess is only a couple of decimeters off it is likely to work, and the robot can probably calculate a starting guess within those limits based on visual information of the door.

Chapter 5

Discussion

OpenRAVE proved to be far from easy to learn, despite the extensive documentation available. The original plan was to do the coding in C++, but since all examples were written in Python, it was decided that at least the simulation would be created using Python. As a result, additional time had to be placed on learning the basics of Python. Luckily, Python is fairly easy to read, but programming in an unfamiliar programming language is always a rather time consuming task.

A main part of the simulation was the idea that the robot end-effector would be directly locked to the handle, as the task of actually grabbing the handle with a robot hand was not part of this experiment. It was planned that OpenRAVE would take care of these simulations, but this turned out to be impossible, or at least very unstable.[2] As the project still required this behaviour, a new, simple physics engine for this specific purpose had to be implemented. While this was not particularly difficult in the end, much time elapsed before it could be concluded that this was the only way out.

5.1 Additional Testing

Due to these delays, the final steps of the simulation were developed in the last few weeks of the project, much later than estimated in the original plan. This late schedule probably increased the possibility of errors in the code, even though it has been written with readability in mind. Some of the results were certainly confusing, like the radial force not converging to its desired value, but hopefully this was due to what type of door was used rather than simple programming errors.

The late schedule unfortunately made it impossible to extend the project to the level it first was intended, as there was not enough time left. One of the things that could have been expanded more was the testing of different controller gains. Since there are so many combinations to try, it would have taken up much more time than originally anticipated to do it thoroughly, and with the delays mentioned, there was even less time than planned to do it. This testing is not difficult to do and could be a potential future extension to this work.

5.2   The Physics Model

The physics model used in this thesis relies on the robot end-effector being connected by a spring to the handle of a door. This may not be the best way to do this simulation. Even though it works well as an approximation, this is not what is going on in the real world.

This thesis only works the *position* of the end-effector. In a more detailed simulation, one could also choose to include the *orientation* of the end-effector. In the performed simulations, the orientation is automatically updated to stay perpendicular to the door, only to make sure the arm can reach slightly longer. The orientation could instead be made part of the simulation by assigning a desired angle and updating the orientation to be perpendicular to the estimated radial direction rather than to the door. This would make the simulation more accurate, but would also introduce more room for errors.

Bibliography

[1] Rosen Diankov. *Automated Construction of Robotic Manipulation Programs.* PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010. URL `http://www.programmingvision.com/rosen_diankov_thesis.pdf`.

[2] Rosen Diankov. Openrave mailing list – grab part of a kinbody. `http://openrave-users-list.185357.n3.nabble.com/Grab-part-of-a-KinBody-tc3828672.html`, April 2012.

[3] Rosen Diankov et. al. Openrave custom xml format. `http://openrave.programmingvision.com/wiki/index.php/Format:XML`, April 2012.

[4] D. Halliday, R. Resnick, and J. Walker. *Fundamentals of Physics. 8th ed., Extended.* John Wiley & Sons, Inc., New Jersey, 2008. ISBN 9780471758013.

[5] Y. Karayiannidis, C. Smith, P. Ögren, and D. Kragic. Adaptive force/velocity controls for opening unknown doors. Has not been published at the time of this report.

[6] Richard Paul. *Robot manipulators: mathematics, programming, and control: the computer control of robot manipulators.* MIT Press, Cambridge, 1981. ISBN 9780262160827.

[7] J. J. Uicker, G. R. Pennock, and J. E. Shigley. *Theory of Machines and Mechanisms.* Oxford University Press, New York, 2003. ISBN 9780195155983.

Appendix A

Source code

A.1   dooropening.py

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from __future__ import with_statement # for python 2.5

import time
import openravepy
if not __openravepy_build_doc__:
from openravepy import *
from numpy import *

class DoorOpener:
def length(self, a):
return sqrt(vdot(a, a))

def norm(self, a):
return a / self.length(a)

def BlackBox(self, xPos, f):
# Here are some constants
vd = 0.5
frd = 0.5

# These are the control gains
gamma = 0.001
alpha = 0.05
kf = 2
kI = 5

# Calculate the radial estimate
radialEstimate = self.hingeEstimate - xPos
```

```
# Ignore z axis
radialEstimate[2] = 0

# Print the error
error = 1 - dot(self.norm(radialEstimate), self.
    radialDirection)
#print("%.2f|t%.4f" % ((self.i * self.dt), error))

# Time
# Desired radial force
# Radial force
# Estimated radial force
pt = self.i * self.dt
prf = dot(transpose(self.radialDirection), f)
perf = dot(self.norm(transpose(radialEstimate)), f)
print("%.2f\t0.5\t%.4f\t%.4f" % (pt, prf, perf))

# Create the transformation to the other basis
R = array([[0, -1, 0],
       [1, 0, 0],
       [0, 0, 1]])

# Calculate the estimated velocity direction
velocityDirectionEstimate = dot(R, self.norm(
    radialEstimate))

# Remove the vertical component
velocityDirectionEstimate[2] = 0

# And finally, calculate the estimated velocity
velocityEstimate = velocityDirectionEstimate * vd - alpha
     * self.norm(radialEstimate) * self.
    radialForceEstimateErrorIntegral

# Let's update the handle position estimate!
# Caluclate the radial force estimate thingys
radialForceEstimate = dot(transpose(self.norm(
    radialEstimate)), f)
radialForceEstimateError = radialForceEstimate - frd
self.radialForceEstimateErrorIntegral += self.
    physicLoopsPerControlLoop * self.dt *
    radialForceEstimateError
radialLength = self.length(radialEstimate)

# Calculate the estimated hinge position change
factor = (gamma /  radialLength) * ((kf + 1) *
    radialForceEstimateError + kI * self.
    radialForceEstimateErrorIntegral)
hingePositionChangeEstimate = factor * transpose(self.
    xVel)
```

```
# Finally , update the estiated hinge position
self . hingeEstimate += self . physicLoopsPerControlLoop *
    self . dt * hingePositionChangeEstimate

return velocityEstimate

def OpenDoor ( self , env , options ) :
"Main example code . "
env . Load ( options . scene )

#raw_input (" Press ENTER to start !")

# Get the robot and select the left arm
robot = env . GetRobots () [0]
robot . SetActiveManipulator ( 'leftarm ')

# Lock environment
with env :
 # Create the inverse kinematics model
 ikmodel = databases . inversekinematics .
     InverseKinematicsModel ( robot=robot , iktype=
     IkParameterization . Type . Transform6D )
 if not ikmodel . load () :
  ikmodel . autogenerate ()
 basemanip = interfaces . BaseManipulation ( robot )

 # Get the door handle
 door = env . GetKinBody ( 'door ')
 handle = door . GetLink ( 'handle ')
 handleTransform = handle . GetTransform ()

 # Get the specific box that is the handle
 geom = handle . GetGeometries () [1]

 # Calculate the world transform of the box
 handleTransform = dot ( handleTransform , geom . GetTransform
     () )

# Move left arm to door handle
param = openravepy . IkParameterization ( handleTransform ,
    IkParameterization . Type . Transform6D )
sol = robot . GetActiveManipulator () . FindIKSolution ( param ,
    IkFilterOptions . IgnoreEndEffectorCollisions ) # get
    collision−free solution

time . sleep (1)

# Were we able to find a solution ?
if ( sol is not None and len ( sol ) > 0) :
```

```python
# Let's move the arm to the start position!
time.sleep(1)
robot.SetDOFValues(sol, ikmodel.manip.GetArmIndices())

# Wait for a little while before starting the simulation
time.sleep(1)

# Setup the controller estimates
self.hinge = [-0.5, -0.3, 0]
self.hingeEstimate = [-0.5, -0.2, 0]

# Set the constant values
K = 1000 * identity(3)
c = 5
self.dt = 0.001
radius = 0.5
goal = math.radians(60)
self.radialForceEstimateErrorIntegral = 0
self.physicLoopsPerControlLoop = 10

# Set start values
self.xVel = [0.0, 0.0, 0.0]

self.i = 0
while(True):
  if(door.GetDOFValues()[0] >= goal):
    print("Door opened the full %d degrees!" % math.ceil(
        math.degrees(goal)))
    break
  if(self.i >= 10000):
    print("The physics loop ran for too many iterations.
        Door opened %d degrees!" % math.degrees(theta))
    break

  self.i += 1
  # Update the end effector position
  xPos = robot.GetActiveManipulator().GetTransform()
      [0:3,3]
  xTrans = robot.GetActiveManipulator().GetTransform()
  xVelDelta = [x*self.dt for x in self.xVel]
  xPos += xVelDelta
  xTrans[0:3,3] = xPos

  # Get handle position
  handleTransform = dot(handle.GetTransform(), geom.
      GetTransform())
  handlePos = handleTransform[0:3,3]

  # Calculate the force between the door and handle
  handleDirection = xPos - handlePos
```

```python
handleDirection[2] = 0
force = dot(K, handleDirection)

# Get the door angle
theta = door.GetDOFValues()[0]

# Create the transformation to the other basis
R = array([[math.cos(theta), -math.sin(theta), 0],
           [math.sin(theta), math.cos(theta), 0],
           [0, 0, 1]])

# Transform the force to the door basis
doorForce = dot(R, force)

# Update the angle
thetaVel = self.dt * (doorForce[0] * radius / c)
theta += thetaVel
door.SetDOFValues([theta, 0])
self.radialDirection = self.hinge - xPos

# Update the angle of the "hand" so it stays
#    perpendicular to the wall.
R = array([[math.cos(-thetaVel), -math.sin(-thetaVel),
    0],
           [math.sin(-thetaVel), math.cos(-thetaVel), 0],
           [0, 0, 1]])
xTrans[0:3,0:3] = dot(R, xTrans[0:3,0:3])

# Move the arm
param = openravepy.IkParameterization(xTrans,
    IkParameterization.Type.Transform6D)
sol = robot.GetActiveManipulator().FindIKSolution(param
    , IkFilterOptions.IgnoreEndEffectorCollisions)

# Stop if we can't move the end effector there.
if(sol is None or len(sol) == 0):
print("Cannot move the arm any further. Door opened %d
    degrees!" % math.degrees(theta))
break

# Update the end effector to its new position
robot.SetDOFValues(sol, ikmodel.manip.GetArmIndices())

# Run the magic function!
if(self.i % self.physicLoopsPerControlLoop == 0):
self.xVel = self.BlackBox(xPos, force)

# Take a nap
time.sleep(self.dt)
```

```python
# Did we manage to open the door?
if(door.GetDOFValues()[0] > goal):
 # Make victory gesture with the right arm
 time.sleep(1)
 robot.SetActiveManipulator('rightarm')
 basemanip.MoveManipulator(goal=[math.radians(90), math.
     radians(90), 0, 0, 0, 0, 0])

 # Done.
 raw_input('Press any key to quit...')
else:
# Nope, something is wrong.
 print 'Unable to reach the door. Try to move something
     in the environment!'

def main(env, options):
opener = DoorOpener()
opener.OpenDoor(env, options)

from optparse import OptionParser
from openravepy.misc import OpenRAVEGlobalArguments

@openravepy.with_destroy
def run(args=None):
parser = OptionParser(description="A door opening.")
OpenRAVEGlobalArguments.addOptions(parser)
parser.add_option('--scene',
     action="store",type='string',dest='scene',default='
         ourdoor.env.xml',
     help='Scene file to load')
(options, leftargs) = parser.parse_args(args=args)
env = OpenRAVEGlobalArguments.parseAndCreate(options,
    defaultviewer=True)
main(env,options)

if __name__ == "__main__":
run()
```

A.2 ourdoor.env.xml

```xml
<Environment>
<bkgndcolor>1 1 1</bkgndcolor>
<camtrans>0.870919 1.170534 3.276345</camtrans>
<camrotationaxis>
-0.498347 -0.817255 0.289388 170.509065
</camrotationaxis>

<Robot file="robots/schunk-lwa3-dual.robot.xml">
<translation>0.18 0.12 0.57</translation>
<rotationAxis>0 0 1 -30</rotationAxis>
```

```
</Robot>

<KinBody name="door">
<translation>-0.5 0 0</translation>
<rotationaxis>1 0 0 90</rotationaxis>
<rotationaxis>0 0 1 -90</rotationaxis>

<Body name="frame" type="static">
 <Geom type="box">
  <extents>1.5 0.215 0.0274</extents>
  <translation>0 2.27 0</translation>
  <diffusecolor>0.36 0.25 0.0667</diffusecolor>
 </Geom>
 <Geom type="box">
  <extents>0.65 1.245 0.0274</extents>
  <translation>-0.96 1.245 0</translation>
  <diffusecolor>0.36 0.25 0.0667</diffusecolor>
 </Geom>
 <Geom type="box">
  <extents>0.65 1.245 0.0274</extents>
  <translation>0.96 1.245 0</translation>
  <diffusecolor>0.36 0.25 0.0667</diffusecolor>
 </Geom>
</Body>
<Body name="door" type="dynamic">
 <Geom type="box">
  <extents>0.3 1.025 0.0174</extents>
  <translation>0 1.025 0</translation>
  <diffusecolor>0.818 0.6157 0.0941</diffusecolor>
 </Geom>
 <Geom type="cylinder">
  <RotationAxis>1 0 0 90</RotationAxis>
  <radius>0.03</radius>
  <height>0.005</height>
  <translation>-0.25 0.95 -0.02</translation>
 </Geom>
</Body>
<Body name="handle" type="dynamic">
 <Translation>0.15 0 -0.02</Translation>
 <Geom type="cylinder">
  <RotationAxis>1 0 0 90</RotationAxis>
  <radius>0.015</radius>
  <height>0.08</height>
  <translation>-0.4 0.95 -0.0374</translation>
 </Geom>
 <Geom type="box">
  <translation>-0.35 0.95 -0.0704</translation>
  <extents>0.05 0.015 0.007</extents>
 </Geom>
 <Geom type="box">
```

```
  <translation>-0.31 0.95 -0.049</translation>
  <extents>0.014 0.01 0.03</extents>
 </Geom>
</Body>
<Joint name="doorhinge" type="hinge">
 <body>frame</body>
 <body>door</body>
 <anchor>0.3 0 -0.0348</anchor>
 <axis>0 -1 0</axis>
 <limitsdeg>0 120</limitsdeg>
</Joint>
<Joint name="handlehinge" type="hinge">
 <body>door</body>
 <body>handle</body>
 <anchor>-0.4 0.95 0</anchor>
 <axis>0 0 -1</axis>
 <limitsdeg>0 0</limitsdeg>
</Joint>
</KinBody>
<KinBody name="floor">
<Translation>0 0 -.010</Translation>
<RotationAxis>0 1 0 90</RotationAxis>
<Body type="static">
 <Geom type="box">
  <extents>0.005 1.500 2.000</extents>
  <diffuseColor>.3 1 .3</diffuseColor>
  <ambientColor>0.3 1 0.3</ambientColor>
 </Geom>
</Body>
</KinBody>
</Environment>
```