ABSTRACT

Title of the Thesis: DEMONSTRATING COGNITION BY TASK EXECUTION AND MOTION PLANNING WITH DIFFERENT ALGORITHMS FOR MANIPULATION

Dimitrios Dimitriadis, Master of Science, 2018

Thesis directed by: Professor John S. Baras
Department of Electrical and Computer Engineering

In this Thesis we demonstrate the whole path until the manipulation and the planning of the Baxter Robot. We start by analyzing the kinematic analysis of a six degrees of freedom robot. We build our analysis starting from the Denavit-Hartenberg method. We proceed with the kinematic equations of the robot and with the inverse kinematics as well as with a kinematic simulation of its movement with matlab. In order to reach our final goal we continue with the kinematic and dynamic analysis of the Baxter robot. We again state the Denavit-Hartenberg matrix, but this time we continue by building the dynamic model of the Baxter robot through the Euler-Lagrange equations. Moving on, we explore planning algorithms. The knowledge of which will help us in order to finally be able to formulate our path planner for the Baxter robot. We experiment ourselves by implementing four plan-

ning algorithms in different path planning problems. We construct the RRT and the RRT* algorithms in Python and we process them in different planning problems. Moving on, we also implement a planning problem in which Q-Learning and Sarsa algorithms are being used. We demonstrate how those two planning and learning algorithms work in our specified problem and we compare our results. Having knowledge on dynamic and kinematic robotic analysis and planning and motion planning algorithms we then experiment ourselves with the Baxter simulator on Gazebo. Also we plan the Baxter robot with Moveit!, getting familiar with the use of ROS as well as with the software. We add obstacles in our world and we plan our Baxter robot measuring its speed. We finally build a different plan algorithm RRT† by focusing on searching for a secure and realizable path plan starting from the lower dimension space and then adding degrees of freedom to our Baxter robot. Concluding, we have built the desired steps for someone in order to build up the required knowledge to deal with robots and artificial intelligence planning.

DEMONSTRATING COGNITION BY TASK EXECUTION AND MOTION PLANNING WITH DIFFERENT ALGORITHMS FOR MANIPULATION

by

Dimitrios Dimitriadis

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2018

Advisory Committee:
Professor John S. Baras, Chair/Advisor
Professor Yiannis Aloimonos
Professor Nikhil Chopra

Dedication

Να πάρουμε το λάθος αγκαλιά

Let's embrace our mistakes.

In memory of Andreas Iwannou Kassetas (1944-2015)

Dedication

To my mother Antigone, my father Konstantinos and my brother Bill for their invaluable love, support and inspiration throughout my studies. Their support has been inexhaustible.

Acknowledgments

At this point I would like to express my thanks to my supervisor, Professor John S. Baras, for his continuing support throughout my studies as his student. He was patiently providing me with guidance and was always encouraging me with his advice. I mostly feel gratitude for the time that he spent with me, especially when we were talking about some real-world problems. I feel extremely lucky that I had a supervisor who cared so much about my work, and who responded to my questions and queries so promptly. I would also like to thank him, because he trusted me for this work, and he was very helpful providing me his valuable knowledge to all the problems that I encountered. His energy and enthusiasm combined with a deep knowledge of mathematical theory is what I will always admire and remember for the years to come. I have to admit that I have been lucky to have an advisor that was a constant motivation for me. My years of collaboration with Prof. Baras, have been the most fruitful periods of my life. If it was not for Prof. Baras I would not have been able to fullfil my dream and study abroad and have the opportunity to experiment myself with the Baxter robots and work at the ARC Lab.

I would also like to express my special thanks to Dr. Yiannis Aloimonos and Dr. Cornelia Fermüller for their persistent help during my studies. Professor Aloimonos has always been supporting me through my hard times. The discussions that we had where most of the times a solution to my problems and to everything that was troubling me. I am also grateful to Professor Kikhil Chopra for his efforts to serve on my thesis committee. I would also like to thank my Professor Mark D. Fuge for his invaluable insights and for providing me with knowledge on the fields of machine learning and artificial intelligence.

Moreover I feel the need to express my gratitude to my Professors George Fikioris, Panagiotis Cottis, Symeon Papavasileiou, Ioannis Sarantopoulos and Argyris Soldatos from the National Technical University of Athens, because without them, I wouldn't have the chance of reaching this level and studying at the University of Maryland. I would also like to thank all the members of staff from the Maryland University who helped me in my supervisors absence. Especially Ms. Kim Edwards for her great administrative support and help with every bureaucratic issue that I faced. I feel the need to express my greatest gratitudes to my great high school Physics teacher Mr. Andreas Kassetas who has passed away as a real fighter. He will always be remembered as one of the greatest physicists of Greece, not only because of his knowledge, but surely because of his personality.

Many thanks to my fellow student and colleague Michael Karotsieri. Our numerous hours of studying and cooperation are the greatest supplies for my life. I also want to thank the graduate alumni student of our department Evripidis Paraskevas for his support throughout my studies. Many thanks also to the graduate alumnies Christoforos Somarakis, Ren Mao and Wentao Luan and Zois Tsinas. Special thanks to my fellow graduate students Ioannis Demertzis, Lida Apergi, Konstantinos Zampogiannis, Dipankar Maity and Proloy Das. I would also like to thank my friends Jason Thomopoulos, Antonis Xenakis, Iason Papanikolaou and Michael Tsapos that helped me with my familiarity with aryland and the United States.

I would like to acknowledge the support offered by the DARPA STTR (through Anthro-Tronix) grant UMD09142016, also by the DARPA STTR (through Boston Engineering) grant W31P4Q-13-C-0136, by the National Science Foundation (NSF-CPS) grant CNS-

1544787 and also by the National Science Foundation (NSF-CPS) grant 1655009.

Last, the biggest thanks from the bottom of my heart to my family who have been helping, supporting me and advising me to take the right decisions throughout my life. Their moral support and their way of raising me as a person cannot be neither redeemed nor explained it in any way.

Table of Contents

Chapter 1: Introduction

Robotics main idea is based on the fact that machines could assist humans or replace them in professional situations too risky for them and optimally to perform tasks autonomously.

1.1 History and Motivation

First thing one needs to know when it comes to robotics, is what is the true meaning of robotics. To understand that we should investigate the semantics of the word. So, robot is a word first introduced to the world by the Czech novelist Karel Capek at 1920, and the translation in Czech means worker or servant. Therefore robots should be in the use for serving the humans. Actually one can find laws under which robots are obliged to function in the real world, as they exist. The official definition of what a robot is, was settled in 1980 from the Robot Institute of America (RIA) and stands for:

"A robot is a reprogrammable, multifunctional manipulator designed to move material, parts, tools, or specialized devices through variable programmed motions for the performance of a variety of tasks."

The key components of a robot are the sensors, the power conversion units, the actuators, the controllers, the user interface and the linkage base [20],[18], [6]. Nowadays there are a variety of robots in a variety of work sectors. We meet robots in the industry, in space, in hazardous environments, in medicine,in military, and of course at our homes.

1.2 The Baxter robot

Baxter robot [1] is a humanoid robot. Humanoid robots are those whose body shape resemble the human body. As a humanoid robot Baxter can execute a variety of tasks and of course it can be used in human-robot collaborations (HRC). This means that either the required knowledge of how to perform a task is being provided to the robot suppse as a specified algorithm, or with the use of computer vision the humans demonstrate the execution of a task while the robot observes and meanwhile extracts the important aspects and afterwards imitates the human. As an anthropomorphic robot Baxter has two arms each of which has seven degrees of freedom. In the following parts, when we say manipulator, it refers to the arm of Baxter. The Baxter robot is useful for experimenting with the controls at every joint of the robot and of course for human robot interaction as it is armed with buttons and knobs that make this interaction easier. Baxter supports also computer vision applications and also its arm is useful for planning and manipulating. We will focus on the dynamic modeling of the Baxter robot with particular focus on the Newton-Euler formulation. Also, we will be planning the Baxter's arm. A remarkarble and popular software library and toolbox that is used for planning the Baxter robot is called Moveit! as we will see later on. Nowadays Baxter, despite its high cost it continues to be widely used

especially for research purposes.

Figure 1.1: The Baxter robot

1.3 Software

Multiple computer programs and softwares have been used throughout this thesis assignment. Following is a short description of this software and the area of utilization.

Mathematica

Mathematica is a modern technical computing system which is really useful when it comes to mathematical computations. Of course it is also used in a variety of areas such as machine learning, algebra, geometry and many more. It uses its own kind of ”language”

which is user friendly, and that is the main reason that we used it in many of our mathematical calculations.

Matlab

Matlab [17] is a software simulator and also a numerical computing environment like Mathematica. It is developed by Mathworks and mostly used for performing computationally tasks much faster compared to traditional program languages. Although Matlab is intended primarily for numerical computing, an additional package called Simulink is especially used for multidomain simulation and Model-Based Design for dynamic and embedded systems. A great advantage of high-level language is that allows interfacing with programs written in Python, C, C++ and more. Of course Matlab offers access to symbolic computing abilities, to matrix manipulations, algorithmic implementations as well as plotting functions and data. We used Matlab for simulating one of our robotic arm motions.

Python

Python is a high-level programming language which can be used for a wide variety of applications. Python is open source making it free to use. The advantage of Python is that it has a user friendly syntax and still can power some really complex applications. Having learned C as a first base programming language, Python is a logical step, because Python is written in C. Nowadays Python is considered to be one of the most widely used programming languages. This actually makes Python the first choice when it comes up to which

programming language should be used in order to implement a solution to a problem or a standalone application. This happens also because there is already a strong documentation, tutorials, guides and examples that one can find useful on the web.

C, C++

C is one of the most basic programming languages which supports structured programming. C was originally developed and used to re-implement the Unix system. It has become one of the most powerful and widely used programming languages of all time. It forms the core of the modern languages like C++.

C is a very basic language making all the knowledge you acquire in C transferable to the future languages. The main reason one should use it is because it allows you direct control over hardware.

Lastly, program execution in C is fast because the language is compiled and turned into machine code. Programs in C are very closely related to Matlab making the transition easy. C++ is a highly portable programming language, which is object-oriented and includes classes, data abstraction and encapsulation. It has imperative and generic programming features and also provides features for a low-level manipulation. C++ is powerful fast and efficient. Both C and C++ where developed at Bell labs and C++ can be considered as an extension of C which provides more high-level features for program organization.

ROS

The Robot Operating System (ROS) is not an actual operating system, but a flexible framework that provides the functionality of an operating system for writing robot software. We can say it is a collection of software frameworks specialized for robot software development. Therefore ROS can be integrated with real-time code, but still is not considered to be an OS. ROS services provide design for heterogeneous clustering, like package management and low-level device control. Also the main ROS client libraries are C++(roscpp) and Python(rospy) and those are geared towards a Unix-like system. ROS client library implementations such as roscpp and rospy packages contain application-related code which uses one or more ROS client libraries.

Gazebo

Gazebo [4] is a set of ROS packages used for a robot simulator in a 3-dimensional world. In our case we used Gazebo for simulating the Baxter robot. With the use of Gazebo you can simulate the interaction of the robot with objects as well as study and experiment with the controls of the robot.

MoveIt!

MoveIt! [25] is an open source software for manipulation. Nowadays for robotics it is the state of the art software for robot manipulation, incorporating the latest advances in motion planning and 3D perception. Moreover includes kinematics, controls and navigation. It provides a great platform for developing advanced robotics applications and evaluating

new robot designs as well as planning and executing path planning algorithms. For the aforementioned reasons we used it for our work.

1.4 Background Theory

1.4.1 End effector

In robotics, the end effector is the final action element on the robot, meaning the last link or the device at the end of a robotic arm which is designed to interact with the environment. The end effector's nature depends on the application of the robot. Additionally, in our work whenever we mention the end effector, we always refer to the gripper.

1.4.2 The Tool Frame - TCP

To define the Tool Frame, we first need a reference point in the world coordinate system, and in our case is indicated by the coordinate system of our end effector. Let Ω be the space that contains all the homogeneous matrices. Then :

$$A : SE(3) \Rightarrow \Omega$$

$$T(R,t) = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix}$$

where $SE(3)$ is the Euclidean group which is used for the kinematics of our robot. $SO(3)$ is the group of all rotations about the origin of the three-dimensional Euclidean

space R^3. $R = (x\ y\ z)$ is the rotation matrix $R \in SO(3)$ and $\mathbf{t} = (t_x, t_y, t_z) \in \Re^3$ the position vector and x, y, z are the orthogonal basis vectors.

1.4.3 Kinematic chains

In robotics, the separate rigid bodies that are called links, are connected with each other via joints and this way they form a so called kinematic chain. There are 2 kind of joints in robotics: the revolute ones and the prismatic ones [6]. A prismatic joint allows the two attached links to have a linear motion. A revolute joint allows a relative motion between the two connected links. Each joint has a single degree of freedom and is often represented by the single joint variable q_i. Below in Figure 1.2 are presented those two types of joints.

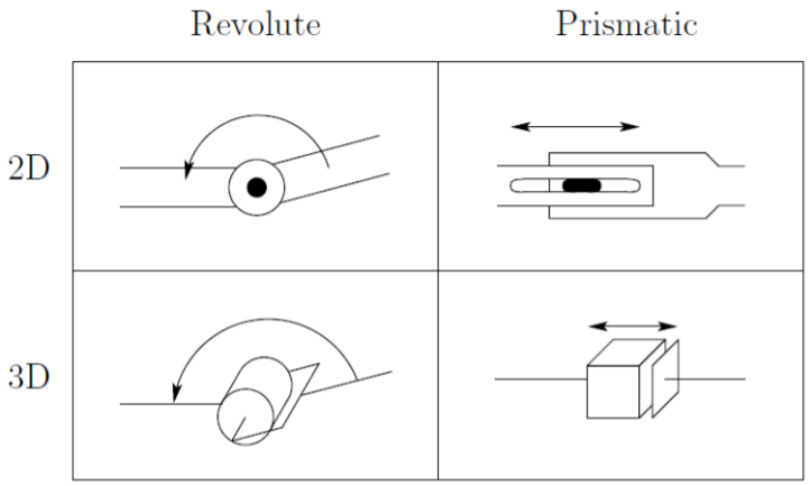


Figure 1.2: A symbolic representation of the two types of robotic joints

1.4.4 Jacobian Matrix computation

The Jacobian matrix is often used in the kinematics and the dynamics of robotic systems and is really important to robotics and control theory. It relates differences between

two different representations of a system, meaning from joint to configuration space, while representing an approximation in the context of finite differences. If we assume that we have a really simple kinematic chain with one degree of freedom, meaning with only two links, then the current position of our robotic arm can either be described from the orientation and the position of our robots final action element or from a set of joint angles. We usually denote the position of our robot's final action element with **x** and the joint angles with **q**. Now what Jacobian does is that it relates the movement of **x** as a consequence of the movement of the elements of **q**. The Jacobian is the matrix which includes all the first-order partial derivatives of our vector-valued function, meaning :

$J = \frac{\vartheta x}{\vartheta q} \Rightarrow$

and by using the mathematical chain rule for partial derivatives we get:

$\dot{x} = J \cdot \dot{q}$

1.4.5 The Inertia Tensor

The moment of inertia is the value that determines the resistance that an object has regarding rotation changes. It is also known as rotational inertia. Basically it depends on the object's mass distribution as well as the axis chosen. If the mass distribution of our object is symmetric with respect to the attached frame, then the inertia tensor is diagonal(cross products of inertia are identically zero). If the rotation axis is not given, then one can generalize the scalar moment of inertia to a 3×3 matrix which will be the moment of inertia about an arbitrary axis. This matrix is the inertia tensor matrix [6]. If we let the mass density of an object to be represented as a function of position $p(x,y,z)$, then the

inertia tensor is a frame attached to the center of mass of an object and is computed as:

$$I = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix}$$

where the diagonal elements Ixx, Iyy, Izz are the principal moments of inertia and the off-diagonal terms are the cross products of inertia, with:.

$$\begin{aligned} I_{xx} &= \int\int\int (y^2 + z^2) p(x,y,z) dxdydz \\ I_{yy} &= \int\int\int (x^2 + z^2) p(x,y,z) dxdydz \\ I_{zz} &= \int\int\int (x^2 + y^2) p(x,y,z) dxdydz \\ I_{xy} &= I_{yx} \\ &= \int\int\int xyp(x,y,z) dxdydz \\ I_{yz} &= I_{zy} \\ &= \int\int\int yzp(x,y,z) dxdydz \\ I_{zx} &= I_{xz} \\ &= \int\int\int zxp(x,y,z) dxdydz \end{aligned}$$

Chapter 2: A six degrees of freedom robotic kinematic analysis and simulation

2.1 The KR 360 FORTEC

Specification of the movement of a robot so that our robot's end-effector achieve the desired task is known as motion planning. We will be dealing with motion planning in this section and in the following ones. We simulated the kinematics of the KR 360 FORTEC robotic arm. The KR 360 FORTEC is a robot with six axis, meaning six degrees of freedom. This robot is being manufactured by KUKA and is an industrial robot particularly suited to handle heavy assemblies. We are not interested in the robot itself, we just used it to demonstrate a kinematic robotic simulation via Matlab, following our kinematic analysis on it. Here we have to mention also that all degrees of freedom are still modeled in this thesis. Also some definitions and some parameters are modified to better suit our complete model. The KR 360 FORTEC robot is being presented in Figure 2.1:

Figure 2.1: The KR 360 FORTEC from both sides

2.2 Theoretical Analysis

We begin our forward kinematic analysis of our robot with the calculation of the Denavit-Hartenberg parameter array of the robotic arm, with the use of the Figure 2.2 and Figure 2.3 of our robot:

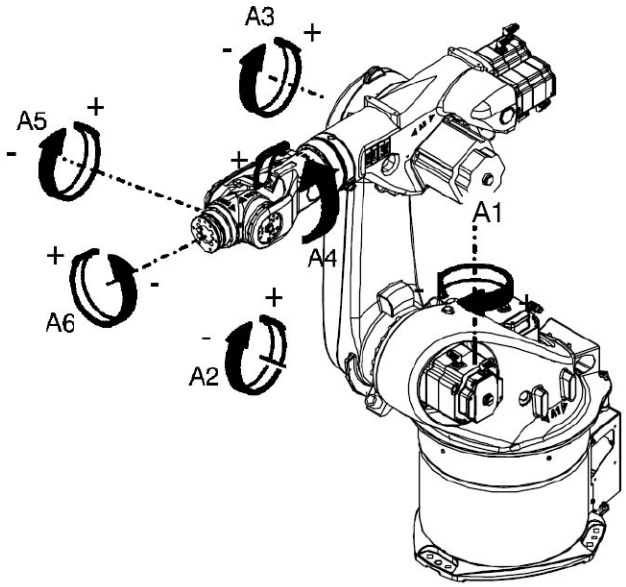


Figure 2.2: Direction of the rotations of the Robotic Arms of our robot(The six degrees of freedom)

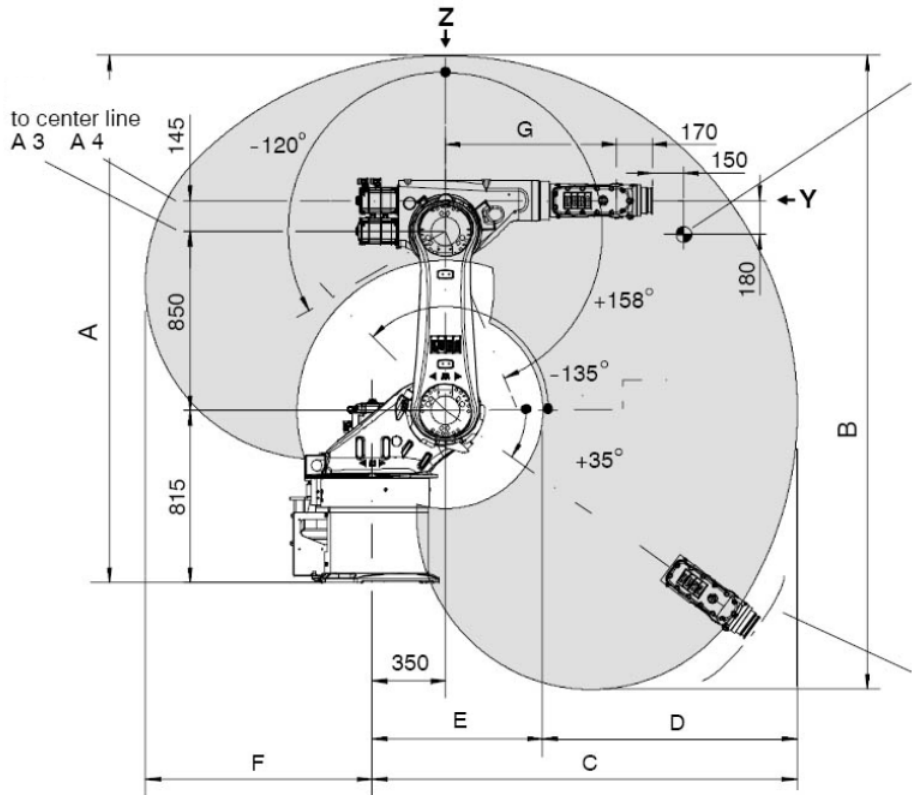


Figure 2.3: Design of the lengths of our robot joints

Initially we will describe the geometrical structure of the robot with the Denavit-Hartenbeg table, the initial idea of which is based on the use of 4 parameters for the relative placement of the frame (i) towards the frame (i-1). Those are the angle α, the transpositions a and d and the angle θ. More specifically, d is the offset along previous z to the common normal, θ is the angle about previous z, from the old x to the new x, a is the length along the previous x to the common normal and α is the angle about the common normal, from old z axis to the new z axis.

D-H Parameters of the Baxter Robot				
Link	θ	d(mm)	a(mm)	α(rad)
1	θ_1	815	350	$+\frac{\pi}{2}$
2	$\theta_2 + \frac{\pi}{2}$	0	850	0
3	θ_3	0	145	$+\frac{\pi}{2}$
4	θ_4	G	0	$-\frac{\pi}{2}$
5	θ_5	0	0	$+\frac{\pi}{2}$
6	θ_6	170+100	0	0

The pose of the tool center point coordinate system or TCP is computed by the forward kinematics $T(q)$ of our six degrees of freedom robot in a configuration q.In general the TCP is defined as the coordinate system A_{TCP}^{base}, where base defines the robot arm coordinate system $T(q)$ can be computed from the following equation:

$$T(q) = A_{TCP}^{Base} = A_0^{Base} \cdot A_2^1(q_1)...A_n^{n-1}(q_{(n-1)}) \cdot A_{TCP}^n \tag{2.1}$$

where all the link transformation matrices A_i^{i-1} are being computed using the link's DH-parameters.

In our case according to the above parameters we calculated the following matrices which were used for the finding of the kinematic equation.

$$T(q) = A_1^0(q_1) \cdot A_2^1(q_2) \cdot A_3^2(q_3) \cdot A_4^3(q_4) \cdot A_5^4(q_5) \cdot A_6^5(q_6) \tag{2.2}$$

The matrix A_i^{i-1} represents the position and the orientation of the frame i in relation with the frame i-1. The first three columns of the A_i^{i-1} matrix contain the directional cosine of the frame i, whereas the 4th column represents the position of our start coordinate frame O_0.

Below are our computations (Our calculations were made with the use of Mathematica) for all the A_i^{i-1} matrices for each one of the rotors of our robot:

$A_1^0(q_1) = Tra(z,815) \cdot Rot(z,\theta_1) \cdot Tra(x,350) \cdot Rot(x,90)$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 815 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} cos\theta_1 & -sin\theta_1 & 0 & 0 \\ sin\theta_1 & cos\theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 350 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Therefore:

$$A_1^0(q_1) = \begin{bmatrix} cos\theta_1 & 0 & sin\theta_1 & 350 \\ sin\theta_1 & 0 & -cos\theta_1 & 0 \\ 0 & 1 & 0 & 815 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

Likewise we compute the rest A_i^{i-1} matrices for $i = 2,3,4,5,6$ and we get the following results:

$$A_2^1(q_2) = Rot(z, \theta_2 + 90) \cdot Tra(x, 850)$$

$$A_2^1(q_2) = \begin{bmatrix} sin\theta_2 & -cos\theta_2 & 0 & 850 \\ cos\theta_2 & -sin\theta_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

Moving on from joint 2 to 3 we get:

$$A_3^2(q_3) = Rot(z, \theta_3) \cdot Tra(x, 145) \cdot Rot(x, 90)$$

$$A_3^2(q_3) = \begin{bmatrix} cos\theta_3 & 0 & sin\theta_3 & 145 \\ sin\theta_3 & 0 & cos\theta_3 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

From joint 3 to 4 we have:

$$A_4^3(q_4) = Tra(z, G) \cdot Rot(z, \theta_4) \cdot Rot(x, -90)$$

$$A_4^3(q_4) = \begin{bmatrix} cos\theta_4 & 0 & -sin\theta_4 & 0 \\ sin\theta_4 & 0 & cos\theta_4 & 0 \\ 0 & -1 & 0 & G \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

where G is a constant.

From rotors 4 to 5 we get:

$$A_5^4(q_5) = Rot(z,\theta_5) \cdot Rot(x,90)$$

$$A_5^4(q_5) = \begin{bmatrix} cos\theta_5 & 0 & -sin\theta_5 & 0 \\ sin\theta_5 & 0 & cos\theta_5 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.7}$$

Finally from joint 5 to our final rotor 6 or E (we symbolize it as E because it is our end effector, meaning the device at the end of our robotic arm), we get:

$$A_6^5(q_6) = Tra(z,270) \cdot Rot(z,q_6)$$

$$A_6^5(q_6) = \begin{bmatrix} cos\theta_6 & sin\theta_6 & 0 & 0 \\ sin\theta_6 & cos\theta_6 & 0 & 0 \\ 0 & 0 & 1 & 270 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.8}$$

After having calculated all our coordinate transformation matrices, we then calculate the coordinate frames with reference to O_0 of our robot. Moving on, we calculate the ori-

entation of our final action element, or else our end effector with respect to the base of our robot. Based on the eq. (2.1) we will have:

$$A_2^0(q_1,q_2) = A_1^0(q_1)\cdot A_2^1(q_2)$$
$$A_3^0(q_1,q_2,q_3) = A_2^0(q_1,q_2)\cdot A_3^2(q_3)$$
$$A_4^0(q_1,q_2,q_3,q_4) = A_3^0(q_1,q_2,q_3)\cdot A_4^3(q_4)$$
$$A_5^0(q_1,q_2,q_3,q_4,q_5) = A_4^0(q_1,q_2,q_3,q_4)\cdot A_5^4(q_5)$$
$$A_6^0(q_1,q_2,q_3,q_4,q_5,q_6) = A_5^0(q_1,q_2,q_3,q_4,q_5)\cdot A_6^5(q_6)$$

Ending up to:

$$A_6^0 = A_E^0 = \begin{bmatrix} -cos\theta_1 sin(\theta_2+\theta_3+\theta_5) & -sin\theta_1 & cos\theta_1 cos(\theta_2+\theta_3+\theta_5) & A \\ -sin\theta_1 sin(\theta_2+\theta_3+\theta_5) & cos\theta_1 & 0 & B \\ cos(\theta_2+\theta_3+\theta_5) & 0 & sin(\theta_2+\theta_3+\theta_5) & C \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.9)$$

where,

$$A = 350cos\theta_1 - 850cos\theta_1 sin\theta_2 - 145cos\theta_1 sin(\theta_2+\theta_3) + 820cos\theta_1 cos(\theta_2+\theta_3) + 270cos\theta_1 cos(\theta_2+\theta_3+\theta_5)$$

$$B = 350sin\theta_1 - 850sin\theta_1 sin\theta_2 - 145sin\theta_1 sin(\theta_2+\theta_3) + 820sin\theta_1 cos(\theta_2+\theta_3) + 270sin\theta_1 cos(\theta_2+\theta_3+\theta_5)$$

$$C = 815 + 850cos\theta_2 + 145cos(\theta_2+\theta_3) + 820(\theta_2+\theta_3) + 270sin(\theta_2+\theta_3+\theta_5)$$

In our above calculations we used the well known trigonometric identities:

i.)$cos(\theta_2 + pi/2) = -sin(\theta_2)$

ii.)$sin(\theta_2 + pi/2) = cos(\theta_2)$

iii.)$cos\theta_1 cos\theta_2 - sin\theta_1\theta_2 = cos(\theta_1 + \theta_2)$

iv.)$cos(\theta_1 + \theta_2)cos\theta_3 - sin(\theta_1 + \theta_2)sin\theta_3 = cos(\theta_1 + \theta_2 + \theta_3)$

v.)$sin\theta_1 cos\theta_2 + sin\theta_2 cos\theta_2 = sin(\theta_1 + \theta_2)$

vi.)$sin(\theta_1 + \theta_2)cos\theta_3 + sin\theta_3 cos(\theta_1 + \theta_2) = sin(\theta_1 + \theta_2 + \theta_3)$

2.3 Forward differential model

Starting our theoretical approach we assume dx_E the 3×1 infinitely small transpose vector, meaning $dx_E = [dx \quad dy \quad dz]^T$ and $d\phi_E$ the 3×1 infinite small rotating vector of our end effector towards our coordinate system $O_{ox_oy_oz_o}$ base. If we symbolize:

$$dp = \begin{bmatrix} dx_E \\ \\ d\phi_E \end{bmatrix} \tag{2.10}$$

Then if we divide both members of the equation by dt, we get:

$$\dot{p} = \begin{bmatrix} v_E \\ \\ \omega_E \end{bmatrix} \tag{2.11}$$

where v_E and ω_E stand for the linear and the angular velocity respectively. If J is the Jacobian matrix of the partial derivatives of the dx_E and $d\phi_E$ with respect to the variables of the joints, which represents the differential relationship between the joints displacements

and our end effector [28], we then have:

$$\dot{p} = J \cdot \dot{q} \tag{2.12}$$

where $\dot{q} = [\dot{q_1}, \dot{q_2}, \dot{q_3}, \dot{q_4}, \dot{q_5}, \dot{q_6}]^T$ is the 6×1 vector of the velocity of our joints. Unfolding the eq. (2.12) it becomes:

$$\begin{bmatrix} v_{E_x} \\ v_{E_y} \\ v_{E_z} \\ \omega_{E_x} \\ \omega_{E_y} \\ \omega_{E_z} \end{bmatrix} = J \cdot \begin{bmatrix} \dot{q_1} \\ \dot{q_2} \\ \dot{q_3} \\ \dot{q_4} \\ \dot{q_5} \\ \dot{q_6} \end{bmatrix} \tag{2.13}$$

The first three rows of our Jacobian matrix correspond to the linear velocity v_E and the last three to the angular velocity ω_E of our end effector. Every column of our Jacobian matrix represents the linear and the angular velocity that is caused due to each joint. If J_{L_i} and J_{A_i} are the 3×1 vectors of our Jacobian matrix that correspond to our linear and angular velocities correspondingly, then our Jacobian matrix will be of the form:

$$J = \begin{bmatrix} J_{L_1} & J_{L_2} & J_{L_3} & J_{L_4} & J_{L_5} & J_{L_6} \\ J_{A_1} & J_{A_2} & J_{A_3} & J_{A_4} & J_{A_5} & A_{L_6} \end{bmatrix} \tag{2.14}$$

Therefore the linear velocity of our end effector will be:

$$v_E = J_{L_1} \cdot \dot{q_1} + J_{L_2} \cdot \dot{q_2} + \cdots J_{L_6} \cdot \dot{q_6} \tag{2.15}$$

If the joint i is prismatic it will effect the linear velocity of our end effector towards the direction of the joint axis. If b_{i-1} is the unitary vector along the axis of the joint i and $\dot{d_i}$ the scalar linear velocity of the prismatic joint (we symbolize it that way, because of our Denavit-Hartenberg matrix symbolization) then following [28]:

$$J_{L_i} \cdot \dot{q_i} = b_{i-1} \cdot \dot{d_i} \tag{2.16}$$

The above equation gives us the solution to the answer to the question what is the linear velocity of the end effector, when the joint is prismatic. Now if the joint is a revolute one, then it rotates the whole system of our robot joints starting from itself i up until to the final action element of the robot with angular velocity:

$$\omega_i = b_{i-1} \cdot \dot{\theta_i} \tag{2.17}$$

In fact this angular velocity causes a corresponding linear velocity to the end effector. Let $r_{i-1,E}$ the vector of the position of the end effector from the point O_{i-1}, then the linear velocity of the end effector which is caused from the angular velocity of the ω_i will be:

$$J_{L_i} \cdot \dot{q_i} = \omega_i \times r_{i-1,E} = (b_{i-1} \times r_{i-1,E}) \cdot \dot{\theta_i} \tag{2.18}$$

The above equation gives us the linear velocity of the end effector if our joint is a revolute one.

Working in the same direction the angular velocity of our end effector will be:

$$\omega_E = J_{A_1} \cdot \dot{q_1} + J_{A_2} \cdot \dot{q_2} + \cdots J_{A_6} \cdot \dot{q_6} \tag{2.19}$$

In this case of course it is obvious that if the joint is a prismatic one it can not cause any angular velocity to the end effector. Hence:

$$J_{A_i} \cdot \dot{q_i} = 0 \tag{2.20}$$

In the case of a revolute joint, then the angular velocity that is being caused to our gripper at the end of our robotic arm is:

$$J_{A_i} \cdot \dot{q_i} = \omega_i = b_{i-1} \cdot \dot{\theta}_i \tag{2.21}$$

Concluding all the above, the columns of our Jacobian matrix are given by:

$$\begin{bmatrix} J_{L_i} \\ \\ J_{A_i} \end{bmatrix} = \begin{bmatrix} b_{i-1} \\ \\ 0 \end{bmatrix} \tag{2.22}$$

for a prismatic joint, and:

$$\begin{bmatrix} J_{L_i} \\ J_{A_i} \end{bmatrix} = \begin{bmatrix} b_{i-1} \times r_{i-1,E} \\ b_{i-i} \end{bmatrix} \tag{2.23}$$

for a revolute joint.

As we said before b_{i-1} is the unitary vector along the axis of the joint i and $r_{i-1,E}$ is the vector from O_{i-1} to O_E. Both b_{i-1} and $r_{i-1,E}$ are functions of the transposition of the joints and can be computed through coordinate transformations. The direction of the $i-1$ axis with regard to frame $i-1$ is presented with $\vec{b} = [0 \quad 0 \quad 1]^T$ because the axis has the direction of the z_{i-1} axis. $\vec{b}$ can be transformed to a well defined vector related to the base frame with the help of the 3×3 rotation matrices of our robot system $R_i^{i-1}(q_i)$ (This R_i^{i-1} 3×3 matrix is the rotation matrix inside the A_i^{i-1} matrix that we have already calculated) [28]. Therefore:

$$\begin{aligned} b_{i-1} &= R_1^0(q_1) \cdot \cdots \cdot R_{i-1}^{i-2}(q_{i-1}) \cdot \vec{b} \\ &= R_{i-1}^0(q_1, ..., q_{i-1}) \cdot \vec{b} \end{aligned} \tag{2.24}$$

The position vector $r_{i-1,E}$ can be computed with the help of the $A_i^{i-1}(q_i)$ matrices and will be:

$$r_{i-1,E} = A_n^0(q_1, ..., q_i - 1) \cdot \vec{r} - A_{i-1}^0(q_1, ..., q_{i-1}) \tag{2.25}$$

where $\vec{r} = [0 \quad 0 \quad 0 \quad 1]^T$

Therefore in order to achieve our initial goal which was to compute the Jacobian matrix,

we have to compute the b_{i-1} and the $r_{i-1,E}$ for all i's. Based on the eq. (2.25) we start our calculations for the $r_{i-1,E}$ for $i = 1, 2, \cdots 6$. Beginning with i=1 we have:

$$\begin{bmatrix} r_{0,E} \end{bmatrix} = \begin{bmatrix} A_1 \\ B_1 \\ C_1 \end{bmatrix} \tag{2.26}$$

where,

$$A_1 = 350cos\theta_1 - 850cos\theta_1 sin\theta_2 - 145cos\theta_1 sin(\theta_2 + \theta_3) + 820cos\theta_1 cos(\theta_2 + \theta_3) + 270cos\theta_1 cos(\theta_2 + \theta_3 + \theta_5)$$

$$B_1 = 350sin\theta_1 - 850sin\theta_1 sin\theta_2 - 145sin\theta_1 sin(\theta_2 + \theta_3) + 820sin\theta_1 cos(\theta_2 + \theta_3) + 270sin\theta_1 cos(\theta_2 + \theta_3 + \theta_5)$$

$$C_1 = 815 + 850cos\theta_2 + 145cos(\theta_2 + \theta_3) + 820(\theta_2 + \theta_3) + 270sin(\theta_2 + \theta_3 + \theta_5)$$

For i=2 we get:

$$\begin{bmatrix} r_{1,E} \end{bmatrix} = \begin{bmatrix} A_2 \\ B_2 \\ C_2 \end{bmatrix} \tag{2.27}$$

where,

$$A_2 = -850cos\theta_1 sin\theta_2 - 145cos\theta_1 sin(\theta_2 + \theta_3) + 820cos\theta_1 cos(\theta_2 + \theta_3) + 270cos\theta_1 cos(\theta_2 + \theta_3 + \theta_5)$$

$$B_2 = -850sin\theta_1 sin\theta_2 - 145sin\theta_1 sin(\theta_2 + \theta_3) + 820sin\theta_1 cos(\theta_2 + \theta_3) + 270sin\theta_1 cos(\theta_2 + \theta_3 + \theta_5)$$

$$C_2 = 850cos\theta_2 + 145cos(\theta_2 + \theta_3) + 820(\theta_2 + \theta_3) + 270sin(\theta_2 + \theta_3 + \theta_5)$$

For i=3 we have:

$$\begin{bmatrix} r_{2,E} \end{bmatrix} = \begin{bmatrix} A_3 \\ B_3 \\ C_3 \end{bmatrix} \tag{2.28}$$

where,

$$A_3 = -145cos\theta_1 sin(\theta_2+\theta_3) + 820cos\theta_1 cos(\theta_2+\theta_3) + 270cos\theta_1 cos(\theta_2+\theta_3+\theta_5)$$

$$B_3 = -145sin\theta_1 sin(\theta_2+\theta_3) + 820sin\theta_1 cos(\theta_2+\theta_3) + 270sin\theta_1 cos(\theta_2+\theta_3+\theta_5)$$

$$C_3 = 145cos(\theta_2+\theta_3) + 820(\theta_2+\theta_3) + 270sin(\theta_2+\theta_3+\theta_5)$$

For i=4 we compute:

$$\begin{bmatrix} r_{3,E} \end{bmatrix} = \begin{bmatrix} 820cos\theta_1 cos(\theta_2+\theta_3) + 270cos\theta_1 cos(\theta_2+\theta_3+\theta_5) \\ 820sin\theta_1 cos(\theta_2+\theta_3) + 270sin\theta_1 cos(\theta_2+\theta_3+\theta_5) \\ 820(\theta_2+\theta_3) + 270sin(\theta_2+\theta_3+\theta_5) \end{bmatrix} \tag{2.29}$$

For i=5 we get:

$$\begin{bmatrix} r_{4,E} \end{bmatrix} = \begin{bmatrix} 270cos\theta_1 cos(\theta_2+\theta_3+\theta_5) \\ 270sin\theta_1 cos(\theta_2+\theta_3+\theta_5) \\ 270sin(\theta_2+\theta_3+\theta_5) \end{bmatrix} \tag{2.30}$$

Finally for i=6 it is:

$$\begin{bmatrix} r_{5,E} \end{bmatrix} = \begin{bmatrix} 270cos\theta_1 cos(\theta_2+\theta_3+\theta_5) \\ 270sin\theta_1 cos(\theta_2+\theta_3+\theta_5) \\ 270sin(\theta_2+\theta_3+\theta_5) \end{bmatrix} \tag{2.31}$$

Moving on, from eq. (2.24) we then calculate our $b_{i-1}\ \forall\ i = 1,2\cdots 6$. For i=1:

$$\begin{bmatrix} b_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{2.32}$$

For i=2:

$$\begin{bmatrix} b_1 \end{bmatrix} = \begin{bmatrix} sin\theta_1 \\ -cos\theta_1 \\ 0 \end{bmatrix} \tag{2.33}$$

For i=3:

$$\begin{bmatrix} b_2 \end{bmatrix} = \begin{bmatrix} sin\theta_1 \\ -cos\theta_1 \\ 0 \end{bmatrix} \tag{2.34}$$

For i=4:

$$\begin{bmatrix} b_3 \end{bmatrix} = \begin{bmatrix} cos\theta_1 cos(\theta_2+\theta_3) \\ sin\theta_1 cos(\theta_2+\theta_3) \\ sin(\theta_2+\theta_3) \end{bmatrix} \tag{2.35}$$

For i=5:

$$\begin{bmatrix} b_4 \end{bmatrix} = \begin{bmatrix} sin\theta_1 \\ -cos\theta_1 \\ 0 \end{bmatrix} \tag{2.36}$$

Finally for i=6:

$$\begin{bmatrix} b_5 \end{bmatrix} = \begin{bmatrix} cos\theta_1 cos(\theta_2+\theta_3+\theta_5) \\ sin\theta_1 cos(\theta_2+\theta_3+\theta_5) \\ sin(\theta_2+\theta_3+\theta_5) \end{bmatrix} \tag{2.37}$$

Based on the above mathematical analysis from the equation eq. (2.22) we have, we calculate the 3×1 vectors of our Jacobian matrix which correspond to the angular velocities of our robot's end effector and those are the J_{A_i} matrices.

$$J_{A_1} = b_0 = \begin{bmatrix} b_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{2.38}$$

and

$$J_{A_2} = b_1 = \begin{bmatrix} \ b_2 \ \end{bmatrix} = \begin{bmatrix} sin\theta_1 \\ -cos\theta_1 \\ 0 \end{bmatrix} \tag{2.39}$$

and

$$J_{A_3} = b_2 = \begin{bmatrix} \ b_2 \ \end{bmatrix} = \begin{bmatrix} sin\theta_1 \\ -cos\theta_1 \\ 0 \end{bmatrix} \tag{2.40}$$

and

$$J_{A_4} = b_3 = \begin{bmatrix} \ b_2 \ \end{bmatrix} = \begin{bmatrix} sin\theta_1 \\ -cos\theta_1 \\ 0 \end{bmatrix} \tag{2.41}$$

and

$$J_{A_5} = b_4 = \begin{bmatrix} \ b_2 \ \end{bmatrix} = \begin{bmatrix} sin\theta_1 \\ -cos\theta_1 \\ 0 \end{bmatrix} \tag{2.42}$$

and finally

$$J_{A_E} = b_5 = \begin{bmatrix} b_2 \end{bmatrix} = \begin{bmatrix} sin\theta_1 \\ -cos\theta_1 \\ 0 \end{bmatrix} \tag{2.43}$$

Now based on eq. (2.23) we calculate the 3×1 vectors of our Jacobian matrix which correspond to the linear velocities of our robot's end effector and those are the J_{L_i} matrices. For $i = 1$ we have:

$$\begin{bmatrix} J_{L_1} \end{bmatrix} = b_0 \times r_{0,E} = \begin{bmatrix} AL_1 \\ BL_1 \\ 0 \end{bmatrix} \tag{2.44}$$

where,

$$\begin{aligned} AL_1 = & -350sin\theta_1 - 820sin\theta_1 cos(\theta_2 + \theta_3) - 270sin\theta_1 cos(\theta_2 + \theta_3 + \theta_5) \\ & + 850sin\theta_1 sin\theta_2 + 145sin\theta_1 sin(\theta_2 + \theta_3) \end{aligned}$$

$$\begin{aligned} BL_1 = & \ 350cos\theta_1 + 820cos\theta_1 cos(\theta_2 + \theta_3) + 270cos\theta_1 cos(\theta_2 + \theta_3 + \theta_5) \\ & - 850cos\theta_1 sin\theta_2 - 145cos\theta_1 sin(\theta_2 + \theta_3) \end{aligned}$$

For $i = 2$ we have:

$$\begin{bmatrix} J_{L_2} \end{bmatrix} = b_1 \times r_{1,E} = \begin{bmatrix} AL_2 \\ BL_2 \\ CL_2 \end{bmatrix} \tag{2.45}$$

where,

$$\begin{aligned} AL_2 = & -850cos\theta_1 cos\theta_2 - 145cos\theta_1 cos(\theta_2 + \theta_3) \\ & - 820cos\theta_1 sin(\theta_2 + \theta_)3 - 270cos\theta_1 sin(\theta_2 + \theta_3 + \theta_5) \end{aligned}$$

$$BL_2 = -850cos\theta_2 sins\theta_1 - 145cos(\theta_2+\theta_3)sin\theta_1 - 820sin\theta_1 sin(\theta_2+\theta_3) - 270sin\theta_1 sins(\theta_2+\theta_3+\theta_5)$$

$$CL_2 = 820cos(\theta_2+\theta_3) + 270cos(\theta_2+\theta_3+\theta_5) - 850sin\theta_2 - 145sin(\theta_2+\theta_3$$

For $i = 3$ we have:

$$\begin{bmatrix} J_{L_3} \end{bmatrix} = b_2 \times r_{2,E} = \begin{bmatrix} AL_3 \\ BL_3 \\ CL_3 \end{bmatrix} \tag{2.46}$$

where

$$AL_2 = -145cos\theta_1 cos(\theta_2+\theta_3) - 820cos\theta_1 sin(\theta_2+\theta_3) - 270cos\theta_1 sin(\theta_2+\theta_3+\theta_5)$$

$$BL_2 = -145cos(\theta_2+\theta_3)sin\theta_1 - 820sin\theta_1 sin(\theta_2+\theta_3) - 270sin\theta_1 sin(\theta_2+\theta_3+\theta_5))$$

$$CL_2 = 820cos(\theta_2+\theta_3) + 270cos(\theta_2+\theta_3+\theta_5 - 145sin\theta_2+\theta_3$$

For $i = 4$ we have:

$$\begin{bmatrix} J_{L_4} \end{bmatrix} = b_3 \times r_{3,E} = \begin{bmatrix} 270sin\theta_1 sin\theta_5 \\ -270cos\theta_1 sin\theta_5 \\ 0 \end{bmatrix} \tag{2.47}$$

For $i = 5$ we have:

$$\begin{bmatrix} J_{L_5} \end{bmatrix} = b_4 \times r_{4,E} = \begin{bmatrix} -270cos\theta_1 sin(\theta_2+\theta_3+\theta_5) \\ -270sin\theta_1 sin(\theta_2+\theta_3+\theta_5) \\ 270cos(\theta_2+\theta_3+\theta_5) \end{bmatrix} \tag{2.48}$$

For $i = 6$ we have:

$$\begin{bmatrix} J_{L_6} \end{bmatrix} = b_5 \times r_{5,E} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.49}$$

Now that we have calculated all the J_{L_i} and J_{A_i} matrices, from the eq. (2.38), eq. (2.39), eq. (2.40), eq. (2.41), eq. (2.42), eq. (2.43) and eq. (2.44),eq. (2.45),eq. (2.46),eq. (2.47),eq. (2.48) and eq. (2.49) we form our final form of our Jacobian from the equation eq. (2.14).

For the convenience of our simulation and our calculations, we consider the first joint as stable (meaning $q_1 = 0$ stable), and therefore we have the updated Jacobian matrix with $cos(\theta_1) = 1$ and $sin(\theta_1) = 0$.

2.4 Inverse kinematics

In robotics, the inverse kinematics calculation is the exact opposite of the forward kinematics calculation in terms of the end result, meaning we want to calculate the displacement of the joints (that is for $i = 1, 2, \cdots 6$) which result the end effector to be in a desired position and orientation. In other words, inverse kinematics determine the joint parameters that provide a desired position for our robot's end-effector with the use of kinematic equations. Specification of the movement of a robot so that our end-effector achieves the desired task is known as motion planning, with which we will also be dealing in this thesis. In order to compute the exact q_i's for all i's we need to solve the eq. (2.13), where J is the Jacobian that we calcualted. In our simulation, in order to simplify our calculations and our simulation

we suppose that the joint 4 and 6 are fixed and therefore do not affect in any way our end effector [28].

From overview of the eq. (2.9), if we compare it with:

$$A_E^0 = \begin{bmatrix} a_x & h_x & r_x & p_x \\ a_y & h_y & r_y & p_y \\ a_z & h_z & r_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.50}$$

We get that :

$r_x = cos\theta_1 cos(\theta_2 + \theta_3 + \theta_5)$

$r_z = sin(\theta_2 + \theta_3 + \theta_5)$

$p_x = 350cos\theta_1 - 850cos\theta_1 sin\theta_2 - 145cos\theta_1 sin(\theta_2 + \theta_3) + 820cos\theta_1 cos(\theta_2 + \theta_3)$
$+ 270cos\theta_1 cos(\theta_2 + \theta_3 + \theta_5)$

$p_z = 815 + 850cos\theta_2 + 145cos(\theta_2 + \theta_3) + 820(\theta_2 + \theta_3) + 270sin(\theta_2 + \theta_3 + \theta_5)$

Solving the eq. (2.13), with $q_1 = \dot{q_1} = q_4 = \dot{q_4} = q_6 = \dot{q_6} = 0$ we acquire our solution:

$$\begin{cases} q_2 = arcsin(\frac{K \cdot cos(\theta)}{\alpha}) - \alpha \\ q_3 = -arccos((\alpha - 850) \cdot \frac{cos(\phi)}{820}) + \phi - q_2 \\ q_5 = arccos(r_x) - q_2 - q_3 \end{cases} \tag{2.51}$$

where:

$$K = \frac{\alpha^2 + \beta^2 + 850^2 - G^2 - 145^2}{1700}$$
$$\alpha = p_x - 270r_x - 350$$

$$\beta = p_z - 270r_z - 815$$

$$\theta = arctan(\frac{\beta}{\alpha})$$

$$\phi = arctan(\frac{145}{220})$$

and the values of r_x, r_z, p_x, p_z are the ones we obtained by spectating the A_E^0 matrix.

2.5 Inverse differential model

The equation eq. (2.12) gives us the linear speed and the angular velocity of our robot's end effector as a linear function of the joint velocities. In practice, we have to compute the velocities $\dot{q}_i$'s of the joints which lead the end effector to our desired linear and angular velocity. Therefore we have to solve the eq. (2.12) towards $\cdot q$. Our robot is a six degrees of freedom $q_1, q_2, q_3, q_4, q_5, q_6$ robot, with a 6×6 Jacobian matrix.If our Jacobian matrix is nonsingular then the solution of eq. (2.12) will be:

$$\dot{q} = J^{-1} \cdot \dot{p} \tag{2.52}$$

The above equation gives us the required speeds of the joints in order to acquire a specific linear or angular velocity $\dot{p}$ on our end effector [28]. Of course here we must mention that there might be forms on which J might not be invertible. Therefore J^{-1} does not exist and so no solution exists. That is what we call a special configuration of our robotic arm. In this special configuration the columns of our J matrix are linearly depended and there exists at least one direction on which the robot can not move, independent of our

way of choosing our joint velocities $\dot{q_1}, \dot{q_2}, \dot{q_3}, \dot{q_4}, \dot{q_5}, \dot{q_6}$.

Concluding in order to compute the inverse differential model we need to inverse our Jacobian matrix. In our case of movement simulation though we turn it for q_1. Therefore with $q_1 = q_4 = q_6 = \dot{q_1} = \dot{q_4} = \dot{q_6} = 0$ results in some all-zeros rows which do not allow the inverse of the matrix. Therefore, we erase them and conclude to the reduced Jacobian matrix, which is presented below:

$$J(q_2, q_3, q_5) = \begin{bmatrix} J_{11} & J_{12} & -270sin(\theta_2 + \theta_3 + \theta_5) \\ J_{21} & J_{22} & 270cos(\theta_2 + \theta_3 + \theta_5) \\ -1 & -1 & -1 \end{bmatrix} \tag{2.53}$$

where:

$J_{11} = -850cos(\theta_2) - 145cos(\theta_2 + \theta_3) - 820sin(\theta_2 + \theta_3) - 270sin(\theta_2 + \theta_3 + \theta_5)$

$J_{12} = -145cos(\theta_2 + \theta_3) - 820(\theta_2 + \theta_3) - 270(\theta_2 + \theta_3 + \theta_5)$

$J_{21} = 820cos(\theta_2 + \theta_3) + 270cos(\theta_2 + \theta_3 + \theta_5) - 850sin(\theta_2) - 145sin(\theta_2 + \theta_3)$

$J_{22} = 820cos(\theta_2 + \theta_3) + 270cos(\theta_2 + \theta_3 + \theta_5) - 145sin(\theta_2 + \theta_3)$

With the use of Mathematica we compute the pseudo-inverse Jacobian matrix J^{-1}:

$$J^{-1}(q_2, q_3, q_5) = \begin{bmatrix} J_{11}^{-1} & J_{12}^{-1} & J_{13}^{-1} \\ J_{12}^{-1} & J_{22}^{-1} & J_{23}^{-1} \\ J_{31}^{-1} & J_{32}^{-1} & J_{33}^{-1} \end{bmatrix} \tag{2.54}$$

with,

$$J_{11}^{-1} = \frac{-820cos(\theta_2+\theta_3)+145sin(\theta_2+\theta_3)}{det(J)}$$
$$J_{12}^{-1} = \frac{145cos(\theta_2+\theta_3)-820sin(\theta_2+\theta_3)}{det(J)}$$
$$J_{13}^{-1} = \frac{-39150cos(\theta_5)+221400sin(\theta_5)}{det(J)}$$
$$J_{21}^{-1} = \frac{820cos(\theta_2+\theta_3)-850sin(\theta_2)-145sin(\theta_2+\theta_3)}{det(J)}$$
$$J_{22}^{-1} = \frac{850cos(\theta_2)+145cos(\theta_2+\theta_3)+820sin(\theta_2+\theta_3}{det(J)}$$
$$J_{23}^{-1} = \frac{229500cos(\theta_3+\theta_5)+39150cos(\theta_5)-221400sin(\theta_5)}{det(J)}$$
$$J_{31}^{-1} = \frac{850sin(\theta_2)}{det(J)}$$
$$J_{32}^{-1} = \frac{-850cos(\theta_2)}{det(J)}$$
$$J_{33}^{-1} = \frac{-697000cos(\theta_3)-229500cos(\theta_3+\theta_5)+123250sin(\theta_3)}{det(J)}$$

Also it is obvious based on what we said above that we will need to find the determinant of our Jacobian. That is because in order to find the special configurations of our system that the robot has, we need to solve the equation $Det(J) = 0$. Solving the equation we get:

$$Det(J) = 697000cos(\theta_3) - 123250sin(\theta_3) \tag{2.55}$$

Therefore we conclude that we have a singularity for:

$$q_3 = arctan^{-1}\frac{697000}{123250} = 79.97^\circ \tag{2.56}$$

2.6 Kinematic Simulation

The kinematic simulation of our model was done in Matlab and our results are given below. For our simulation we considered that the angular deviations q_4 and q_6 (two of the "wrist" joints) are fixed in their zero configuration (ie, $q_4 = q_6 = 0 =$ station). Also the points p_1, p_2 and p_3 belong to a vertical plane defined by an angle $\theta_z = 30^\circ$ in relevance to the zero base position of the robot.

We assume that at time $t = 0$ the robot is already in the initial position and that the desired (segmentally linear) trajectory of the final action element must last altogether within 10 seconds. The duration of our robot simulation lasts 4 seconds per linear segment of the track, and 2 secondss for the intermediate shift phase of the orientation of our robot. The desired position of the robot end effector $(p_{E_x}, p_{E_y}, p_{E_z})$ at every moment follows at Figure 2.4, Figure 2.5, and Figure 2.6.

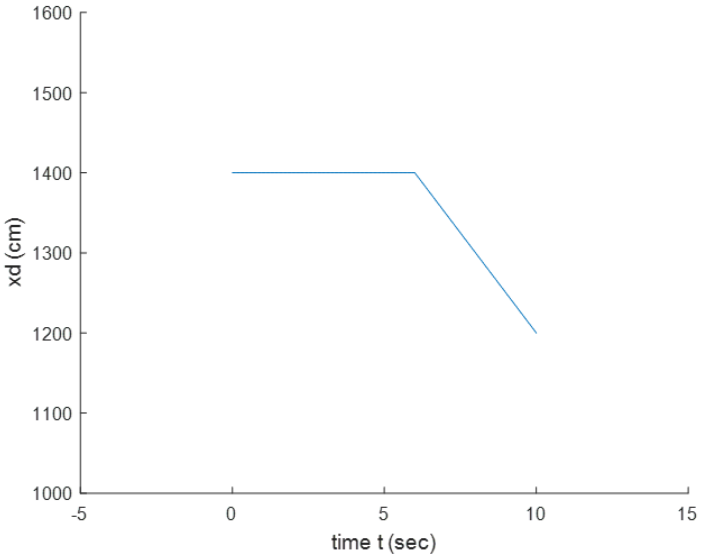


Figure 2.4: Desired end effector x-position during our motion

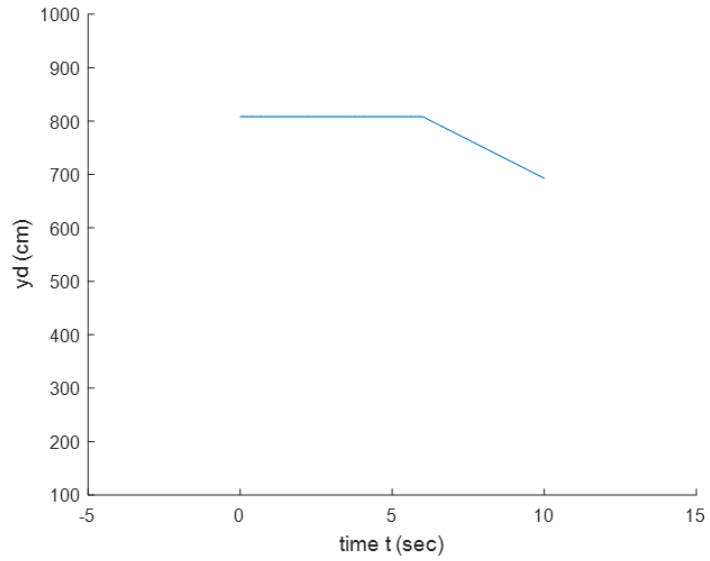


Figure 2.5: Desired end effector y-position during our motion

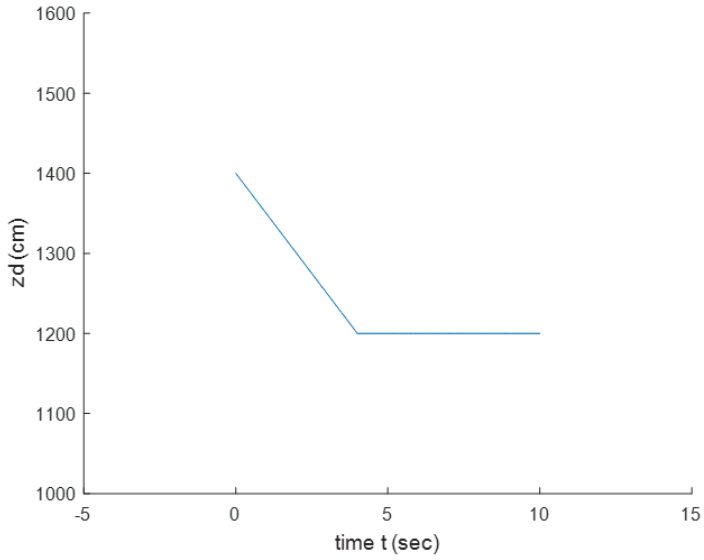


Figure 2.6: Desired end effector z-position during our motion

At Figure 2.7 follows the the orientation angle at our work level.

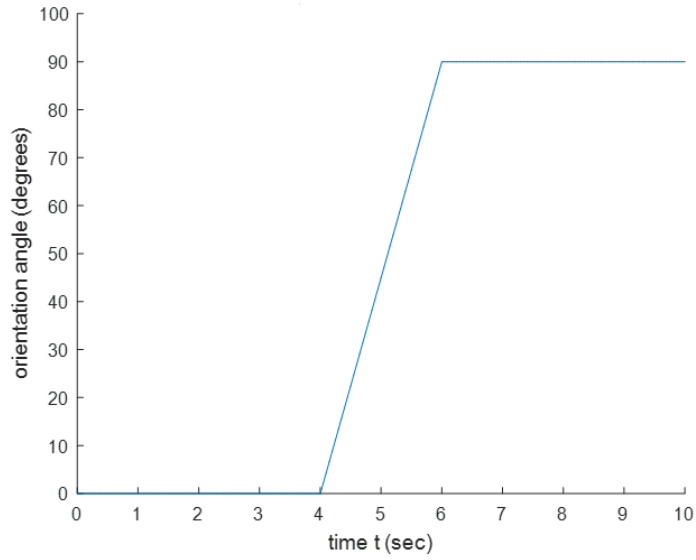


Figure 2.7: The orientation angle of our robot's end effector during our motion

At Figure 2.8 we present the linear and angular velocity of the action tool of our robot.

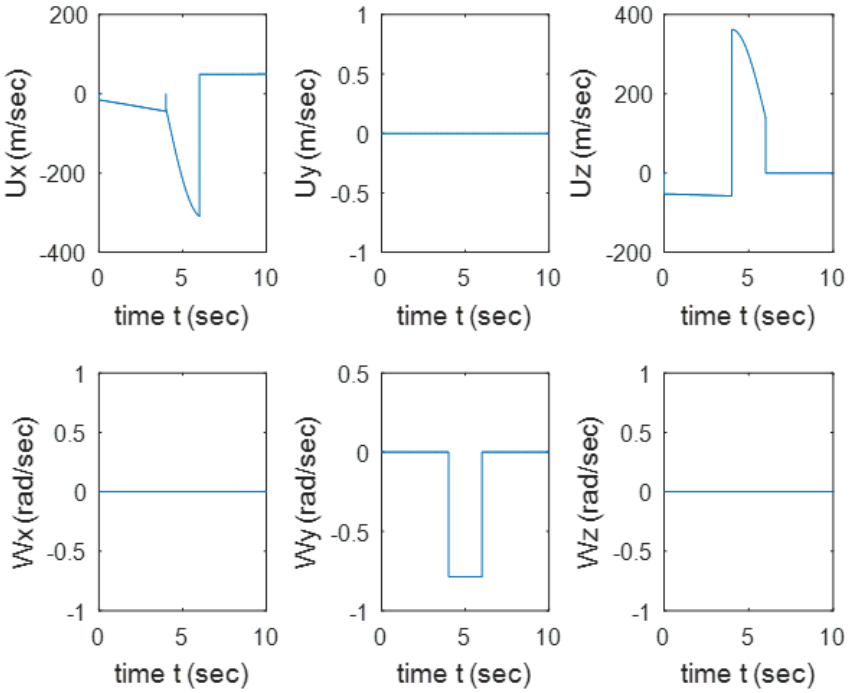


Figure 2.8: Linear and angular velocities of the end effector during our motion time

At Figure 2.9 are presented the joint angles at each time point t.

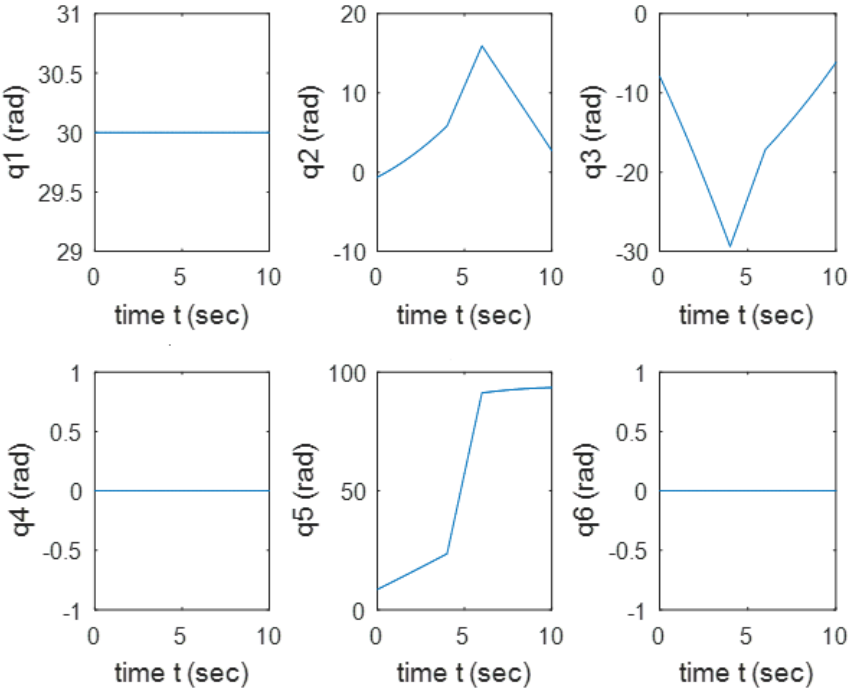


Figure 2.9: Our robotic joints angles at each time t

At Figure 2.10 are the angular velocities of all of our robot's joints at each moment t.

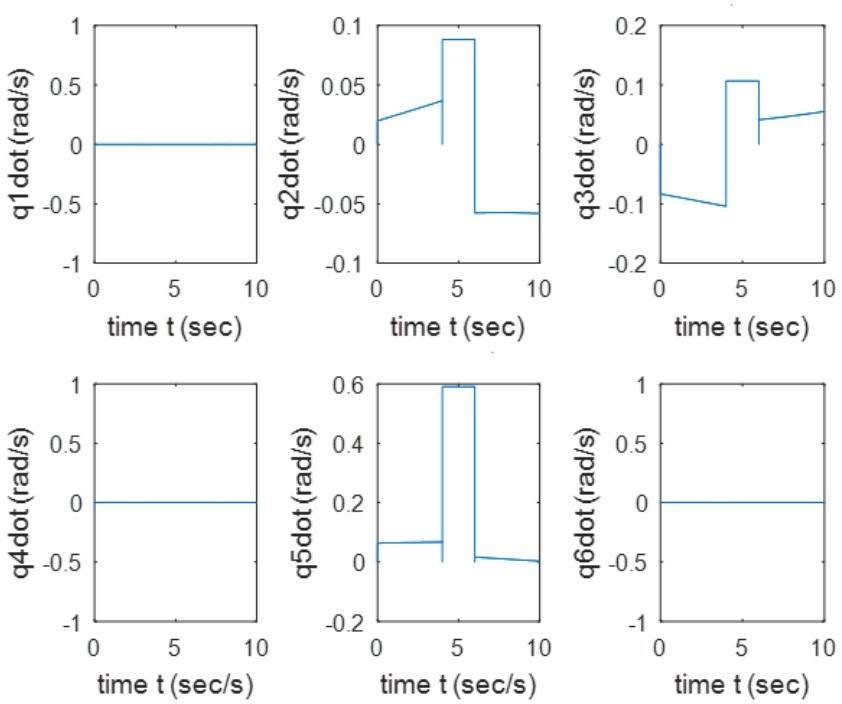
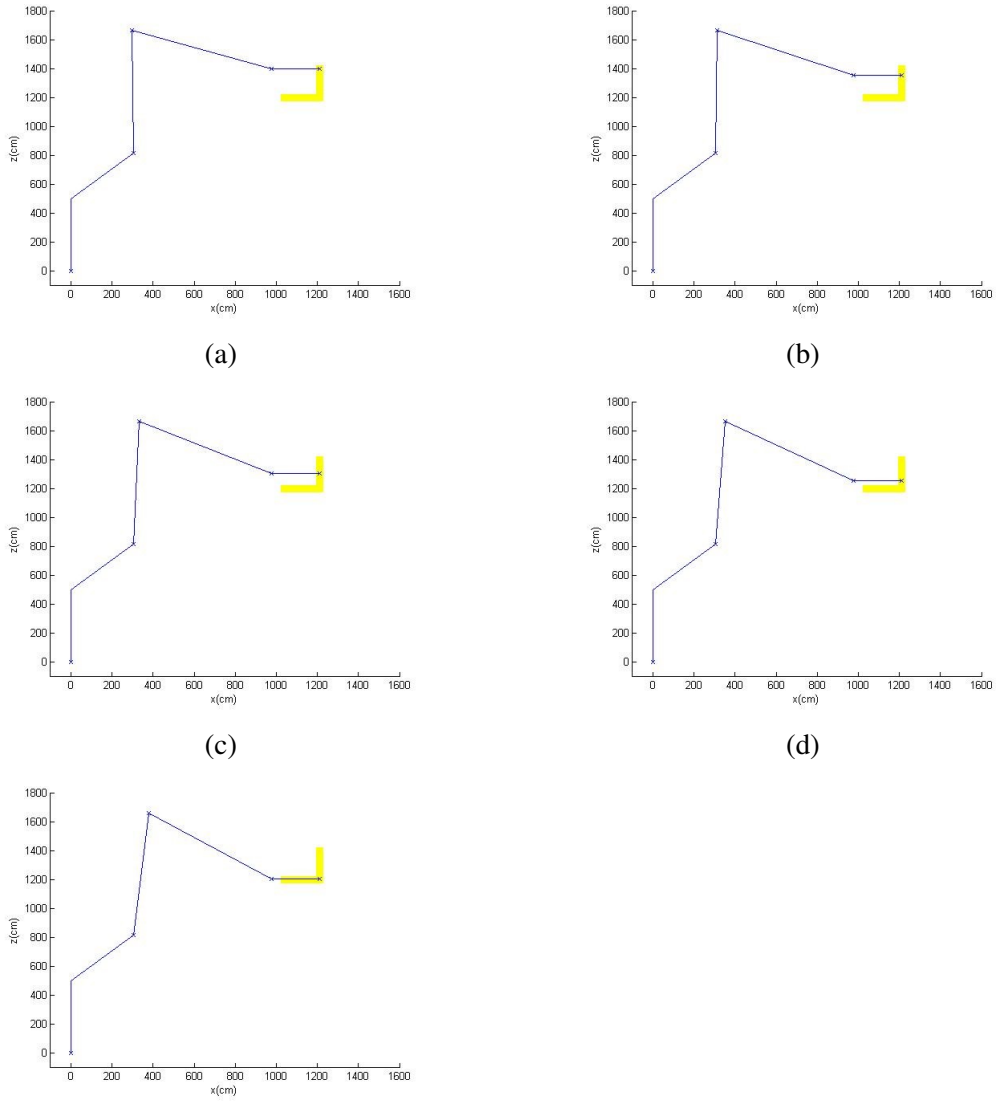


Figure 2.10: Our robotic joints angular velocities during the simulation movement

Below we present a step by step simulation of the motion of our robot. We start by presenting the first four seconds of the motion in Figure 2.11.



(a) (b)

(c) (d)

(e)

Figure 2.11: The first phase(linear) of our robot's KR 360 FORTEC movement simulation

We continue by presenting the next two seconds of the motion in Figure 2.12.

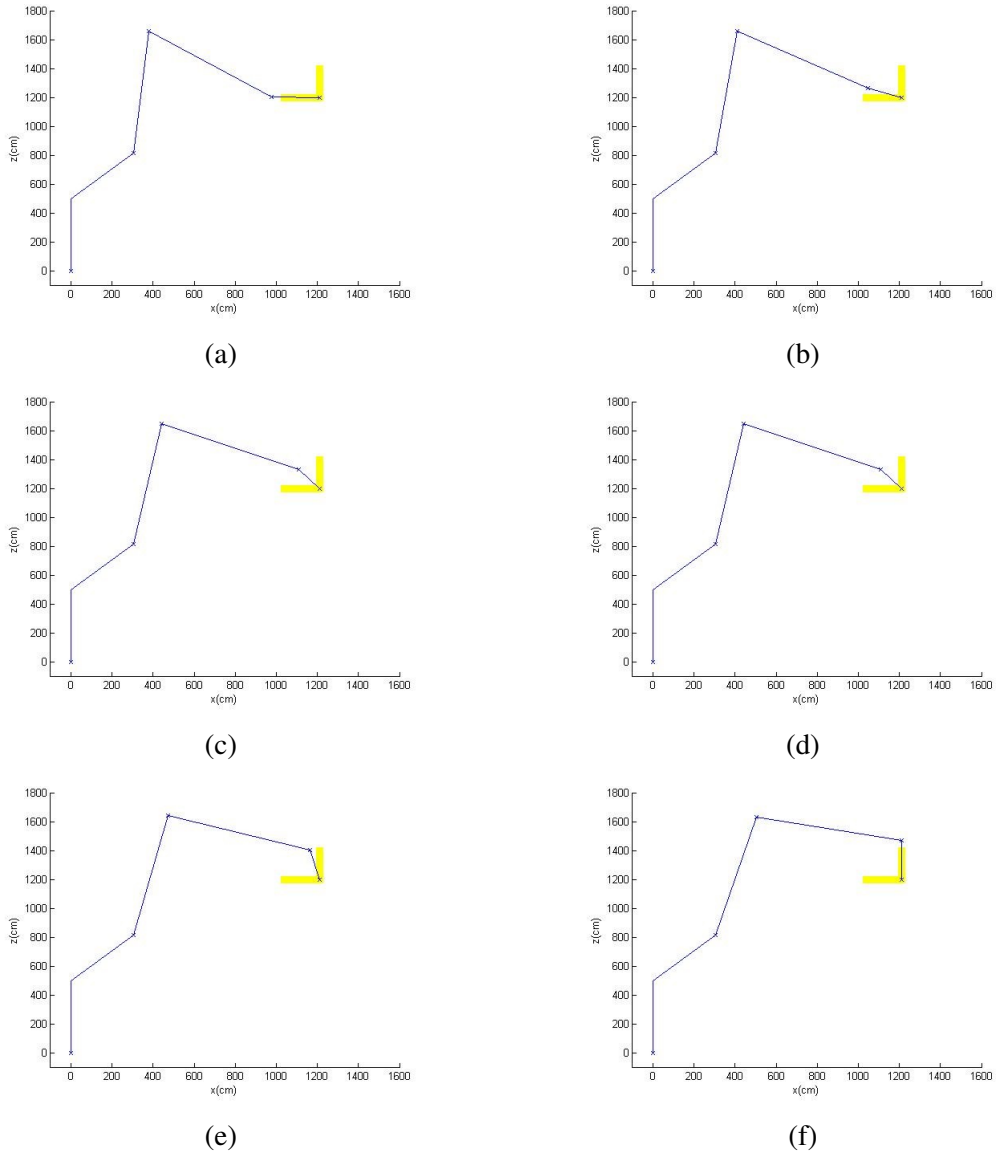
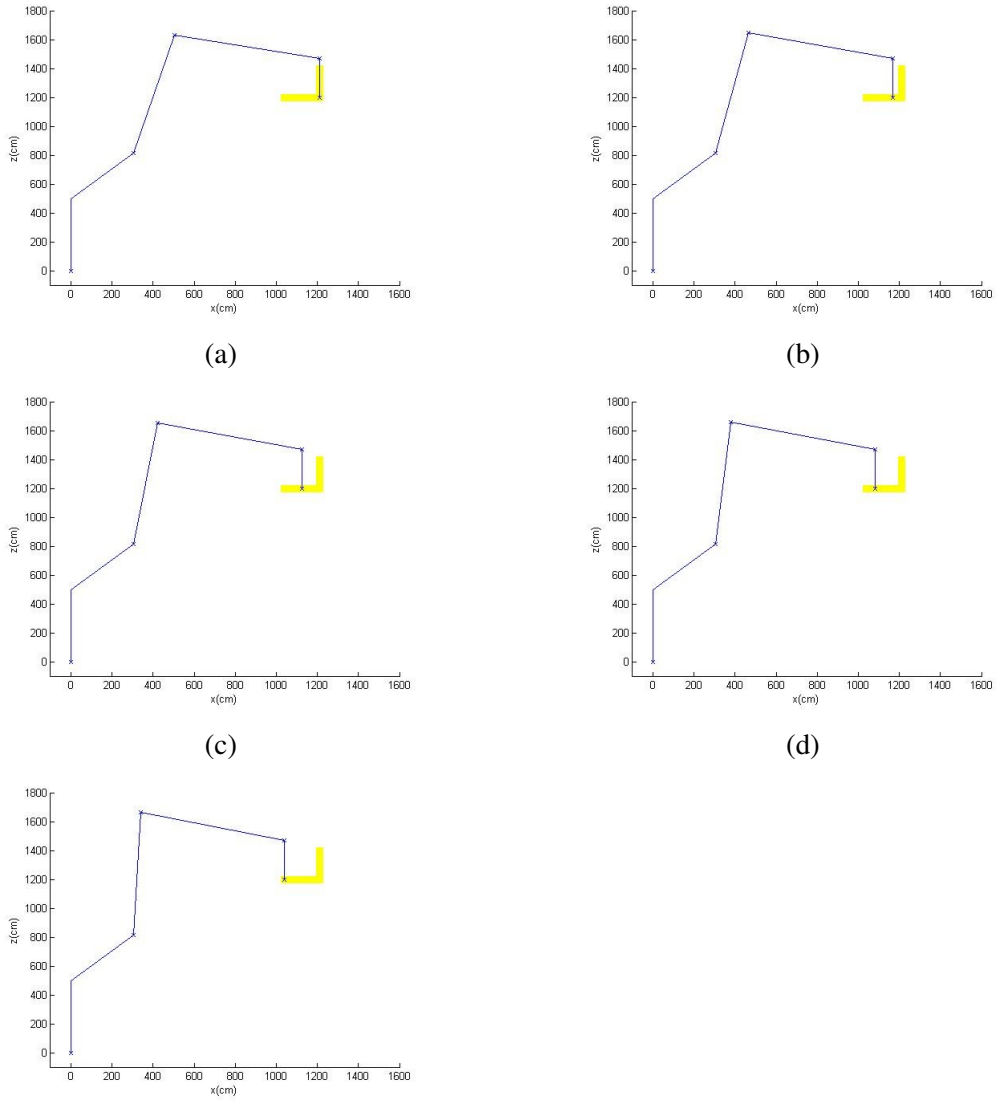


Figure 2.12: The second phase of our robot's KR 360 FORTEC movement simulation

Finally at Figure 2.13 are the last four seconds of the linear movement of our robot's end effector.



(a) (b)

(c) (d)

(e)

Figure 2.13: The last phase(linear) of our robot's KR 360 FORTEC movement simulation

The complete robot motion simulation follows at Figure 2.14 :

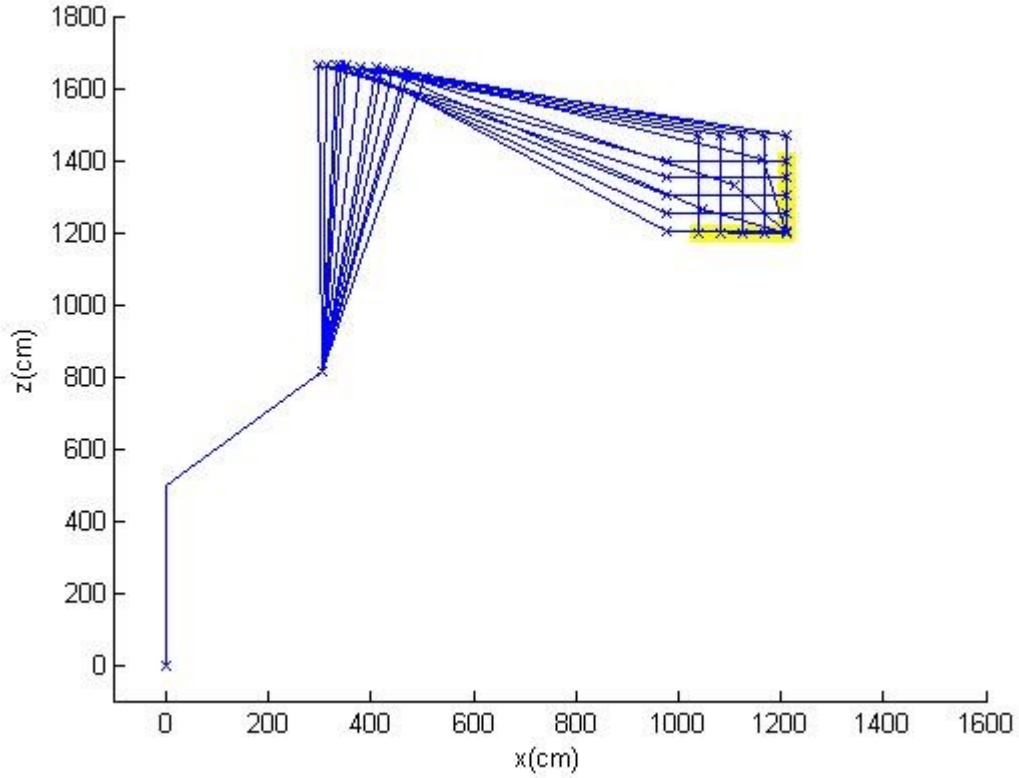


Figure 2.14: The complete robot motion simulation

Chapter 3: Baxter robot dynamics and introduction to planning algorithms

3.1 Motivation

Having seen and fully analyzed the kinematics of a 6-degree of freedom robotic arm we move on and work with the Baxter robot, starting from analyzing its dynamics. Our approach to the dynamics of the Baxter robot will be done through the Euler-Lagrange equation, as we will see later on. A significant mention at this point is that in order for a robot to be able to place its end effector in any arbitrary position with any kind of orientation, then it must have at least six degrees of freedom. Also six degrees of freedom are needed in order for the end effector to move on any orientation with any desired angular velocity. That is also the reason why most of the industrial robots have six degrees of freedom, just like the one we studied before. The Baxter robot has seven degrees of freedom, adding more freedom to our movement capability.

3.2 General Robot Dynamics

As we saw before we fully analyzed a six degrees of freedom robot and simulated its motion. A good way to understand Baxter robot dynamics, is to analyze an accurate

dynamic model. The accurate dynamic model of a robot manipulator can be useful for the design of motion control systems, for the simulation of the manipulators motion or even for the analysis of our mechanical design. An accurate model [23] of the manipulator dynamics is commonly of the Lagrangian form:

$$B(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = u \tag{3.1}$$

where,

- q denotes the vector of joint angles.
- $B(q) \in \Re^{n\times n}$ is the symmetric, bounded, positive definite inertia matrix.
- n is the degree of freedom (DoF) of the robot arm.
- $C(q)\dot{q} \in \Re^n$ denotes the Coriolis and Centrifugal force.
- $g(q) \in \Re^n$ is the gravitational force.
- $u \in \Re^n$ is the vector of actuator torques.

At the above form the kinetic energy of the manipulator is described within $B(q)\ddot{q} + C(q,\dot{q})$, and the gravity term $g(q)$ stands for the potential energy. This can then be used for the computation of the forward dynamics, where the manipulator motion is computed based on a vector of applied torques. It can also be used to calculate the inverse dynamics (useful in control designs), where the torques for a given set of joint parameters can be found. There are many control algorithms that normally require an accurate model of the manipulator dynamics as the one we described in eq. (3.1). It is also important to

mention that there exist two different methods that one can use to formulate the dynamics in eq. (3.1). The first is the Euler-Lagrange formulation and the other the Newton-Euler. Both of those methods are based on the specific and inertial parameters of the robot and both equivalently describe the dynamic behaviour of the robotic motion. Below we present some of the basic symbolism we used before and we will use afterwards [29], [24].

Nomenclature	
n	Degrees of freeedom
$q, \dot{q}, \ddot{q} \in \Re^{n\times 1}$	Vector of joint position, angular velocity and acceleration respectively
a, d, α, θ	Variables denoting the Denavit-Hartenberg parameters
$I_i \in \Re^{3\times 3}$	Inertia tensor of link i
V	Volume occupied by the body
$p(r)$	Mass density
$m = \int_V p(r)dV$	Is the link mass
$\bar{r}_i \in \Re^{4\times 1}$	Centre of mass of link i
$A^i_j \in \Re^{4\times 4}$	Homogeneous transform from link i to j

The Euler-Lagrange method is an easy method for computing the exact kinetic and the potential energies of the rigid body system. The Baxter robot is made up of two seven degrees of freedom arm manipulators in a semi-manual configuration, attached to a central pedestal. The configuration of the Baxter's manipulator is a complete specification of every point on the manipulator. The design aim of Baxter was to create a safe, flexible and affordable robot for integration into low-volume production. The Software Development

Kit(SDK) of the Baxter is released and free, and therefore opening Baxter up to research opportunities.

3.3 Dynamics of the Baxter

3.3.1 Baxter's kinematic equation

As we did in the previous section with the KR 360 FORTEC, we start our dynamics analysis from the Denavit-Hartenberg matrix. The Denavit-Hartenberg parameters and link masses of the Baxter manipulator are presented in 3.1 and are derived from the Universal Robot Descriptor File (URDF) for the Baxter. As we said the Denavit Hartenberg parameters describe the configuration of the links, and form the basis of the Lagrange-Euler formulation. The homogeneous link transform matrices are formed from the Denavit Hartenberg matrix also.

D-H Parameters of the Baxter Robot					
Link	θ	d(m)	a(m)	a(rad)	m(kg)
1	θ_1	0.2703	0.069	$-\frac{\pi}{2}$	5.700443
2	θ_2	0	0	$\frac{\pi}{2}$	3.22698
3	θ_3	0.3644	0.069	$-\frac{\pi}{2}$	4.31272
4	θ_4	0	0	$\frac{\pi}{2}$	2.07206
5	θ_5	0.3743	0.01	$-\frac{\pi}{2}$	2.24665
6	θ_6	0	0	$\frac{\pi}{2}$	1.60979
7	θ_7	0.2295	0	0	0.54218

Table 3.1: Denavit Hartenberg matrix of our Baxter robot

We must mention that the center of mass as well as the inertia tensors I_{xx}, I_{yy}, I_{zz}, I_{xy}, I_{xz}, I_{yz} of each joint of the Baxter robot are also known to us. The homogeneous link transform matrices are formed from the Denavit Hartenberg parameters accordingly [29], [24]. The Baxter's robot kinematic equation will be formed upon:

$$A_i^{i-1} = \begin{bmatrix} cos\theta_i & -cos\alpha_i sin\theta_i & sin\alpha_i sin\theta_i & \alpha_i cos\theta_i \\ sin\theta_i & -cos\alpha_i sin\theta_i & -sin\alpha_i cos\theta_i & \alpha_i sin\theta_i \\ 0 & sin\alpha_i & cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.3.2 Euler-Lagrange analysis

The Euler-Lagrange equations of motion for a conservative system are given by:

$$L(q,\dot{q}) = K(q,\dot{q}) - U(q) \quad \text{and}$$
$$u = \frac{d}{dt}(\frac{\partial L}{\partial \dot{q}}) - \frac{\partial L}{\partial q}$$

where $K(q,\dot{q})$ and $U(q)$ are the total kinetic and potential energies of our system respectively, $q \in \Re^n$ are the generalized robot coordinates equivalent to θ in the Denavit Hartenberg matrix, and u is the torque at the robot joints [29], [24]. The kinematic energy of our system will be :

$$K = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=1}^{i}[Tr(U_{ij}J_iP_{ik}^T)\dot{q}_j\dot{q}_k]$$

where J_i is the Jacobian of our system-robot and P_{ij} is the rate of change of the positions of the points on link i while the joint position is changing compared to our robot's base. Therefore P_{ij} will be:

$$P_{ij} = \frac{\partial A_i^0}{\partial q_j}$$

The potential energy [29], [24] is going to be computed from:

$$U = \sum_{i=1}^{n} -m_i\mathbf{g}(T_i^0\bar{r}_i)$$

By substitution of $K(q,\dot{q})$ and $U(q)$ to the Euler-Lagrange equation we obtain both L and u. Of course in order to compute the kinetic energy, one has to evaluate the Jacobian of the Baxter. Having the Baxter's link mas, the inertia tensors and the center of mass known,

due to the fact that the Jacobian is independent of the links positions or motions, it can be computed directly from:

$$J_i = \begin{bmatrix} \frac{-I_{xx_i}+I_{yy_i}+I_{zz_i}}{2} & I_{xy_i} & I_{xz_i} & m_i\bar{x}_i \\ I_{xy_i} & \frac{I_{xx_i}-I_{yy_i}+I_{zz_i}}{2} & I_{yz_i} & m_i\bar{y}_i \\ I_{xz_i} & I_{yz_i} & \frac{I_{xx_i}+I_{yy_i}-I_{zz_i}}{2} & m_i\bar{z}_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Having computed $K(q,\dot{q})$ and $U(q)$ and the Jacobian J_i we have concluded all we needed to form the Euler Lagrange dynamics of the Baxter's arm.

3.4 Robot planning algorithms

3.4.1 Deliberative Acting

In this chapter we are moving on and implementing in Python two robot planning algorithms, the RRT [9] and the RRT* [21],[22]. Planning is motivated by acting. Therefore, before someone goes into planning should become familiar with the word acting or actor. Planning is well formalized from an Artificial Intelligence point of view whereas acting is something harder to formalize.

Action [11] is something that the actor (can be the robot) does in order to achieve a change in his environment, that is make a motion, or exert a force or communicate. The actor is the person who performs an action. An actor can either be versatile or autonomous. In both cases deliberation is needed. Deliberation for acting deals with what kind of actions are needed to be performed in order an objective to be achieved and also how those actions

are going to be performed. Deliberative acting can either rely on a model-based behaviour, on a learned behaviour or on a innate behavior, which can either be programmed beforehand or evolved.

3.4.2 Planning

Planning as in [11] is a mapping from abstract domains and problems into plans. This means to synthesize an organized set of actions to achieve some purpose or to achieve a desired goal by organizing the required activities. Planning relies on prediction and search. Descriptive models of actions to predict their effects and search over predicted states and possible organizations of feasible actions. A complex task is considered to be a task that consists of a number of sub-tasks which are further decomposable into another set of sub-tasks. Those sub-tasks need different planner types, in order to be accomplished. This is the so called task planner. In order to achieve a goal situation it decomposes a high-level task into another set of possibly parallel sub-tasks. High-level planners deal with how task planners work in a general aspect and how they are used for solving manipulation problems. And as we said above those sub-tasks might often need different low-level planners to be solved.

3.4.3 Planner Integration

As mentioned, assume that a plan is generated and then mapped onto low-level planners with the use of a hierarchical top-down approach. Now we assume that the world model is

correct, which means that all the selected actions are correct and that our initial symbolic abstraction is also correct. Afterwards we use a hierarchical bottom up approach. That means that all our low-level planners compute our task planner using all the relevant geometric information of the scene and then try to combine our low level solution planners for our initial task planner.

3.4.4 Planner types

There are three planner types in general [11]. In our thesis we will be dealing only with path planners. There exist also grasp planners and robot placement planners. The most important things one should be aware of the two other planners are that the grasp planners provide grasps for handling objects and that robot placement planners position the robot for a task. Each of the domains of path planning, grasp planning and task planning use separate benchmarks.

Grasp planners [11] either compute grasps for a robotic hand or given a center point coordinate (TCP) they return a valid reachable grasp. The main objective of a planned grasp is to hold an object firmly and safely, also in the presence of disturbances on the object. If the grasp has the force closure property, then a set of valid contract forces exists and allows a grasp to balance any occurring disturbance forces or torques. Meaning it includes control on the object.

Robot placement planners [11] determine the position for a manipulator to execute the task. Few planners are used for placing the robot for grasping tasks. Most approaches combine the search for feasible trajectories for the robot arm with positioning of the robot

in the configuration space(C-space).

Our main interest focuses on path planners. Path-planning requires a map of the environment and the robot to be aware of its location with respect to the map. Paths are a sequence of points in the configuration space of the robotic arm. Frequently, the points are connected via straight line segments. There exist two classes of path planners and those are the regional path planners and the global path planners.

Regional path planners [11] locally distort the direct path connecting the given start and goal configuration of the robot's arm. Local distortions are used to avoid obstacles. A path is found by avoiding the revolting forces and moving towards the attractive ones. Regional path planners perform well in scenarios where only few objects exist and many times give smoother paths than those generated from the global planners.

Global path planners search the whole configuration space of the robot for valid paths. In global motion planning, target space, meaning the linear subspace of free space which denotes where we want our robot to move to, is observable by the robot's sensors. A robot configuration is randomly sampled from the C-space and it is accepted if it is collision-free. Frequently, it is connected to other valid configurations with the use of collision-free straight line segments in the C-space. The most useful and well known global planners are the probabilistic ones. The most common probabilistically complete path planners are the RRT [9] and the PRM [10]. In this section we will be dealing and presenting how our robot can be planned with the use of RRT and RRT* which is a variant of RRT. Later on, we will be experimenting with both of them on Baxter with Moveit! [25] applying planning experiments, including obstacle avoidance.

3.4.5 Configuration Space

A key concept for motion planning is configuration. The configuration space (C-space) is the space of all possible configurations. C-space topology is usually non Cartesian and is described as a topological manifold. The main idea behind the configuration space comes from the convenience of our calculations. Due to the fact that most of the robots are rigid bodies and therefore not single points the planning, the obstacle avoidance and the exact specification of the position of every point of our system is really hard to determine. Therefore instead of working in "the real world" with our robot having real dimensions, we consider our robot as a single point and we expand our obstacles dimensions in a uniform way. (Most of the times we expand our obstacles by a constant same or slightly bigger than the maximum size of our robot). The free configuration space is obtained by sliding the robot along the edge of the obstacle regions "enlarging them" by the robot radius.
This operation is called the Minkowski sum :

$$\mathrm{A}\oplus B = \{a+b \,|\, a \in A, b \in B\}$$

where A,$B \subset \Re^n$.

The configuration space can be randomly sampled with a uniform distribution. The notation that we used and it is generally used for the configuration space is :

- Configuration space $C \subset \Re^n$
- Configuration $q \in C$
- Free configuration space C_{free}

- Obstacle space C_{obs},

3.4.6 Domain Model

A way to model a complex environment is via simplifying assumptions and with planning. Most of the planning systems are being formulated on the model of State-transition system. State-transition system (or classical planning domain) is denoted by:

- System denoted as : $\Sigma = (S, A, \gamma)$ or $\Sigma = (S, A, \gamma, cost)$
- States $S = \{s_1, s_2, ...\}$ describe the current state. S is a finite set of states that the system may be in.
- $A = \{a_1, a_2, ...\}$ are actions through which the current transition to new states is triggered. A is a finite set of actions. That means, things the actor can do.
- $\gamma : S \times A \rightarrow S$ is the prediction function (or state-transition function) partial function: $\gamma(s, a)$ isn't defined unless a is applicable in s.
- $Dom(a) = \{s \in S | \, \gamma(s, a) \text{ is defined}\} = \{\text{s} \in S | \, a \text{ is applicable}\}$ .Range(a) = $\{ \gamma(s, a) | \, s \in Dom(a)\}$
- $cost$: $S \rightarrow A \rightarrow R$ could be monetary cost, time required, something else

In general we have the planning problem $P = (\Sigma, s_0, S_g)$ (planning domain, initial state, set of goal states). The solution for P, will be a plan (sequence of actions) that will produce a state in S_g

If S and A are small enough then we give each state and action a name. For each s and a we then store $\gamma(s,a)$ in a look-up table.

In larger domains we do not represent all states explicitly. There exist a Language for describing properties of states and a Language for describing how each action changes those properties. We start with an initial state and use actions to produce other states.

3.5 Planning dilemma

In the planning "world" there exists a big dilemma, known as the exploration versus exploitation. Exploration is the need of our actor, robot, our planner or our learning planning algorithm to gather more information, whereas exploitation is the need to make the best decision given any current information we might have. It is an online decision-making that involves a fundamental choice between those two. On one hand the best long-term goal strategy may involve short-term sacrifices, but on the other hand gathering more information might be more useful to make the best overall decisions. We solve this dilemma with the right selection of the discount factor in our planners. The selection of this value is often empiric and based upon the experimental results and of course based on the nature of our planning problem and world.

3.6 Robot using RRT and RRT* in a realistic simulation environment

Our main interest focuses on path planners. In our simulation we build the RRT as our robot's planning algorithm in an attempt to experiment ourselves with the way RRT works. This will help us understand later on how RRT works on Baxter.

3.6.1 RRT planner

In searching for a path RRT planner iteratively grows a tree in the configuration space. The search is successful when the goal configuration q_{goal} is reached. RRTs can naturally consider constraints during the tree construction. The RRT algorithm follows :

1.) Initialize tree with first node q_i

2.) Pick a random target location (every k_{th} iteration, choose q_g), because else it will fail and run into C_{obs} and it will not explore, if it did not chose a random target.

3.) Find closest vertex in roadmap

4.) Extend this vertex towards target location

5.) Repeat steps until goal is reached

The formal RRT algorithm follows 1:

A few things one should know about the Rapidly Exploring Random Trees is that they are probabilistic complete as we mentioned before and they have a good balance between exploration and greedy search. RRT is a very popular algorithm and has many extensions, like the RRT* which we will be simulating as well.

Starting from a world without obstacles, our algorithm awaits for the user to enter the initial position of the robot in the world. Afterwards, we also give as an input the goal position to our planner. Our robot is assumed to be a single point being planned in the C-space as the obstacles have been ”enlarged” from their real dimensions. In the following images we present our robot using RRT and starting from an initial position trying to achieve the desired end position. One can also see how our robot explores its

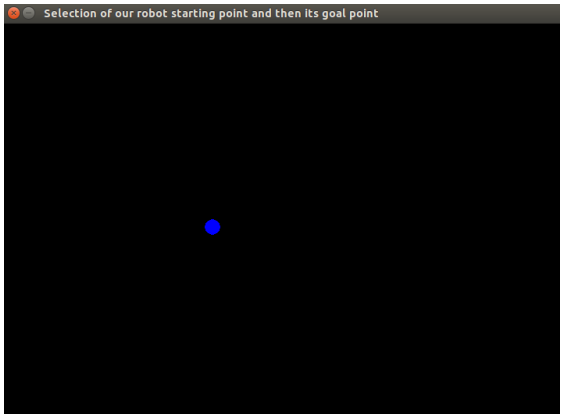
Algorithm 1: RRT

Data: Initial configuration q_{init}, number of vertices in RRT K, incremental distance Δ_x

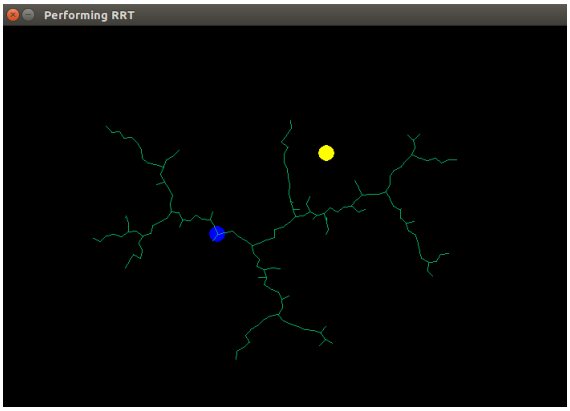
Result: Graph G

```
G.init(q_init);
for k = 1 : K do
    q_rand ← Randconf();
    q_near ← Nearestvertex(q_rand, G);
    q_new ← Newconf(q_near, q_rand, Δ_x);
    G.addvertex(q_new);
    G.addedge(q_near, qnew);
return G
```

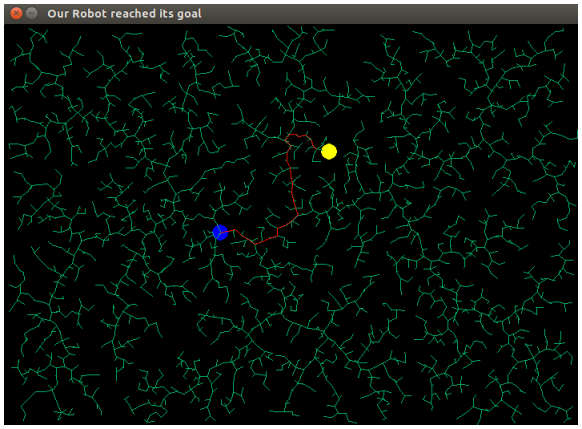
environment in its attempt to reach its goal.We started by experimenting in a obstacle-free world and then moved on and added some obstacles to see how our robot is going to perform. Our simulations where done in Python.



(a) Initialization



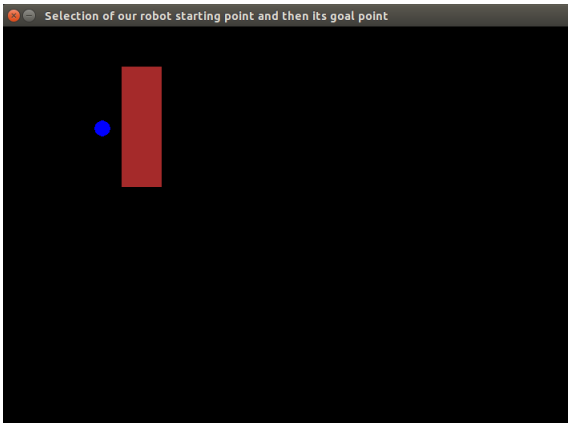
(b) Goal position and exploration phase



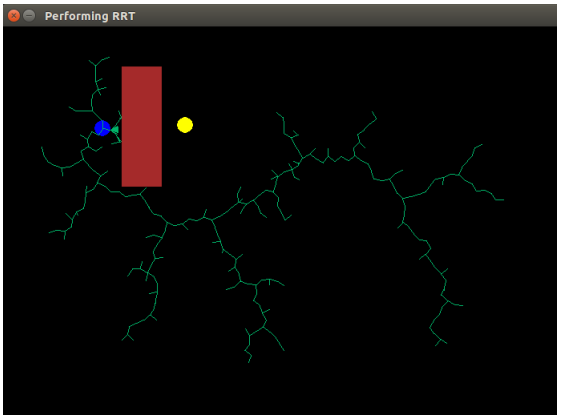
(c) Feasible path found

Figure 3.1: Our robot using RRT in a obstacle-free world

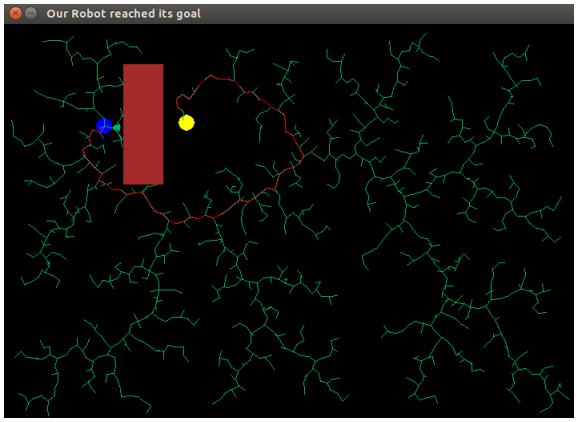
Having figured out the basics around RRT we added a rectangle as an obstacle to our 2D robot's world.



(a) Initialization



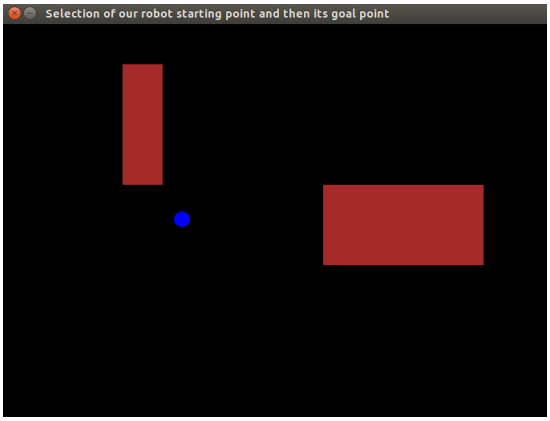
(b) Goal position and exploration phase



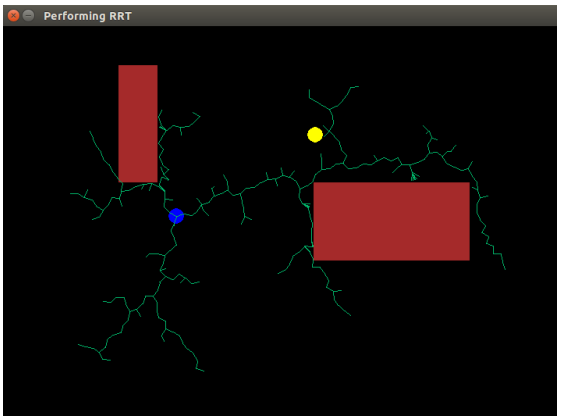
(c) Feasible path found

Figure 3.2: Our robot using RRT in a world with one obstacle

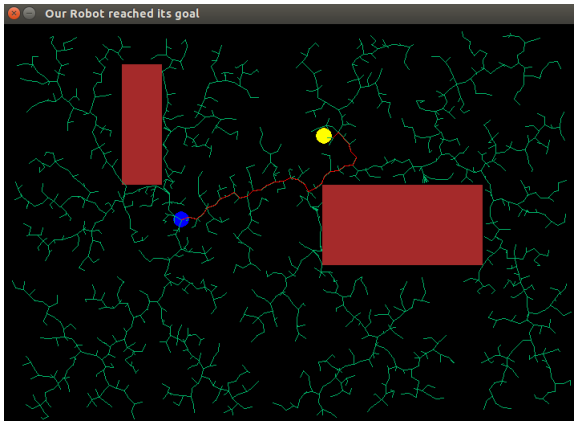
Finally in a attempt to make it even more challenging we added one more obstacle.



(a) Initialization



(b) Goal position and exploration phase



(c) Feasible path found

Figure 3.3: Our robot using RRT in a world with two obstacles

3.6.2 RRT* planner

Same with RRT* we simulated an environment with obstacles in which our robot from and initial position q_{init} explores its search in the workspace until it finally reaches our q_{goal}. We give two points q_{init} and q_{goal} we use a path defined by $p(\tau) = \tau \cdot q + (s - \tau) \cdot q'$, $\forall \tau \in [0, s]$, where $s = \| q' - q \|$.The pseudocode for RRT* [21],[22] as in algorithm 2:

Algorithm 2: RRT*

Data: Given a tree G = (V, E) and a vertex v V

Result: Graph G

$V' \leftarrow E$;

$E' \leftarrow V$;

$q_{near} \leftarrow Nearest(G, x)$;

$q_{new} \leftarrow Steer(q_{near}, x)$;

if $ObstacleFree(q_{near}, q_{new})$ **then**

    $V' \leftarrow V' \cup x_{new}$;

    $x_{min} \leftarrow x_{nearest}$;

    $Q_{near} \leftarrow Near(G, q_{new}, |V|)$;

    for $all q_{near} \in Q_near$ **do**

        if $ObstacleFree(q_{near}, q_{new})$ **then**

            $c' \leftarrow Cost(q_{near}) + c(Line(q_{near}, q_{new}))$;

            if $0 < Cost(q_{new}$ **then**

                $q_{min} \leftarrow q_{near}$;

    $E' \leftarrow E' \cup (q_{min}, q_{new})$;

    for $all q_{near} \in Q_{near} \setminus x_{min}$ **do**

        if $ObstacleFree(q_{near}, q_{new})$ *and*

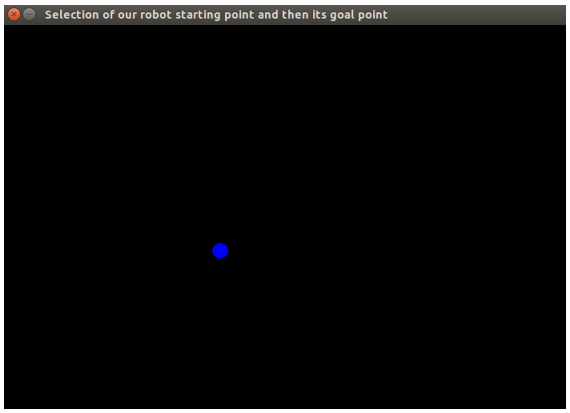
        $Cost(q_{near}) > Cost(q_{new}) + c(Line(q_{new}, qnear))$ **then**

            $q_{par} Parent(q_{near})$;

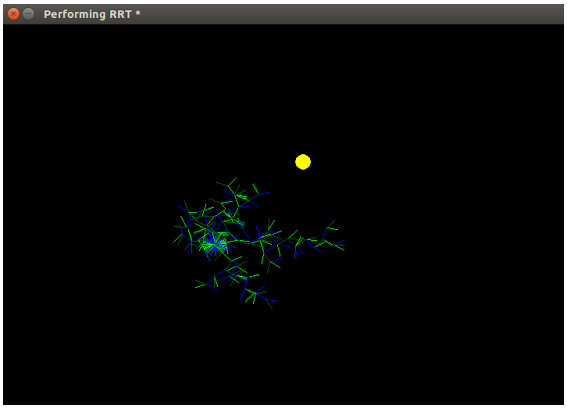
            $E' \leftarrow E' \setminus (q_{par}, q_{near})$;

            $E' \leftarrow E' \cup (q_{new}, q_{near})$;

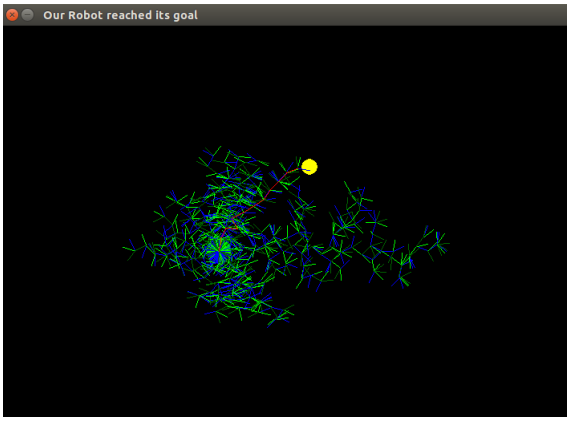
return G



(a) Initialization

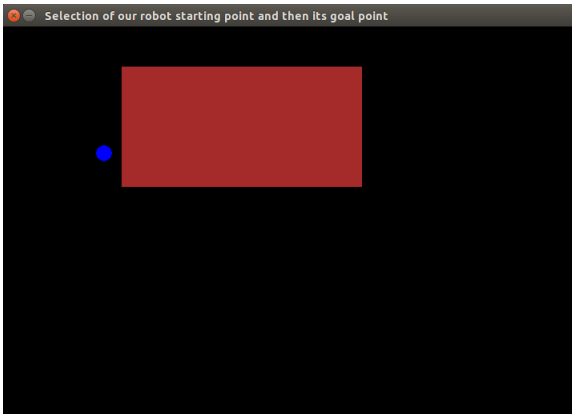


(b) Goal position and exploration phase

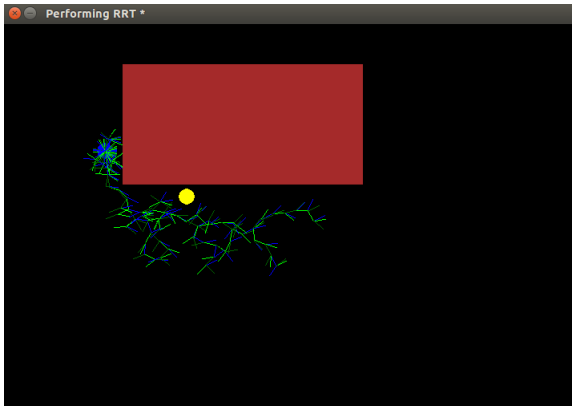


(c) Feasible path found

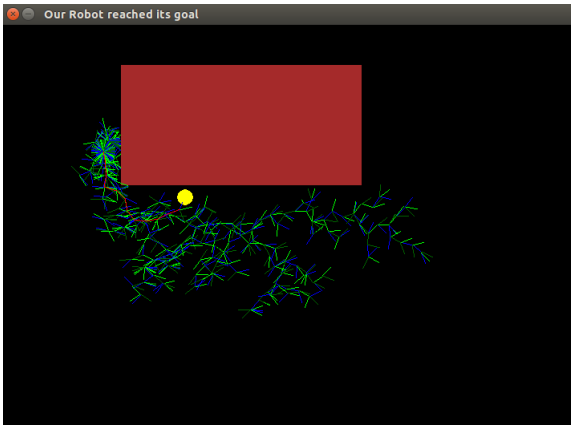
Figure 3.4: Our robot using RRT* in a obstacle-free world



(a) Initialization



(b) Goal position and exploration phase



(c) Feasible path found

Figure 3.5: Our robot using RRT* in a world with one rectangle obstacle

Chapter 4: Motion and Planning of a Robot through Reinforcement Learning Algorithms

4.1 Introduction

In this section, we will be dealing with a robot planning problem, in order to take a deeper dive in the artificial intelligence planning field. Our robot problem describes a robot that moves in a fixed grid world. The world may have obstacles, walls and borders that the robot should not pass through(limits of the world). The robot's purpose is to successfully navigate to a location where an item is, pick it up and deliver it to a destination point where it drops it. The robot receives negative and positive rewards depending on its actions. Positive rewards are given when the robot picks up the item and delivers it to the destination point. Negative rewards are given when the robot attempts to step outside of the world, pass through a wall, try and pick up in the wrong point and deliver on the wrong point.

4.1.1 MDP

Specifically, we will be dealing with a Markov Decision Process problem (MDP) which is a discrete Stochastic Control Processes, which in general are used to solve many optimization problems. MDPs can be solved with the use of Bellman equation and are an

extension of Markov chains. We will be using 2 different algorithmic methods the Q-Learning and the Sarsa which are Reinforcement Learning algorithms and are used to find the optimal policy, which is a behaviour that produces an optimal control output and can model our problem as an MDP.

4.2 Problem Formulation

4.2.1 Assumptions

The Data will be taken from our robot's world. This means each time we will place the object at a different position and therefore our robot will try to learn all possible combinations that may occur at its world. We formulate our problem with the following assumptions:

- State variables: robotLocation {1, ..., 25}, itemLocation {1, ..., 5} (i.e. waiting at pickup/drop-off {R,G,B,Y} or in the robot), drop-offLocation {1, ..., 4} (i.e. {R,G,B,Y}).
- Initialisation of a trail: Robot is uniformly randomly in any of the 25 grid squares, itemLocation is uniformly randomly in one of the 5 item states, dropoffLocation is uniformly randomly one of the 4 drop-off locations
- Termination of a trial: item was successfully dropped-off or after a time constraint (item just wants to get out off the robot and does not care where)
- Actions: 1: go north, 2: go south, 3: go west, 4: go east, 5: pick up item, 6: drop off

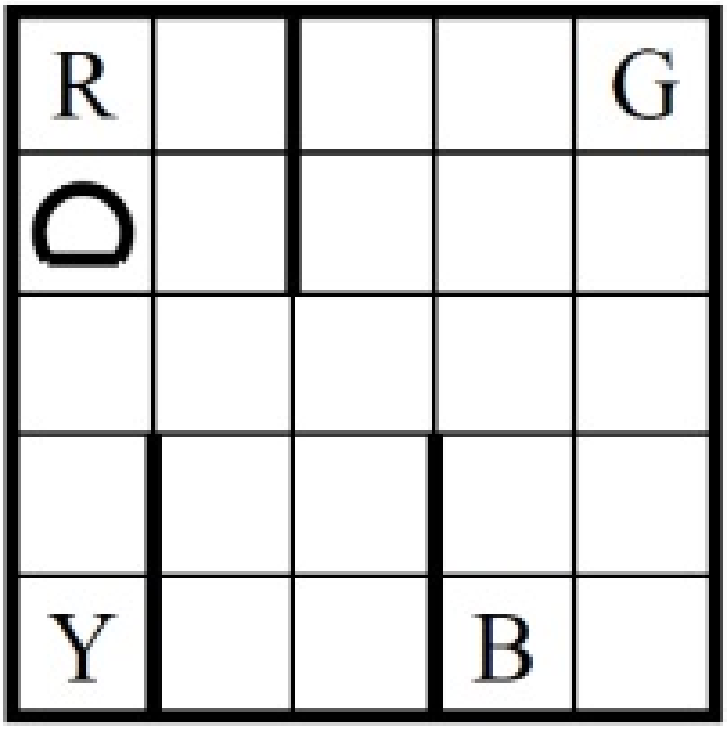


Figure 4.1: The 25 grid squares of our robot's world

item

- Reward is 0, except in the following cases: -1 for an unsuccessful movement (e.g. if blocked by a wall), 1 for a successful pick-up, 10/(number of steps since pick-up) for a successful drop-off, -1 for an attempted drop-off with no item or at the wrong location, -1 for an attempted pick-up at the wrong location (or if the item is already on the robot).

4.3 Dynamic formulation of our Problem

One can easily observe that every state in our problem has 3 variables in order to explicitly formulate the environment in our world and that is that every state consists of the

Robot's location, the Item location and the Drop off location.

- Let s be the variable of our state.
- Also let $x_t \in S = \{x_1, x_2, ...x_{25}\}$ denote our robot's position, as our robot can move and visit every single square of our world.
- And $p \in P = \{x_1, x_5, x_{20}, x_{21}, x_0\}$ denote our item's position, as our item can be either on one of the R,Y,B,G positions or grabbed from the robot (where x_0 stands for item being on the robot).
- Also let $g \in G = \{x_1, x_5, x_{20}, x_{21}\}$ denote our drop-off location, as our drop-off location can be either on one of the R,Y,B,G positions.
- Let $a_t \in A = \{a_1, a_2, a_3, a_4, a_5, a_6\}$ be our possible set of actions, where those actions stand for : go north, go south, go west, go east, 5: pick up item, drop off item respectively and for every move our robot picks a different one.
- We denote t_1 the time the robot successfully picks up the item at p
- And t_f the time the robot drops off the item successfully at the correct drop off location g

Therefore one can completely represent a state in our problem as:

$$s = \begin{bmatrix} x_t & p & g \end{bmatrix} \tag{4.1}$$

For every single sub-problem; the position of our item p and our drop-off location g are fixed and the only variable that changes for every step is x_t and of course the action that the

robot chooses at every move a_t. The robot as we know can perform one out of 6 possible actions.

The dynamics of x_t of the robot's position are going to be :

$$x_{t+1} = f(x_t, a_t), \forall t \in [0, t_f] \tag{4.2}$$

where for our function $f(x_t, a)$ we use the following Look up Table 4.1, in order for our robot to make every step from a state to the next one.

Every problem with a different initialization to our variables x_t, p and g can be tackled as two separate sub-problems. That is, the first sub-problem is until our robot picks up the item and the second one is after he has picked it up, until it drops the item to its drop-off location.

That means that the first of our sub-problem ends when $x_{t_1} = p$ at some time $t_1 > 0$ and at that time t_1 the second sub-problem begins and stops at time $t_f > t_1$ when $x_{t_f} = g$.

For the first sub-problem (until the robot picks up the item) the cost function $g_1[x_t, a_t]$ is based on the rewards that are defined for every move that the Robot makes and it is given below.

where r* at Table 4.2 stand for:

$$\text{r*} = \begin{cases} g[s_1, a_5] = 1 & \text{if } p = s_1, \text{else } \ g[s_1, a_5] = -1 \\ g[s_5, a_5] = 1 & \text{if } p = s_5, \text{else } \ g[s_5, a_5] = -1 \\ g[s_{20}, a_5] = 1 & \text{if } p = s_{20}, \text{else } \ g[s_{20}, a_5] = -1 \\ g[s_{21}, a_5] = 1 & \text{if } p = s_{21}, \text{else } \ g[s_{21}, a_5] = -1 \end{cases}$$

Previous State - Action - Next state (f-function)						
Robot Position	Go north(a1)	Go south(a2)	Go west(a3)	go east(a4)	Pick-up (a5)	drop off (a6)
s_1	s_1	s_2	s_1	s_6	s_1	s_1
s_2	s_1	s_3	s_2	s_7	s_2	s_2
s_3	s_2	s_4	s_3	s_8	s_3	s_3
s_4	s_3	s_5	s_4	s_4	s_4	s_4
s_5	s_4	s_5	s_5	s_5	s_5	s_5
s_6	s_6	s_8	s_1	s_6	s_6	s_6
s_7	s_6	s_8	s_1	s_7	s_7	s_7
s_8	s_7	s_9	s_3	s_{13}	s_8	s_8
s_9	s_8	s_{10}	s_9	s_{14}	s_9	s_9
s_{10}	s_9	s_{10}	s_{10}	s_{14}	s_{10}	s_{10}
s_{11}	s_{11}	s_{12}	s_{11}	s_{16}	s_{11}	s_{11}
s_{12}	s_{11}	s_{13}	s_{12}	s_{17}	s_{12}	s_{12}
s_{13}	s_{12}	s_{14}	s_8	s_{18}	s_{13}	s_{13}
s_{14}	s_{13}	s_{15}	s_9	s_{14}	s_{14}	s_{14}
s_{15}	s_{14}	s_{15}	s_{10}	s_{15}	s_{15}	s_{15}
s_{16}	s_{16}	s_{17}	s_{11}	s_{21}	s_{16}	s_{16}
s_{17}	s_{16}	s_{18}	s_{12}	s_{22}	s_{17}	s_{17}
s_{18}	s_{17}	s_{19}	s_{13}	s_{23}	s_{18}	s_{18}
s_{19}	s_{18}	s_{20}	s_{19}	s_{24}	s_{19}	s_{19}
s_{20}	s_{19}	s_{20}	s_{20}	s_{25}	s_{20}	s_{20}
s_{21}	s_{21}	s_{22}	s_{16}	s_{21}	s_{21}	s_{21}
s_{22}	s_{21}	s_{23}	s_{17}	s_{22}	s_{22}	s_{22}
s_{23}	s_{22}	s_{24}	s_{18}	s_{23}	s_{23}	s_{23}
s_{24}	s_{23}	s_{25}	s_{19}	s_{24}	s_{24}	s_{24}
s_{25}	s_{24}	s_{25}	s_{20}	s_{25}	s_{25}	s_{25}

Table 4.1: The Transition Matrix for our Robot's Dynamics, our look-up Table

Note that the * symbol next to the r in the above table means that for the first sub-problem with $t \in [0, t_1]$ for every new problem, only one of those ones is one depending on our item location in the current problem and all the remaining 3 ones are -1. (i.e Assume our Item is on $p = s_1$ then only $g[s_1, a_5] = 1$, whereas $g[s_5, a_5] = -1$, $g[s_20, a_5] = -1$ and $g[s_21, a_5] = -1$)

Current State - Action - Reward						
Robot Position	Go north(a1)	Go south(a2)	Go west(a3)	go east(a4)	Pick-up (a5)	drop off (a6)
s_1	-1	0	-1	0	r*	-1
s_2	0	0	-1	0	-1	-1
s_3	0	0	-1	0	-1	-1
s_4	0	0	-1	-1	-1	-1
s_5	0	-1	-1	-1	r*	-1
s_6	-1	0	0	-1	-1	-1
s_7	0	0	0	-1	-1	-1
s_8	0	0	0	0	-1	-1
s_9	0	0	-1	0	-1	-1
s_{10}	0	-1	-1	0	-1	-1
s_{11}	-1	0	-1	0	-1	-1
s_{12}	0	0	-1	0	-1	-1
s_{13}	0	0	0	0	-1	-1
s_{14}	0	0	0	-1	-1	-1
s_{15}	0	-1	0	-1	-1	-1
s_{16}	-1	0	0	0	-1	-1
s_{17}	0	0	0	0	-1	-1
s_{18}	0	0	0	0	-1	-1
s_{19}	0	0	-1	0	-1	-1
s_{20}	0	-1	-1	0	r*	-1
s_{21}	-1	0	0	-1	r*	-1
s_{22}	0	0	0	-1	-1	-1
s_{23}	0	0	0	-1	-1	-1
s_{24}	0	0	0	-1	-1	-1
s_{25}	0	-1	0	-1	-1	-1

Table 4.2: Cost function (Metric) for $t \in [0, t_1]$

Therefore, having our dynamics and our metric we can define our first sub-problem as:

$$\begin{aligned} \min_{t_1} \max_{a_t \in \mathbb{A}} \quad & \sum_{t=0}^{t_1} g_1(x_t, a_t) \\ \text{subject to} \quad & x_{t+1} = f(x_t, a_t) \\ & x_{t_1} \in P \end{aligned} \tag{4.3}$$

For the second sub-problem(after the robot has picked up the item and until it success-

fully drops it off) , when $p = s_0$ which means that our item is on the robot, the cost function $g_2[x_t, a_t]$ is based on the rewards that are defined for every move that the robot makes, is given by the Table 4.3.

Current State - Action - Reward						
Robot Position	Go north(a1)	Go south(a2)	Go west(a3)	go east(a4)	Pick-up (a5)	drop off (a6)
s_1	-1	0	-1	0	-1	r**
s_2	0	0	-1	0	-1	-1
s_3	0	0	-1	0	-1	-1
s_4	0	0	-1	-1	-1	-1
s_5	0	-1	-1	-1	-1	r**
s_6	-1	0	0	-1	-1	-1
s_7	0	0	0	-1	-1	-1
s_8	0	0	0	0	-1	-1
s_9	0	0	-1	0	-1	-1
s_{10}	0	-1	-1	0	-1	-1
s_{11}	-1	0	-1	0	-1	-1
s_{12}	0	0	-1	0	-1	-1
s_{13}	0	0	0	0	-1	-1
s_{14}	0	0	0	-1	-1	-1
s_{15}	0	-1	0	-1	-1	-1
s_{16}	-1	0	0	0	-1	-1
s_{17}	0	0	0	0	-1	-1
s_{18}	0	0	0	0	-1	-1
s_{19}	0	0	-1	0	-1	-1
s_{20}	0	-1	-1	0	-1	r**
s_{21}	-1	0	0	-1	-1	r**
s_{22}	0	0	0	-1	-1	-1
s_{23}	0	0	0	-1	-1	-1
s_{24}	0	0	0	-1	-1	-1
s_{25}	0	-1	0	-1	-1	-1

Table 4.3: Cost function (Metric), fot $t \in (t_1, t_f]$

where r** at Table 4.3 stand for:

$$\text{r**} = \begin{cases} g[s_1, a_5] = 10/(t_f - t_1) & \text{if } g = s_1, \text{else } \ g[s_1, a_5] = -1 \\ g[s_5, a_5] = 10/(t_f - t_1) & \text{if } g = s_5, \text{else } \ g[s_5, a_5] = -1 \\ g[s_{20}, a_5] = 10/(t_f - t_1) & \text{if } g = s_{20}, \text{else } \ g[s_{20}, a_5] = -1 \\ g[s_{21}, a_5] = 10/(t_f - t_1) & \text{if } g = s_{21}, \text{else } \ g[s_{21}, a_5] = -1 \end{cases}$$

Therefore, having our dynamics and our metric we can define our second sub-problem as:

$$\begin{aligned} \min_{t_f} \max_{a_t \in \mathbb{A}} \quad & \sum_{t=t_1+1}^{t_f} g_2(x_t, a_t) \\ \text{subject to} \quad & x_{t+1} = f(x_t, a_t) \\ & x_{t_f} \in G \end{aligned} \tag{4.4}$$

Solving the 2 above optimization sub-problems, gives us the optimal solution to our problem for all the different initial s(t=0).

The Bellman's equation tells us that: $V_t^*(j) = \min_{a_t \in \mathbb{A}} [g[j, a_t] + \sum_{i=0}^{t_f} P_{ij} V_t^*(i)]$, where $V_t^*(j)$ is the optimal cost of state j at time t and the $\sum_{i=0}^{t_f} P_{ij} V_t^*(i)$ is the cost-to-go. And because Bellman's equation follows the principle of optimality, in order to find the minimum of $V_t^*(x)$, that means that our solution up until that time is optimal. Same happens with our problem and the two sub-problems. Finding the minimum t_f requires we have found the optimal t_1.

Solving the Bellman's equation in our Problem we followed two different Reinforcement Learning algorithms. The Q-Learning Algorithm and the Sarsa algorithm.

4.4 Estimates

4.4.1 Typical Time horizon

A typical time horizon for a finite Markov Decision Process is the number of future steps that affects the value of any given state[27]. The time horizon value is calculated by the formula $\frac{1}{1-\gamma}$, with γ being the discount factor. In the case of γ = 0.9 the time horizon is $\frac{1}{1-0.9} = 10$ steps. This means that for a state s0 the value contained in the state-value matrix is affected by values of 10 future steps. In addition, by deciding the typical horizon of the problem we can determine the γ value. Given that our grid world is a 5×5 fixed grid (Figure 4.1) a horizon of 9 steps would be enough for designing a good solution which means that $\gamma = \frac{8}{9}$ should be sufficient for the first experiments. The best horizon for this problem is 9 due to the fact that the biggest distance that the robot needs to make, is the distance from the two corners of the 5x5 grid world which equals to 9 cells.

4.4.2 Maximal value of state-action pair

The maximum value of a state action pair can be bounded for $\gamma < 1$ by $\frac{r_{max}}{1-\gamma}$. In order to calculate the upper bound of this value we need to calculate r_{max} .

Given our state vector = [robot position , item position, goal], our rewards per action and a good policy π , the maximum reward that we can get is when our state is initialized with the robot and the item at the same position of our 5x5 grid world and the item is not on the robot. This means that in the first step the robot will receive r=1 for picking up the item. In the second step the robot will deliver the item and receive $r = \frac{10}{numofsteps}$ with number of

steps=1 since the drop off point is the same as the pick point. Hence, $r_{max} = 1 + \frac{10}{1} = 11$.

Given r_{max}=11 and $\gamma = 0.9$ the maximum value of a state action pair is

$$maxQ(state, action) \leq \frac{r_{max}}{1-\gamma} \Rightarrow maxQ(state, action) \leq 110 \tag{4.5}$$

This value can be introduced to the design of our algorithm with the use of optimistic initialization. When we use optimistic initialization, instead of randomly initializing the values of the Q-matrix(state-action matrix) we set them equal to the maximum value that they may reach which is equal to 110 (eq. (4.5)) in our case.

4.4.3 Number of trials that a standard algorithm will need to find a good solution

To determine the maximum number of trials that an algorithm needs to find a solution we must first define our approach for solving it. Our robot problem satisfies the Markov property and hence is a Marcovian Decision Process(MDP)[27]. The simplest approach to find a solution of an MDP is to exhaustively search the total number of deterministic policies which in our case requires 6^{25*5*4} iterations. A much better approach would be to use dynamic programming to solve the problem. Dynamic programming guarantees to find a solution in polynomial time of the states and actions.

4.4.4 Deterministic or Stochastic

In a stochastic problem the result of an action at time T in a specific state can be different than the result produced at time T+N with the same state-action pair. This means that the reward for a given state-action pair is given with some probability. On the other hand in a deterministic problem a state-action pair always yields the same reward. Given the above and the nature of our problem, if the robot makes an action a_0 at state s_0 it will always receive the same reward. Thus the problem is deterministic.

4.5 Q-learning

4.5.1 Short Description and Implementation of the algorithm

The robot problem describes an agent(robot) that moves in a fixed grid world. The world may have obstacles, walls and borders that the robot should not pass through(limits of the world). The robot's purpose is to successfully navigate to a location where an item is, pick it up and deliver it to a destination point where it drops it. The robot receives negative and positive rewards depending on its actions. Positive rewards are given when the robot picks up the item and delivers it to the destination point. Negative rewards are given when the robot attempts to step outside of the world, pass through a wall, try and pick up in the wrong point and deliver on the wrong point.

For the implementation of the algorithm the state-action matrix(Q) must be firstly defined. To create Q we need to take into account all the possible actions and states of the problem. Hence Q is a four dimensional tensor with size 25x5x4x6 containing every pos-

sible combination of all the positions of the robot, the positions of the item, the drop off locations and the actions. The state-value matrix contains only the states of our problem which means it is a 3 dimensional 25x5x6 matrix.

4.5.2 Performance of the algorithm

By storing all the final rewards after each trial of the algorithm and obtaining the max of this vector with size T=number of trials, the maximum reward was 11 which is equal to what we estimated(In Section Estimates). In addition, the mean(immediate) reward for a good policy which is obtained by the mean of the average reward (Figure 4.3, Figure 4.4) is 0.5819. Regarding the number of trials that the algorithm requires to find a solution, the experiments converge at 1000 trials approximately (Figure 4.4). This value is significantly smaller than the exhaustive search which is something that we expected since we use dynamic programming that solves the problem in polynomial time. Finally, the mean reward after each trial is 3.9649.

Below are presented diagrams describing the performance of the algorithm and various experiments about how variables such as gamma and ε effect the convergence of the algorithm. In order to produce these diagrams each test was executed multiple times($samples > 100$) in order to get the mean value of each trial of the different samples to produce smoother and easier to understand curves.

In Figure 4.2 and Figure 4.3 the number of steps and rewards over the number of trials are presented. From this diagram we can see that as the policy changes and the Q-matrix is updated the steps required for the algorithm to find a solution starts from more than

450 steps and is dramatically reduced until trial 200 where it started to converge to value. The reduction and the convergence of this diagrams means that the algorithm makes good decisions which lead it faster to the solution. In Figure 4.3 the rewards follow a positive trend until the 200 step where we can see the rewards per trial are stabilizing and reaching a convergence point. We can see that both diagrams appear to converge at the same point of time.

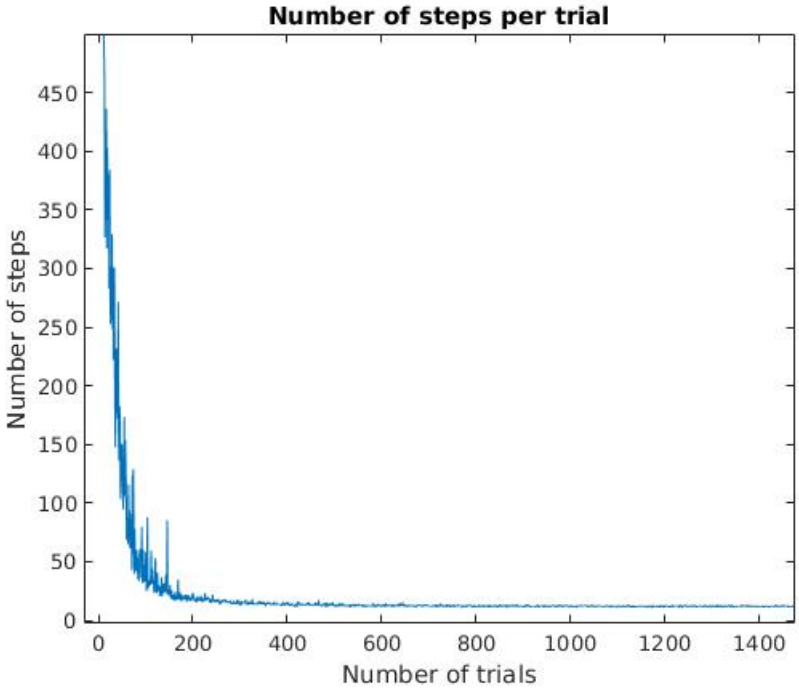


Figure 4.2: Number of steps over learning time

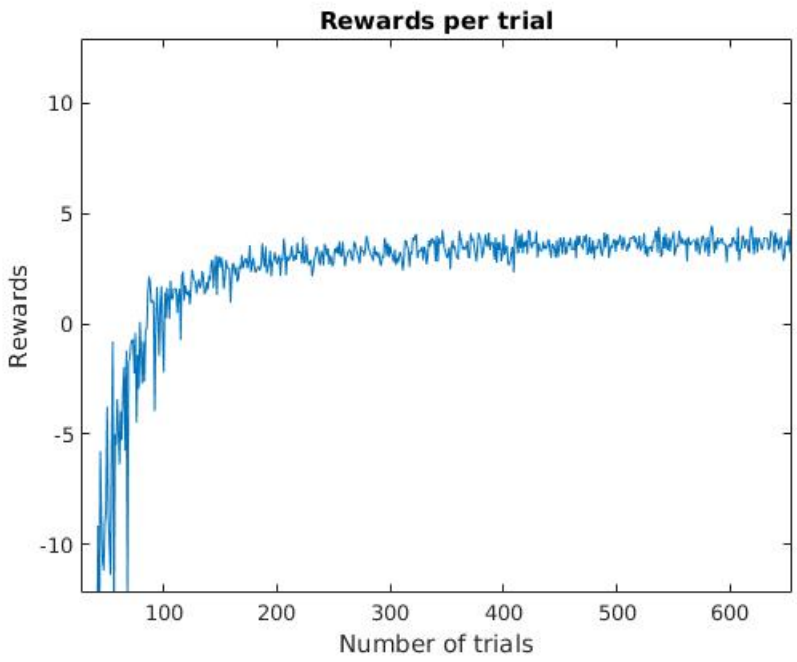


Figure 4.3: Rewards over learning time

The average reward over time (Figure 4.4) experiences a steep increase until the 500 trial and starts to converge towards a value after 1000 trials. We can see that even if the number of steps has already converged at about 200 trials the average rewards converges later at 1000.

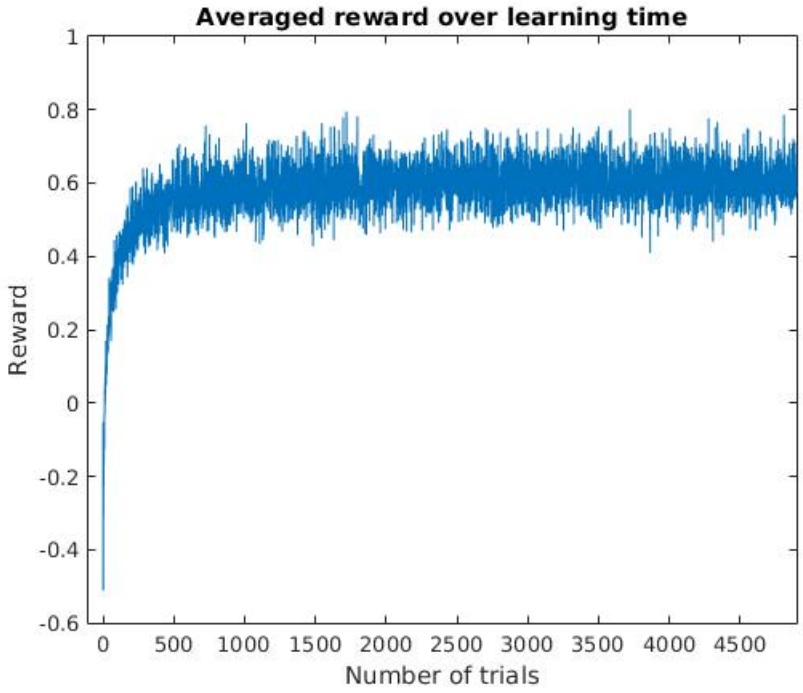


Figure 4.4: Average reward over learning time

This means that even though the algorithms steps do not increase the algorithm does not take the optimal decisions by that time.

4.5.3 Trial of the Q-Learning algorithm

The example presented below is a single trial of the algorithm with the use of policy π after 50000 trials of training. In this example the robot starts at location R, the item is at location B and the destination point is at location R. In Table 4.4 we can see the results of the test run.

Actions 1,2,3,4 denote movements: up,left,down and right while actions 5,6 denote pick up and drop off. The last column is an optimal route that was calculated by a human. As we can see from the table the robot choose the optimal action in every case and received

Test Run				
step	Robot Position	Item Position	Action	Optimal Action
1	1	21	3	3
2	2	21	3	3
3	3	21	4	4
4	8	21	4	4
5	13	21	4	4
6	18	21	4	4
7	23	21	1	1
8	22	21	1	1
9	21	21	5	5
10	21	on robot	3	3
11	22	on robot	3	3
12	23	on robot	2	2
13	18	on robot	2	2
14	13	on robot	2	2
15	8	on robot	1	1
16	7	on robot	2	2
17	2	on robot	1	1
18	1	1	6	6

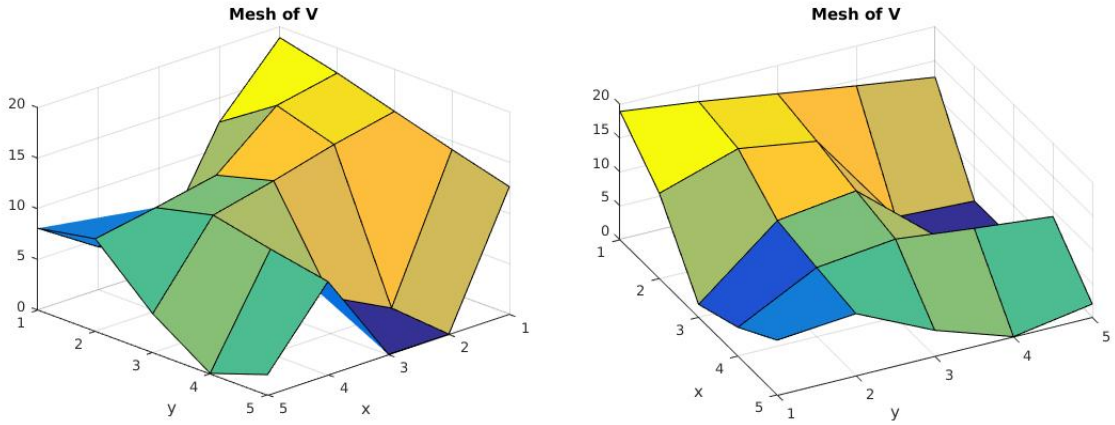
Table 4.4: Example of pick up and drop off using policy π after 50000 trials.

a reward of 2.111 since it did not get penalized, received 1 for pick up and 10/9=1.1 for the pick up.

In Table 4.5 the values of the state-value matrix for the case that the item is being held by the robot with destination R are presented. We can clearly see that the value of point R is the maximum which denotes the significance of the point since it is the destination. In addition, all nearby values are very high. We can see that the further we get from the destination, the values get smaller, especially near the walls.

state-value Matrix				
18.9708	12.6045	1.9142	4.2534	8.0790
17.5652	16.3494	11.4521	10.2307	9.1403
15.8723	14.5921	12.9915	11.5823	3.8271
14.1931	0.5212	3.5503	10.3140	0.0298
12.6160	0.0379	0.0314	9.2017	1.9948

Table 4.5: State value matrix of a policy π when item is held by the robot with destination R.



(a) (b)

Figure 4.5: 3D representation of Table IV

4.5.4 Convergence Time

To define a convergence time we need to evaluate the course of the data. Regarding the average reward from Figure 4.4 we can see that the values converge after a thousand trials. This means that the average reward is reaching its maximum. Following a similar trend the maximum of state-action value increases steadily until the same number of trials(1000) where it starts converging (Figure 4.7). This result also validates the upper bound value that was calculated in eq. (4.5) which denoted that the maximum value of Q-matrix(V contains the maximum values of Q in the action's dimension) will not exceed 120.

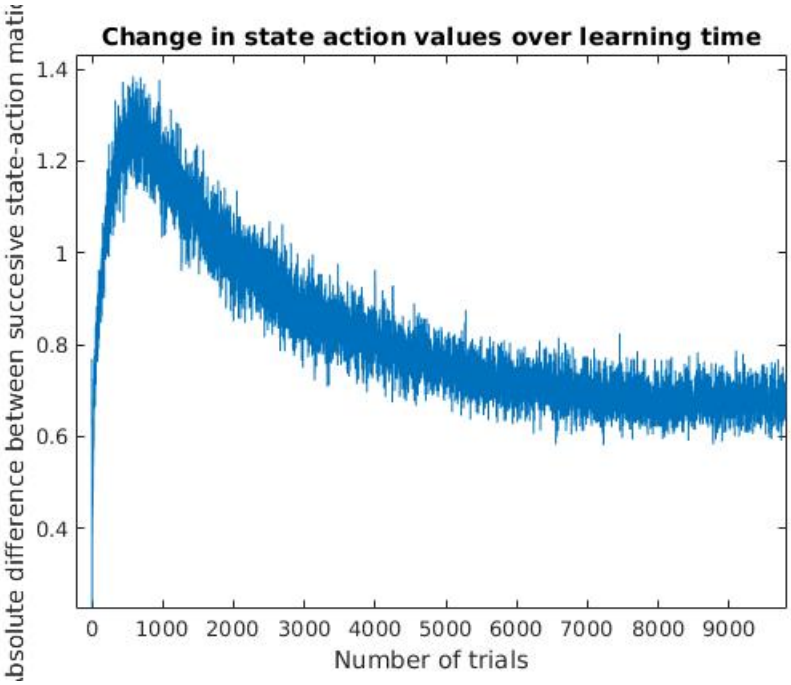


Figure 4.6: Tests of maximum value of V for different gamma values

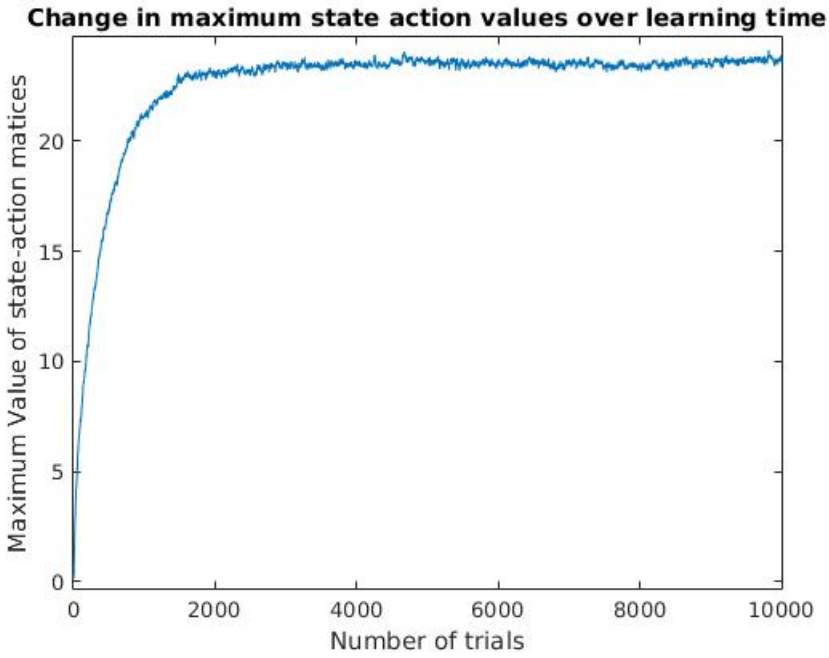


Figure 4.7: Convergence of V and it's maximum value

On the contrary, the absolute difference between successive V-matrices(state-value matrices) keeps decreasing even after eight thousand trials. Without the use of an optimistic initialization we can observe a very fast increase in the difference of successive V-matrices which denotes the fast change in their values after each trial. At the thousandth trial this difference starts to decay and gradually reduces since the problem converges to an optimal solution.

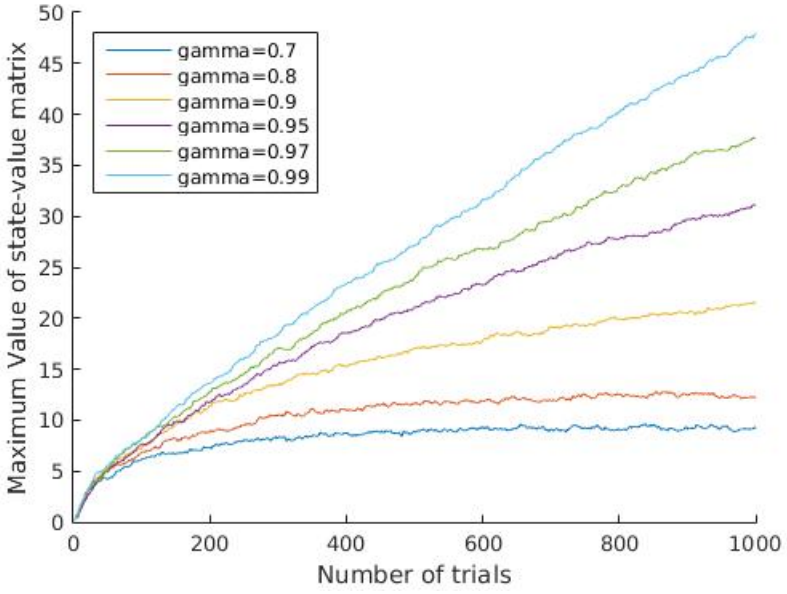


Figure 4.8: Tests of maximum value of V for different γ values

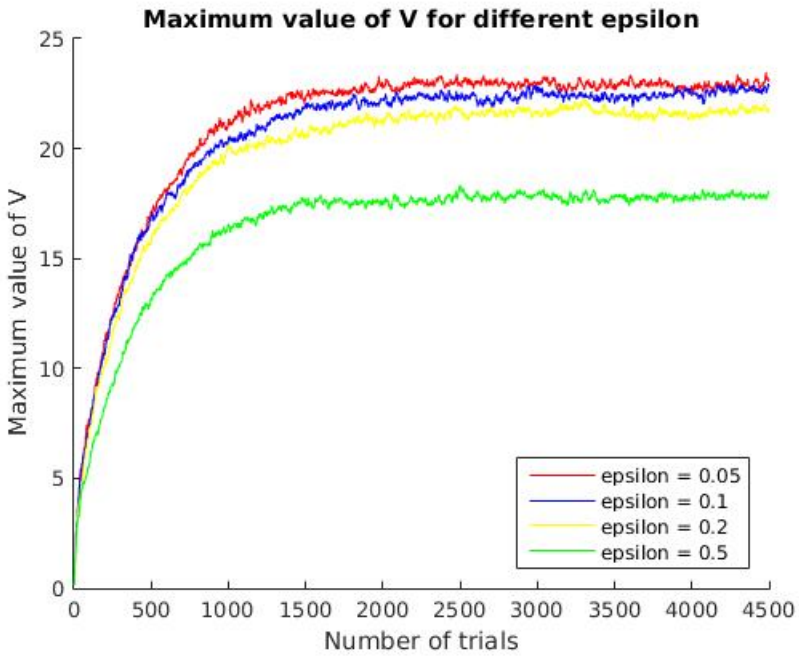


Figure 4.9: Tests of maximum value of V for different ε values

4.5.5 Experimenting with variables and their effect on convergence

In Figure 4.8 we can see how different γ affect the maximum values of the state-action matrix V. From this diagram we can clearly see the correlation between the γ and the horizon. When γ increases, horizon increases too and so does the maximum value of V, from the definition given in section 4.5.1 every value of the V-matrix is affected by future steps equal to the horizon, which means that as the horizon increases more future steps are incorporated in a single cell of the matrix and thus its Values increase too. Figure 4.9 illustrates the effect of different values of exploration on the maximum values of V. We can observe that as the epsilon reduces the maximum value increases. When we have a small exploration our algorithm will not explore much and thus it will end up passing trough the same route every time. Thus specific values will keep increasing. On the other hand when

epsilon is big the algorithm will randomly choose different routs more often and thus the increase will not accumulate only in a few values resulting in lower max values.

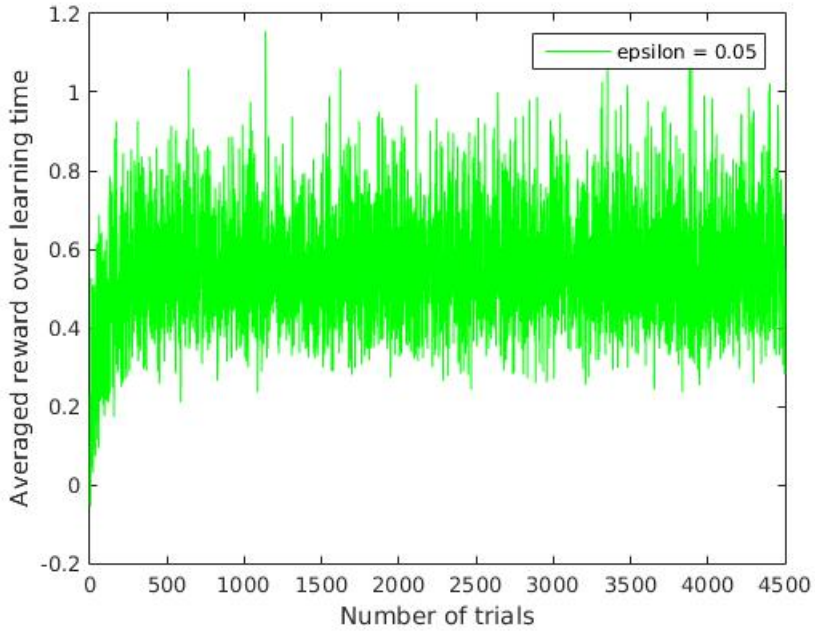


Figure 4.10: Convergence of mean reward for different values of ε. Here ε=0.05

In Figures 4.10, 4.11, 4.12, 4.13 plots of average reward for different values are presented. We can clearly see that as the value of ε decreases so does the average reward. For random actions the robot might get penalised very often. The higher the probability of making a random action the higher is the probability of getting penalised and thus the averaged reward is reduced as the ε increases.

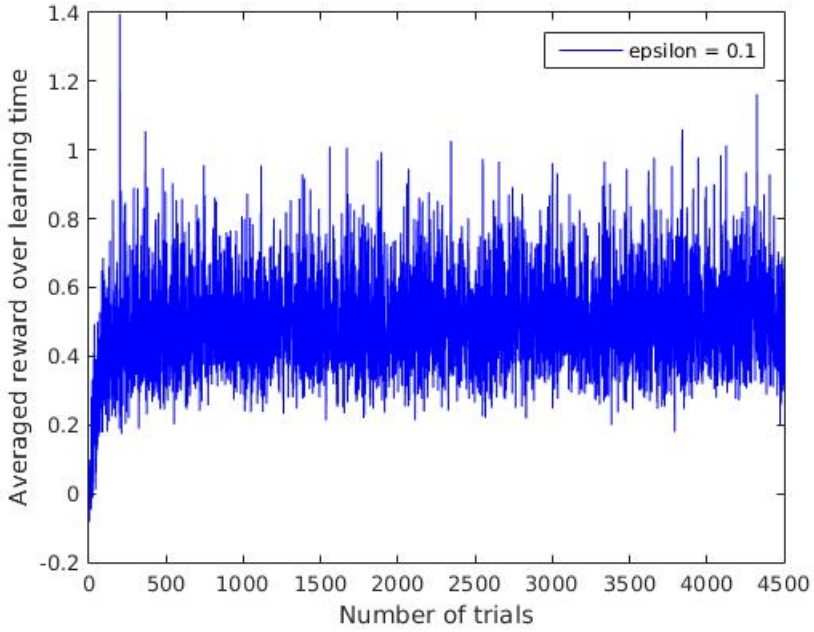


Figure 4.11: Convergence of mean reward for different values of ε. Here ε=0.1

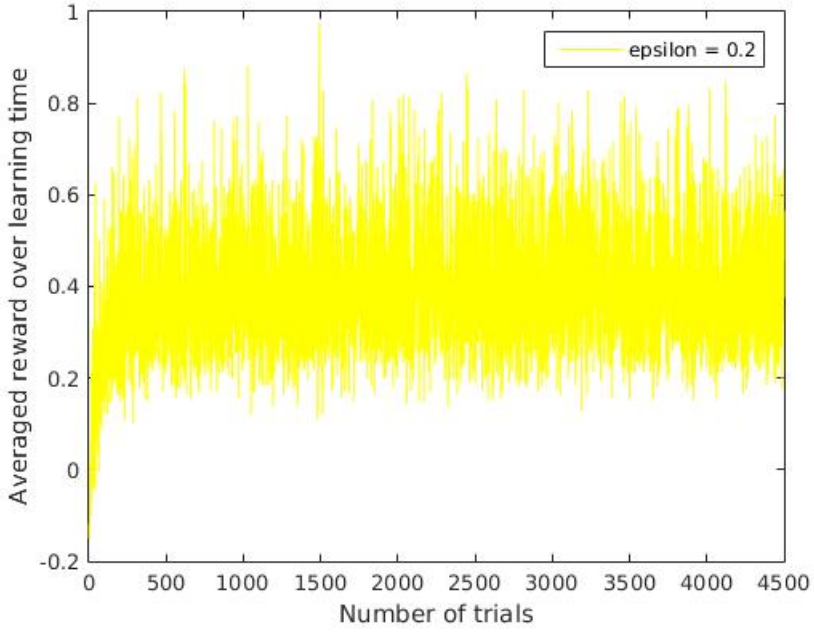


Figure 4.12: Convergence of mean reward for different values of ε. Here ε=0.2

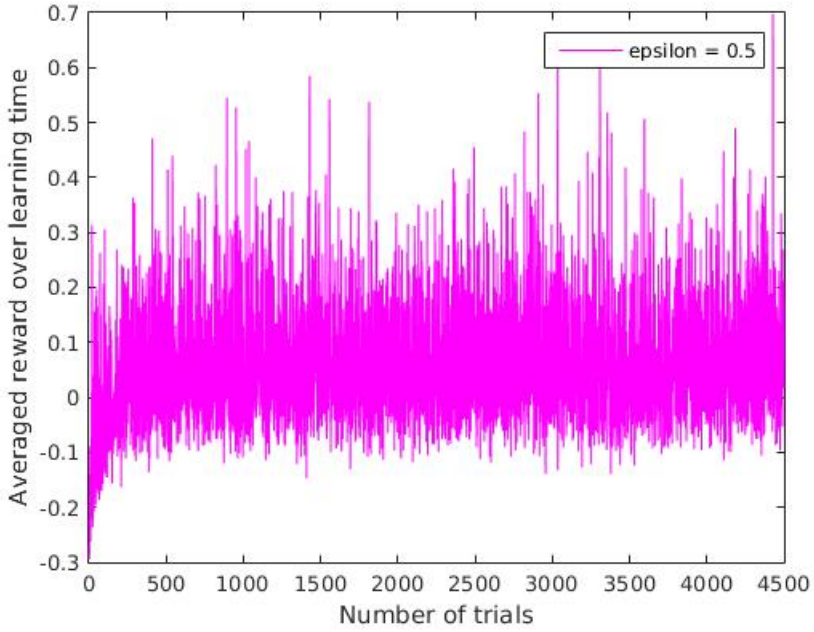


Figure 4.13: Convergence of mean reward for different values of ε. Here ε=0.5

4.6 SARSA

In this section we will be testing and evaluating another MDP. SARSA, unlike Q-Learning, is an on-policy method. In these kind of methods instead of estimating a state-action matrix we consider transitions between state-action pairs[27]. Regarding the modeling of the world and the algorithm remains the same and only the update function of Q changes.

4.6.1 Performance of the algorithm

A SARSA algorithm trained for a hundred thousand trials produced a mean of steps per trial equal to 11.2131. Moreover, the mean(immediate) reward was calculated to be 0.63

and the mean reward per trial is 4.0095.

Figure 4.14 presents us the rewards per trial for the algorithm. The final reward for each trial stabilizes after 200 trials. Similarly Figure 4.15 presents us the number of steps that the algorithm needs to find a solution over trials. Both diagrams converge at about 200 trials. In addition, the mean reward is illustrated in Figure 4.16 where there is a sudden increase in its value followed by convergence at approximately a thousand trials.

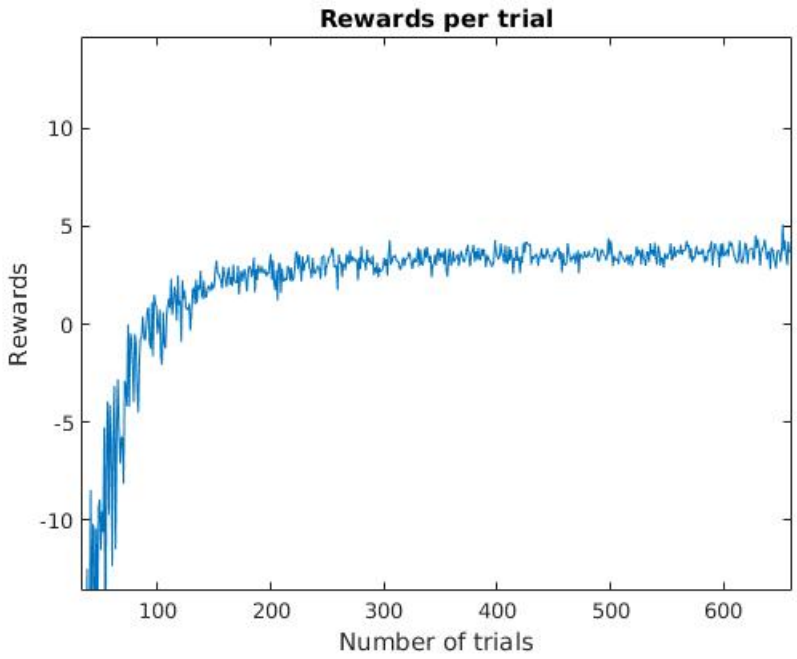


Figure 4.14: Rewards per trial with Sarsa

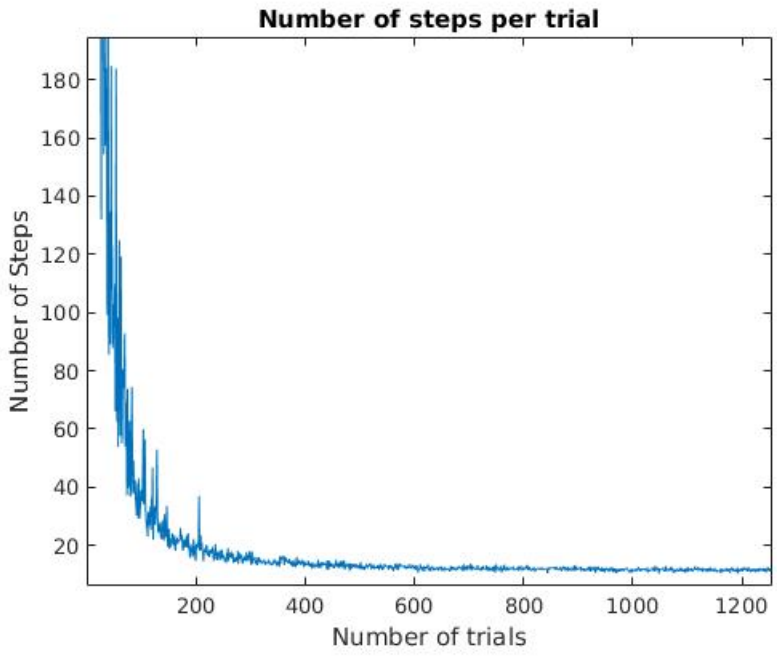


Figure 4.15: Number of steps over learning time

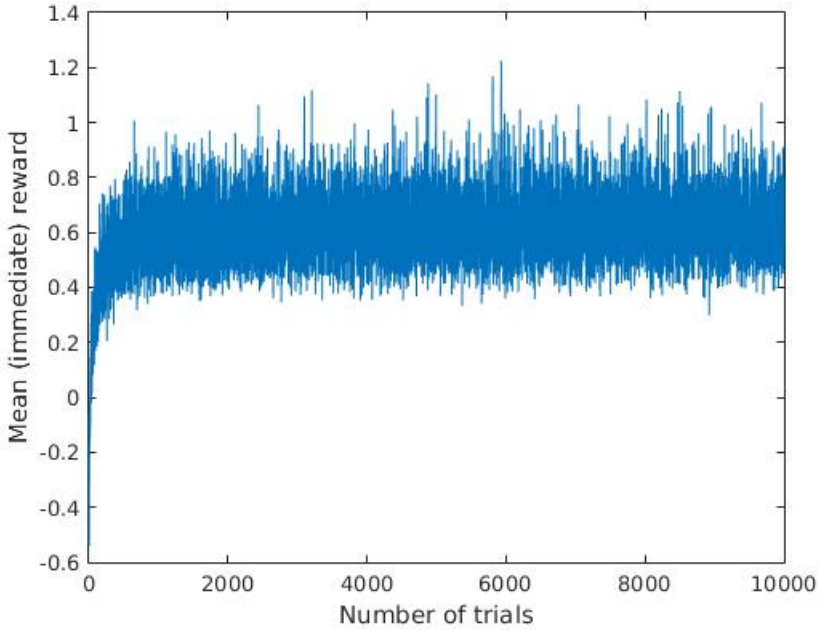


Figure 4.16: Mean (immediate) reward over time

4.6.2 Convergence time

Regarding the convergence time of the algorithm we can examine the convergence of the maximum value of Q to determine it. We can see from Figure 4.17 that the maximum value of Q stops increasing after a thousand trials. In addition to that the mean(immediate) rewards converges at the same time.

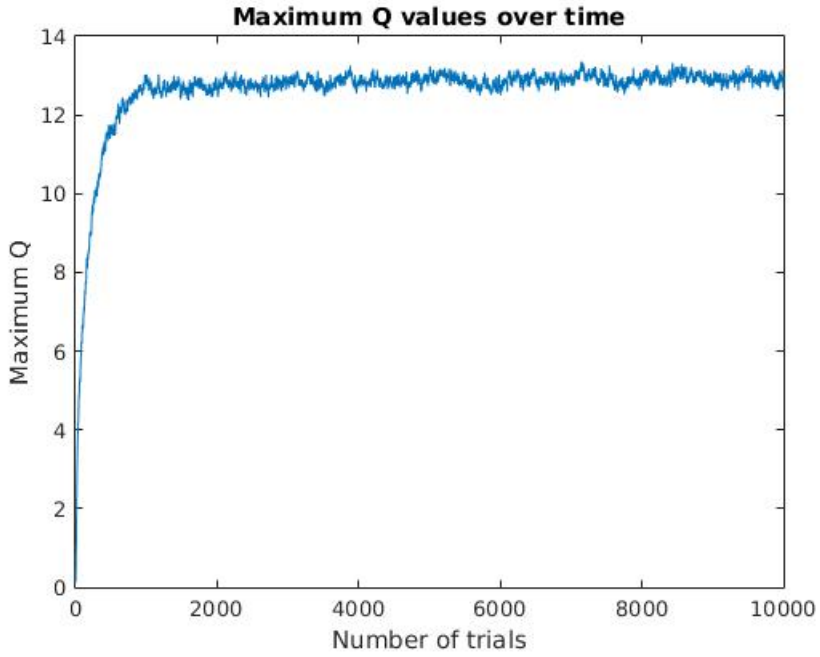


Figure 4.17: Rewards and number of steps per trial

4.6.3 Experimenting with variables and their effect on convergence

Given the diagrams in Figure 4.18, we can clearly see that for different values of γ the mean(immediate) reward remains unchained. This happens due to the fact that γ affects the horizon and not the actions of the robot.

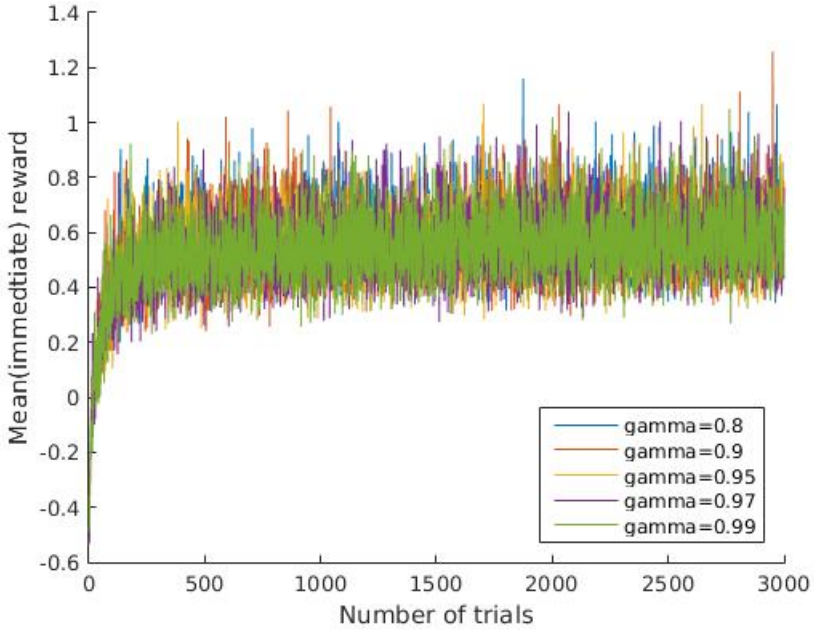


Figure 4.18: Average reward for different γ

On the contrary in Figure 4.19 we can see the obvious effects of different ε on the immediate rewards. As it is mentioned in the section for Q-Learning the higher the ε the higher is the probability to make a random move which is non optimal.

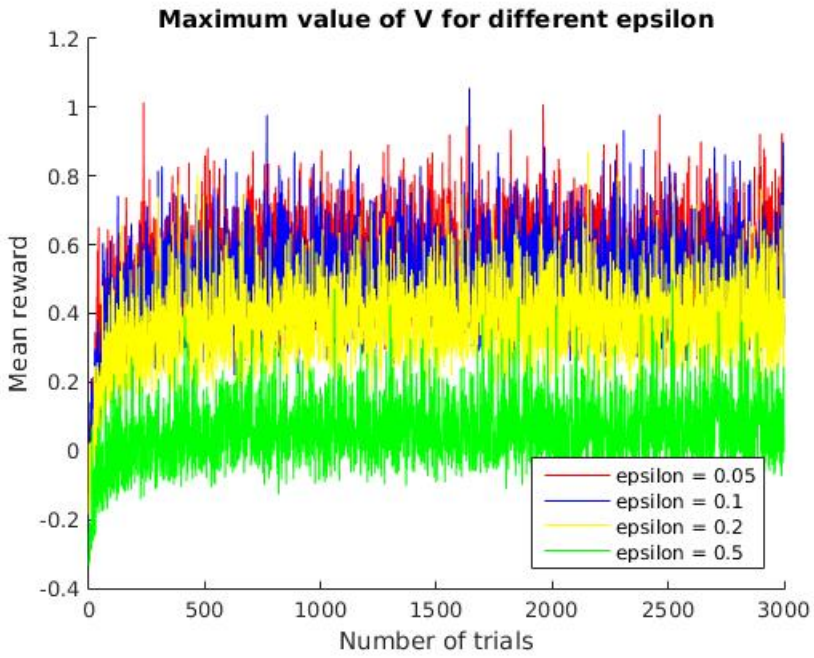


Figure 4.19: Average reward for different ε

4.7 Comparison of SARSA and Q Learning

For comparison purposes we will test how a SARSA algorithm with a good policy responds to the same example that we tested Q-learning with. The problem to test has the robot initialized at point R and the item at point G with destination point R. Similar with the previous experiment the SARSA algorithm is trained for a hundred thousand trials and then the Q-matrix is reused for a singe experiment.

From this experiment we can clearly see that both Q-Learning and SARSA solve the problem with the same amount of steps, receive the same rewards and are both optimal. An interesting observation is that although SARSA's solution is optimal too, it uses a different route to navigate to the pick up point and deliver to the destination both ways different

from Q-Learning route. On the other hand Q-Learning uses the same route for going and returning since the starting point is also the destination point.

Test Run					
step	Sarsa Robot Position	Sarsa Action	Item Position	Q-Learning Action	Q-Learning robot position
1	1	3	21	3	1
2	2	3	21	3	2
3	3	4	21	4	3
4	8	4	21	4	8
5	13	4	21	4	13
6	18	1	21	4	18
7	17	1	21	1	23
8	16	4	21	1	22
9	21	5	21	5	21
10	21	2	on robot	3	21
11	16	3	on robot	3	22
12	17	2	on robot	2	23
13	12	3	on robot	2	18
14	13	2	on robot	2	13
15	8	2	on robot	1	8
16	3	1	on robot	2	3
17	2	1	on robot	1	2
18	1	6	1	6	1

Table 4.6: Step by step comparison of Q-Learning and SARSA in the same problem

The number of steps per trial, the mean of the average rewards per trial are almost the same for both approaches.In contrast to that the values of the Q matrix for each method differ significantly. The values of SARSA are smaller than the values of Q-Learning. Finally both algorithms may produce optimal solutions to a problem but different, This is due to the fact that a different approach is used by the algorithms.

4.8 Improve speed

In order to increase the learning speed of the algorithm and given the compositional structure of the problem we can reduce the state space. We can consider the robot moving to pick up and drop off the same action and remove the pick up variable in the state vector. Hence our resulting state space becomes 25x4. The information of the item, whether is on the robot or not can be stored in another variable and not in the space vector.

4.9 Conclusions

In this section, two MDP algorithms were developed. Experiments were conducted about the influence of their variables (γ, ε) to the convergence time, the mean reward and the values of their variables. Variable γ affects the values of the Q-matrices since it introduces information from the future. The bigger the γ the bigger the information and thus the values of matrices. Variable ε is used to avoid being greedy in our algorithms. High values of ε introduce a lot of randomness in the robot's action which causes it to take non-optimal decisions. Both variables play a major role in the convergence of the algorithm and their values should be decided after consideration and experimentation in order to produce good policies that solve reinforcement learning problems.

Chapter 5: Simulating motion planning of the Baxter robot

5.1 The Baxter robot and the simulator

In the previous chapters we analyzed the dynamics of the Baxter robot and we dived into some planning and motion planning algorithms. Based on our previous steps, we now move on and deal with planning the Baxter robot. As we have already seen, Baxter robot has 2 arms with seven degrees of freedom (DOF) and series of joint actuators which make it a unique manufacturing robot. Those joints are composed of a series of elastic actuators (SEAs) which provide flexibility for control. Baxter has seven rotary joints as shown in the following figure. Each arm is often referred to as a 7-DOF arm, since motion of the arm is controlled by seven actuators (motors) that are capable of independent rotation. As we did with the dynamic model of the Baxter robot where we analyzed just the one arm as the other one is symmetric, the same will happen with our work here.

We all know that simulation is a technique to replace real life experiences and can be a technique used for practising and learning and a method of reproducing aspects of real life in an interactive way. Of course, the best thing one can do before implementing something in the real world, is to check his work and his results in on a simulator.

For our work, testing our planner and working with the pre-existing ones, we used a simulated Baxter humanoid robot.

Gazebo is a powerful simulator that attempts to emulate physics and system dynamics in a more accurate way. One of the difficult problems in robotics is to define a path for the motion of a robot's arms to grasp an object, especially when obstacles may obstruct the most obvious path of motion. Fortunately, a ROS package called MoveIt! allows us to plan and then execute a complicated trajectory, taking into consideration the obstacles in our environment.

Our work was simulated through Gazebo, but mainly through OMPL [26] and Moveit! [25], and was visualized with the use of RViz package. Due to the fact that we were using ROS the best way to use OMPL was through Moveit!. Rviz is a 3D visualization package tool for ROS and we used it to visualize Baxter's current configuration on a virtual model. MoveIt! is a very powerful planning framework built into ROS which allows the robot to plan around obstacles in the environment, among other things. Also MoveIt! comes with a plugin for the ROS Visualizer (RViz), which allowed us to setup different scenes in which the robot will work. Moreover it allowed us to generate plans, for which we mainly used the Open Motion Planning Library. It also gave us the opportunity to visualize any outputs and interact directly with our simulated Baxter robot.

5.2 Setting up our environment

In our work we set up, open and use both Moveit! through Rviz and Gazebo simultaneously and we simulate the kinematic motion of the Baxter simulator in both platforms. We use them to display live representations of planning of the Baxter robot.

Due to the fact that the whole setup process by itself is not trivial at all, we will discuss some crucial steps that had to be made in order to have everything smoothly running.

First of all, we must mention that we used Ubuntu 14.04 for our work. We then installed the ROS indigo on our machine in order to use Moveit! which is a very powerful planning framework built into ROS as we mentioned. By ROS installation Rviz was compiled and build and therefore we could proceed on Moveit! later on. We first installed the latest version of Gazebo simulator [4] and then Moveit! [25]. In particular we worked with the indigo version of Moveit! because the kinetic one did not officially support the Baxter yet. We then incorporated ompl.1.3.0. [26] because that version seemed to have no compatibility issues with Moveit! and the c++ compiler of ROS. To build our workspace we used a convenient tool to build code in our catkin workspace. Some packages needed catkin˙make as they were failing with catkin build and others that fail with catkin˙make needed catkin build to be compiled, but we also ensured that Moveit! accesses the correct ompl libraries. Of course there where some smaller impediments that mostly had to deal with compatibility problems with some libraries and their versions, but in the end were resolved, after a lot

of attention and re-installing some gazebo packages and compile them in the recommended way.

After checking all our ROS enviroment variables, we then were ready to launch our Baxter's simulator in Gazebo and bring it to life. We then also checked our whole ROS environment. In order to get everything running we then simultaneously enabled Gazebo, the robot itself and its tools in it, the joint action server and Moveit!. Thats the only way one can have things running smoothly.

Having successfully set up our whole environment and our workspace we were able to work with some preexisting planners and also test our own new one. We managed to compile all of our planners in ompl with Moveit!.

5.3 Workspace Path Planning and Trajectory Planning

In general the motion planning problem is PSPACE-complete [19], and also there is no guarantee that a solution can be found in a finite time. Workspace path-planning deals with manipulation planning in our workspace. First we planned our robotic manipulator, by specifying the initial and the final position of our end effector in our workspace. Our objective is to compute a dynamically feasible and collision free plan to achieve our goal. [13], [15]

There have been made many approaches [12] on how one can perform a sampling-based motion-planning such as those that we saw in the previous chapters, like the rapidly exploring random treet(RRT) [9] or others like RRT* [21],[22], RRT-connect[8] and the propabilistic roadmap(PRM) framework developed by Kavraki [10]. Other approaches deal with stohastic transitions[5], others with principal component analysis [3] that have to do with motion planning in narrow paths [3] and others rely on propability distributions [7] which extend the propabilistic roadmap (PRM) [10] and result to an undirected graph, called a probabilistic roadmap.

The PRM algorithm 3 can be separated in two stages, the learning one and the query. At the first stage PRM samples the configuration space and builds an undirected graph $G = (V, E)$ which keeps all the information from learning. At the second stage, the algorithm produces a path between q_{init} and q_{goal}, with q_{init} connected to a vertex V_{init} and q_{goal} to another vertex V_{goal} [2],[10].

Algorithm 3: PRM

Data:

Result: Updated graph components, G

$V \leftarrow \emptyset$;

$E \leftarrow \emptyset$;

while *learning-True* **do**

    Generate a random configuration $c \in C_{free}$;

    $V \leftarrow V \cup c$;

    $V_n v \in V \mid Distance(c,v) < M$;

    for $\forall\ v \in V$ *in order of increasing Distance(c, v)* **do**

        if *c and v can be connected and do not lie in the same connected component* **then**

            $E \leftarrow E \cup (c,v)$

$V_s \leftarrow v \in V | Distance(q_{init}, v) < M$;

if $\exists$ *path between* q_{goal} *and a vertex in* V_s **then**

    $v_s \in V_s$ is that vertex;

else

    return failure;

    $V_g \leftarrow v \in V | Distance(q_{goal}, v) < M$;

$V_g \leftarrow v \in V | Distance(q_{goal}, v) < M$;

if $\exists$ *path between* q_{goal} *and a vertex in* V_s **then**

    $v_g \in V_g$ is that vertex;

else

    return failure;

    $V_g \leftarrow v \in V | Distance(q_{goal}, v) < M$;

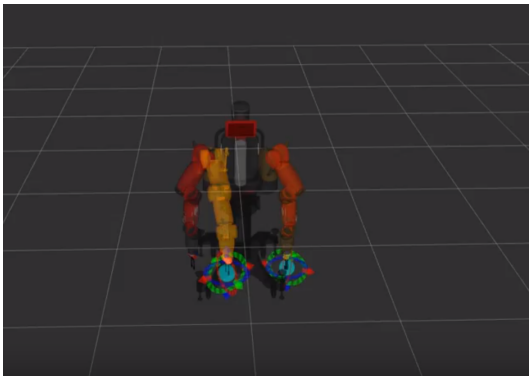
if $\exists$ *path between* v_s *and* v_g **then**

    return solution path $(q_{init}, v_s), P, (v_g, q_{goal})$;

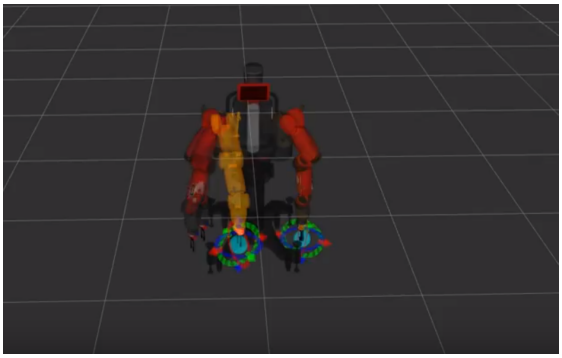
else

    return failure;

Below at Figure 5.1 and Figure 5.2 we show an instance of our experiments while using a preexisting planner in OMPL.



(a) Initial pose and end goal



(b) Planning phase

Figure 5.1: Baxter robot planning with RRT in a free-obstacle world

(a) Initial pose and end goal

(b) After movement completion

Figure 5.2: Baxter robot planning and executing visualized both in Gazebo and Rviz

5.4 Proposed motion planner

We denote our n degrees of freedom configuration space as $C \subset \Re^n$. We then obtain our free configuration space C_{free}. Also the obstacle space C_{obs} is then obtained from collision

checking on single configurations, where:

$$C = C_{free} \cup C_{obs} \text{ and } C_{free} \cap C_{obs} = 0$$

As with the planners above we give initial start and end point, with q_{init} and $x_{goal} \in C_{free}$ and we like to find a path from q_{init} to q_{goal}. We treat our space C as an Euclidean and we assume a parameterization of each of our degrees of freedom as an interval subset of $\Re^n$. Therefore:

$$C = [c_1^{min}, c_1^{max}] \times [c_2^{min}, c_2^{max}] \times ... \times [c_n^{min}, c_n^{max}] \tag{5.1}$$

There have been also other works on reducing the dimensionality in motion planning [16], [12].

Our planner's basic method is that searches for a solution in subspaces of progressively higher dimensions. It searches for a solution in our motion planning problem in the lower dimensional subspaces of C, hoping that in that way a solution will be found faster without having to explicitly expand our search graph in all dimensions. We search for every sub-problem if our system can be reduced. [14]

The planner (algorithm 4), [16] starts its search in the linear one dimensional subspace of C. If our planner fails to find a solution, then it iteratively expands its search subspace by one dimension and follows the same pattern search until eventually finds a path or search the whole C.

Algorithm 4: RRT+

Data: C,q_{init},q_{goal}, distance Δ_x

Result: graph G

```
1  G.init(q_init);
2  C_sub ← 1 − dim subspace of C, through q_init and q_goal;
3  while True do
4      q_rand ← Randconf();
5      q_near ← NearestVertex(q_rand, G);
6      q_new ← NewConf(q_near, q_rand, Δ_x);
7      G.addvertex(q_new);
8      G.addedge(q_near, qnew);
9      if done searching C_sub then
10         if dim(Csub) < dim(C) then
11             if successful search in C_sub then
12                 Expand C_sub by one dim;
13             else
14                 return G
```

The crucial in this planner is to choose wisely the conditions before moving to the next subsearch and to be able to select and reproduce our C_{sub}, which is resolved with us keeping each structure that it is created in the lower dimensions and expand it in the subsequent stages. Of course the question that arises is when we should stop this subspace search [16].

For that, we introduce for our searches a set of timeouts $T_i = t_1, t_2, ..., t_n$ which give us the time spent for each iteration in the subspaces and let t_0 be our base time. If there is a

successive subsearch that means it will be wise to continue searching even more towards this direction. Based upon this simple idea we exponentially increase our search in such subspace, where we had success and stop searching in those that seem unlikely to provide a solution, having in mind the need for sampling in higher dimensions [16] [12]. We assume that each t_i follows a geometric progression and let T be the time of the timeout of our whole algorithm. Let $\lambda > 1$ be our constant factor that gives us the ratio of the runtime between successive subsearches, meaning :

$$t_i = \lambda \cdot t_{i-1} \tag{5.2}$$

and because our t_i's are increased geometrically we get:

$$T = \sum_{i=1}^{n} t_0 \cdot \lambda^i \tag{5.3}$$

From eq. (5.2) and eq. (5.3) we compute:

$$t_0 = \frac{\lambda - 1}{\lambda \cdot (\lambda^n - 1)} \cdot T \tag{5.4}$$

Our main and initial constraint of the contraction of our subspaces C_{sub} is that they must include q_{int} to q_{goal}. We constrain our search in the subspaces by demanding a line passing from q_{init} to q_{goal}. That means having our $C \subset \Re^n$ we can compute C_{sub} with the above requirement. We can then sample within C along the line from q_{init} to q_{goal} simple as:

$$q^i_{rand} = (q^i_{goal} - q^i_{init}) \cdot \kappa + q^i_{init} \tag{5.5}$$

where κ is our random scalar sampling factor, [16]. In the case when a set of constrained degrees of freedom, $G_{con} \subset 1,...,n$ is given, then one can choose a random point x in C along line L from q_{init} to q_{goal} taking into consideration the possible minimum and maximum values of κ that those can achieve and sample as follows:

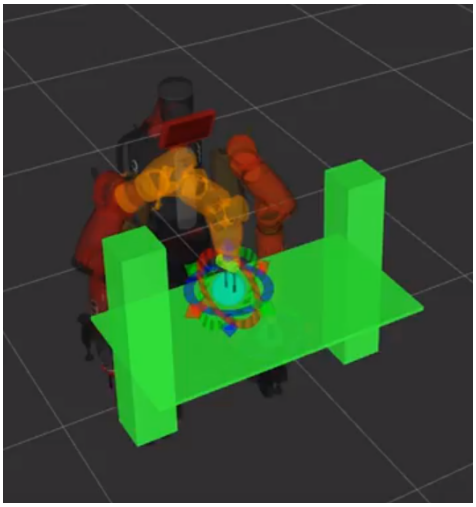
Algorithm 5: RRT+ Samples

Data: q_{init}, q_{goal}, G_{con}, κ_{min}, κ_{max}

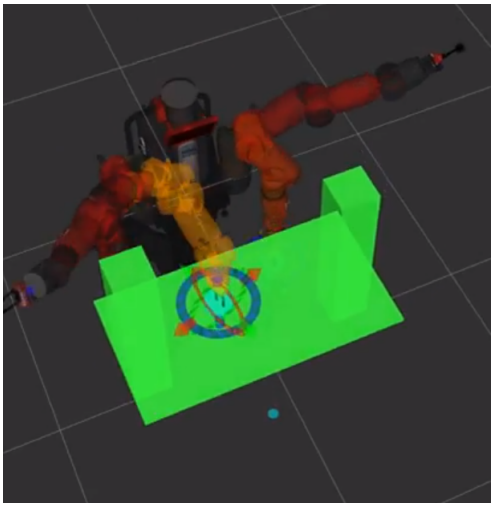
Result: Sample q

```
1 G.init(x_init);
2 q ← x;
3 for i = 1 : n do
4     if i ∉ G_con then
5         q[i] ← Random(0,1) · (c_n^max − c_i^min) + c_i^min;
6     else
7         search for solution without the use of this DoF
8 return q
```

Below at Figure 5.3 and Figure 5.4 we show an instance of our experiments while using our planner after we compiled it in OMPL.

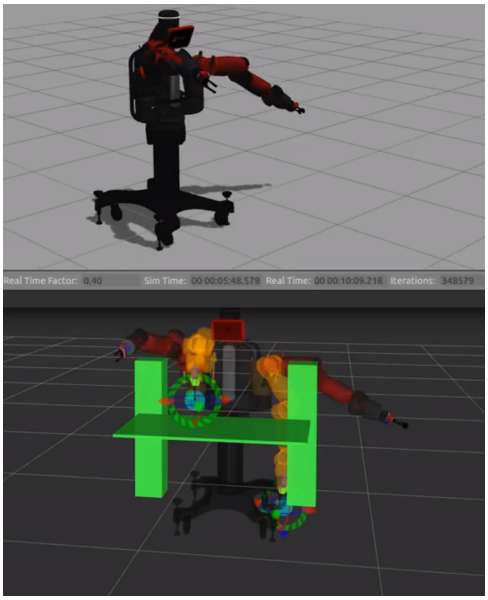


(a) Initial pose and end goal

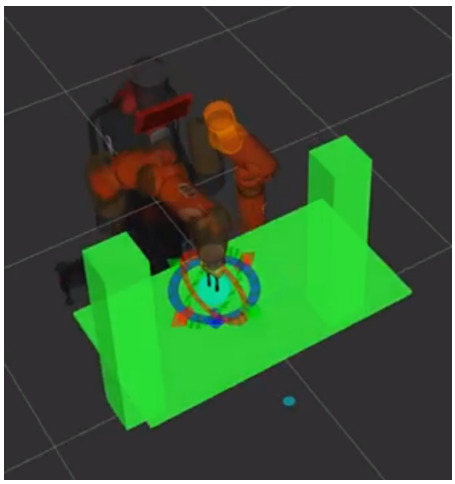


(b) Planning phase

Figure 5.3: Baxter robot planning with RRT+ in a world with two obstacles and a table



(a) Executing, following trajectory path



(b) After movement completion

Figure 5.4: Baxter robot planning and executing RRT+ visualized both in Gazebo and Rviz

Chapter 6: Conclusion and Future Work Directions

6.1 Conclusions

In this thesis, we demonstrated a whole path, building concrete steps one can follow in order to build up the required knowledge to deal with robots and artificial intelligence planning. A kinematic analysis of a 6-degree of freedom robot and simulation based on it was implemented. Also dynamic analysis of a 7-degree of freedom robot was made and planning algorithms were constructed and visualized. Moreover two MDP algorithms were developed and a variety of experiments were conducted about the influence their variables (γ, ε) to the convergence time, the mean reward and the values of their variables. Concerning our work that dealt with planning the Baxter itself, work can be done on improving time planning for high dimensional spaces. One can compute the boundary values, as well as the c_{min} and c_{max} which will give the intersection of the line L between q_{init} and q_{goal}. In general, work can be done on finding a way to sample from projections of arbitrary dimensionality. Last we look forward to implement and run this planner in the real world, with a real Baxter robot, and see how it performs.

Bibliography

[1] Rethink robotics (baxter). `https://www.rethinkrobotics.com/baxter/`, 2018.

[2] S. Carpin. Randomized motion planning, a tutorial. *International Journal of Robotics and Automation*, vol. 21(n. 3):pages 184–196, 2006.

[3] Laumond J.-P. Dalibard, S. Control of probabilistic diffusion in motion planning. In *Algorithmic Foundation of Robotics VIII. STAR*, pages 467–481. Springer, Heidelberg, 2009.

[4] Simeon T.-Corts J. Devaurs, D. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[5] Simeon T.-Corts J. Devaurs, D. Enhancing the transition-based rrt to deal with complex cost spaces. In *IEEE International Conference on Robotics and Automation*, page 41204125.

[6] Herman Høifødt. Dynamic modeling and simulation of robot manipulators the newton-euler formulation, June 2011.

[7] David Hsu, Robert Kindel, Jean-Claude Latombe, and Stephen Rock. Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research*, vol. 21(n. 3):pages 233–255, 2002.

[8] J. J. Kuffner and S. M. LaValle. Rt-connect: An efficient approach to single-query path planning. In *IEEE Int. Conf. on Robotics and Automation, vol. 2*, page 9951001.

[9] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Harvard University, Cambridge, MA, Aiken Computation Lab, Tech. Rep. TR 98-11, 1998.

[10] J.-C. Latomb M.H. Overmars L.E. Kavraki, P. Svestka. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, vol. 12(n. 4):pages 566–580, 1996.

[11] Dana Nau Malik Ghallab and Paolo Traverso. *Automated Planning and Acting*. Cambridge University Press, July 5, 2016.

[12] Ren Mao. *Robots Learning Manipulation Tasks from Demonstrations and Practice*. PhD thesis, 2017. University of Maryland.

[13] Ren Mao, John S. Baras, Yezhou Yang, and Cornelia Fermüller. Co-active learning to adapt humanoid movement for manipulation. In *16th IEEE-RAS International Conference on Humanoid Robots, Humanoids 2016, Cancun, Mexico, November 15-17, 2016*, pages 372–378, 2016.

[14] Ren Mao, John S. Baras, Yezhou Yang, and Cornelia Fermüller. Co-active learning to adapt humanoid movement for manipulation. *CoRR*, abs/1609.03628, 2016.

[15] Ren Mao, Yezhou Yang, Cornelia Fermüller, Yiannis Aloimonos, and John S. Baras. Learning hand movements from markerless demonstrations for humanoid tasks. In *14th IEEE-RAS International Conference on Humanoid Robots, Humanoids 2014, Madrid, Spain, November 18-20, 2014*, pages 938–943, 2014.

[16] Ioannis Rekleitis Jason M. O'Kane Marios P. Xanthidis, Joel M. Esposito. Analysis of motion planning by sampling in subspaces of progressively increasing dimension. University of South Carolina, 2018.

[17] Matlab optimization toolbox. `http://www.mathworks.com/products/matlab/`, 2016. The MathWorks, Natick, MA, USA.

[18] S. Hutchinson M.W. Spong and M. Vidyasagar. *Robot modeling and control*. Wiley New Jersey, 2006.

[19] J. H. Reif. Complexity of the generalized movers problem. Harvard University, Cambridge, MA, Aiken Computation Lab, Tech. Rep., 1985.

[20] L. Sciavicco and B. Siciliano. *Modelling and control of robot manipulators*. Springer Verlag, 2000.

[21] Emilio Frazzoli Sertac Karaman. Incremental sampling-based algorithms for optimal motion planning. arXiv:1005.0416, 2010.

[22] Emilio Frazzoli Sertac Karaman. Sampling-based algorithms for optimal motion planning. arXiv:1105.1186, 2011.

[23] Anuj Shah. Cooperative sequential composition control for compliant manipulation an approach via robot contact language, August 20, 2015.

[24] Yang C. Li C. Ma H. Zhao L. Smith, A. Development of a dynamics model for the baxter robot. In *IEEE International Conference on Mechatronics and Automation (ICMA)*, pages 1244–1249.

[25] Ioan A. Sucan and Sachin Chitta. Moveit! http://moveit.ros.org, 2017.

[26] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. http://ompl.kavrakilab.org.

[27] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

[28] Spyros Tzafestas. *Introduction to Mobile Robot Control*. Elsevier, 1st October 2013.

[29] Ma Hongbin Fu Mengyin Yang, Chenguang. Robot kinematics and dynamics modeling. In *Advanced Technologies in Modern Robotic Applications*. Springer Singapore, 2016.