Single-Query Robot Motion Planning using Rapidly Exploring Random Trees (RRTs)

by

Jonathan Bagot

A thesis submitted to
The Faculty of Graduate Studies of
The University of Manitoba
in partial fulfillment of the requirements
of the degree of

Master of Science

Department of Computer Science
The University of Manitoba
Winnipeg, Manitoba, Canada
August 2014

Thesis advisor
Jacky Baltes

Author
Jonathan Bagot

Single-Query Robot Motion Planning using Rapidly Exploring Random Trees (RRTs)

Abstract

Robots moving about in complex environments must be capable of determining and performing difficult motion sequences to accomplish tasks. As the tasks become more complicated, robots with greater dexterity are required. An increase in the number of degrees of freedom and a desire for autonomy in uncertain environments with real-time requirements leaves much room for improvement in the current popular robot motion planning algorithms. In this thesis, state of the art robot motion planning techniques are surveyed. A solution to the general movers problem in the context of motion planning for robots is presented. The proposed robot motion planner solves the general movers problem using a sample-based tree planner combined with an incremental simulator. The robot motion planner is demonstrated both in simulation and the real world. Experiments are conducted and the results analyzed. Based on the results, methods for tuning the robot motion planner to improve the performance are proposed.

Contents

Abstract ii
Table of Contents vi
List of Figures vii
List of Tables x
List of Algorithms xii
Acknowledgments xiii
Dedication xiv

1 Introduction **1**
1.1 How hard is the robot motion planning problem? 3
1.2 Why is it important to solve the robot motion planning problem? . . 4
1.3 How can the robot motion planning problem be solved? 5
1.4 Summary of Contributions 8
1.5 Thesis Organization 9
1.6 Terminology 10

2 Background **12**
2.1 Configuration Space (CSPACE) 13
2.2 Task Space (TSPACE) 14
2.3 Cartesian Coordinates 15
2.3.1 Left Hand Rule 16
2.3.2 Right Hand Rule 17
2.4 Homogeneous Coordinates 17
2.5 Representing Orientation by Angles 18
2.5.1 Euler 18
2.5.2 Quaternions 21
2.6 Transformation Matrices 22

2.6.1 Notation . . . 22
2.6.2 Rotation Matrices . . . 23
2.6.2.1 Combining Rotation Matrices . . . 24
2.6.2.2 Extracting Euler Angles from Rotation Matrix . . . 26
2.6.3 Translation Matrices . . . 27
2.7 Frames . . . 28
2.8 Kinematics . . . 30
2.8.1 Denavit Hartenberg (D-H) Parameters . . . 32
2.8.2 Forward Kinematics (FK) . . . 35
2.8.3 Inverse Kinematics (IK) . . . 35
2.8.3.1 Jacobian Matrix . . . 37
2.8.3.2 Calculating the Jacobian Matrix . . . 38
2.8.3.3 Jacobian Transpose . . . 39
2.8.3.4 Jacobian Pseudo Inverse . . . 40
2.9 Polygon of Support (POS) and Zero Moment Point (ZMP)/Center of Pressure (COP) . . . 42
2.10 Inverted Pendulum . . . 46
2.11 Proportional-Integral-Derivative (PID) . . . 46
2.12 Nearest Neighbor (NN) . . . 48
2.13 Voronoi Diagram . . . 49
2.14 Graph Theory . . . 50
2.14.1 Shortest Path Problem . . . 52
2.14.1.1 Dijkstra's Algorithm . . . 52
2.14.1.2 A* Algorithm . . . 53
2.14.2 Trees . . . 53
2.15 Incremental Simulator . . . 55
2.16 Collision Detection . . . 55
2.17 Summary . . . 60

3 Related Work **61**
3.1 Randomized Potential Fields . . . 61
3.2 Probabilistic Road Map (PRM) . . . 63
3.3 Ariadne's Clew Algorithm . . . 64
3.4 Flexible Binary Space Partitioning (BSP) . . . 66
3.5 Summary . . . 67

4 Rapidly Exploring Random Tree (RRT) 68
4.1 Sample Bias . . . 74
4.2 Nearest Neighbor (NN) . . . 76
4.3 Distance Metric . . . 77
4.4 Collision Detection . . . 81
4.5 Related Work . . . 81
4.5.1 RRT-Goal Bias . . . 81
4.5.2 RRT-Goal Zoom . . . 82
4.5.3 RRT-EXTEND . . . 82
4.5.4 RRT-CONNECT . . . 86
4.5.5 RRT-Bi-directional . . . 86
4.5.6 Jacobian Transpose RRT (JT-RRT) . . . 87
4.5.7 Multipartite RRT (MP-RRT) . . . 88
4.5.8 RRT-Blossom . . . 88
4.5.9 RRT* . . . 89
4.5.10 Obstacle-Based RRT (OB-RRT) . . . 90
4.5.11 Task Space RRT (TSPACE-RRT) . . . 92
4.5.12 Closed-Loop RRT (CL-RRT) . . . 92
4.5.13 Particle RRT (pRRT) . . . 93
4.5.14 Heuristically-guided RRT (hRRT) . . . 93
4.5.15 Exploring/Exploiting Tree (EET) . . . 94
4.6 Other Applications . . . 95
4.7 Summary . . . 95

5 Implementation 96
5.1 Real World . . . 97
5.1.1 Humanoid Robot . . . 97
5.1.2 High Level Logic . . . 107
5.1.3 Vision . . . 109
5.1.4 Localization and Mapping . . . 110
5.1.5 Trajectory Planning . . . 111
5.1.6 Balancing . . . 112
5.1.7 Firmware . . . 119
5.2 Simulation . . . 121
5.2.1 Computer . . . 122
5.2.2 Motion Simulator . . . 122
5.2.3 Third Party Software . . . 124
5.2.3.1 Open Dynamics Engine (ODE) . . . 124

5.2.3.2 Boost . . . 127
5.2.3.3 Approximate Nearest Neighbor (ANN) . . . 128
5.2.3.4 Graphviz . . . 129
5.2.3.5 Voro++ . . . 129
5.2.3.6 Open Graphics Library (OpenGL) . . . 129
5.2.4 Robot Models . . . 130
5.2.4.1 Sphere Robot . . . 132
5.2.4.2 Humanoid Robot . . . 135
5.2.4.3 Motor Adapter . . . 139
5.2.4.4 Motor Controller . . . 142
5.2.5 World Models . . . 142
5.3 Motion Planner with RRT and Incremental Simulator . . . 153
5.3.1 Random Number Generation . . . 157
5.3.2 Random Configuration Generation . . . 164
5.4 Summary . . . 165

6 Evaluation **167**
6.1 Overview . . . 167
6.2 Experiment Purpose . . . 167
6.3 Experiment Setup . . . 168
6.4 Experiment Criteria . . . 170
6.5 Results, Analysis and Observations . . . 174
6.6 Modifications and Optimizations . . . 224
6.7 Summary . . . 230

7 Conclusion **231**
7.1 Future Work . . . 233

A Acronyms **236**

B Glossary **243**

C Scilab Functions **244**

Bibliography **259**

List of Figures

2.1 2D Cartesian Coordinates . . . 15
2.2 Left Hand Rule . . . 16
2.3 Right Hand Rule . . . 17
2.4 Euler Angles - R_x, R_y, R_z . . . 20
2.5 4 link kinematic model of humanoid robot. . . . 32
2.6 4 link kinematic model of humanoid robot with coordinate frames attached. . . . 35
2.7 First half of gait cycle . . . 44
2.8 Double support POS . . . 45
2.9 Single support POS . . . 46
2.10 Voronoi Diagram in 3D . . . 50
2.11 Example Directed Weighted Graph . . . 52
2.12 Tree . . . 54
2.13 Bounding Spheres Example in 2D . . . 57
2.14 Bounding Spheres . . . 58
2.15 Bounding Multi-Spheres . . . 58
2.16 Bounding Boxes . . . 59
2.17 Bounding Polygons . . . 59

4.1 Rapidly Exploring Random Tree (RRT) . . . 69

5.1 Humanoid Robot - Blitz . . . 98
5.2 Cardinal Planes . . . 100
5.3 Blitz Joints with Servo ID . . . 101
5.4 AX-12 valid angle range (from Dynamixel AX-12 Manual Robotis [2006]) . . . 104
5.5 Nokia N5500 . . . 105

5.6 Motion Editor . . . 106
5.7 Blitz Functional Block Diagram (FBD) . . . 107
5.8 State Machine Diagram for Soccer Player Penalty Kick . . . 109
5.9 Sinusoid . . . 112
5.10 Accelerometer Data for each Cardinal Plane with ESP . . . 116
5.11 Accelerometer Data for each Cardinal Plane with DESP . . . 117
5.12 Accelerometer Data for each Cardinal Plane with Rate of Change . . . 118
5.13 Sphere Model . . . 133
5.14 Blitz Robot Model vs. Blitz . . . 136
5.15 Humanoid Model . . . 137
5.16 Motor Control . . . 141
5.17 World 0 . . . 143
5.18 World 1 . . . 144
5.19 World 1-2 . . . 145
5.20 World 2 . . . 146
5.21 World 3 . . . 147
5.22 World 4 . . . 147
5.23 Real World 5 . . . 149
5.24 Model World 5 . . . 150
5.25 World 5 Tool Trajectory Collision . . . 151
5.26 World 5 Tool Trajectory . . . 152
5.27 Pseudo Random Number Generator (RNG) Incremental Sample Range, Single Radius . . . 159
5.28 Pseudo RNG Incremental Sample Range, Single Radius Increment . . . 160
5.29 Pseudo RNG Incremental Sample Range, Dual Radius . . . 161
5.30 Pseudo RNG in Sphere . . . 163

6.1 Colour Coding . . . 177
6.2 Best CONNECT Result for World 0 - Goal Bias 40 . . . 178
6.3 Best EXTEND Result for World 0 - Goal Bias 40 . . . 179
6.4 Bias Probability vs. Number of Configurations - World 0 . . . 180
6.5 Bias Probability vs. STD Number of Configurations - World 0 . . . 181
6.6 Bias Probability vs. Total Execution Time - World 0 . . . 182
6.7 Best CONNECT Result for World 1 - Goal Bias 1 . . . 189
6.8 Best EXTEND Result for World 1 - Goal Bias 50 . . . 190
6.9 Bias Probability vs. Number of Configurations - World 1 . . . 191
6.10 Bias Probability vs. STD Number of Configurations - World 1 . . . 192
6.11 Bias Probability vs. Total Execution Time - World 1 . . . 193

6.12 Best CONNECT Result for World 2 - Goal Bias 1 197
6.13 Best EXTEND Result for World 2 - Goal Bias 40 198
6.14 Bias Probability vs. Number of Configurations - World 2 199
6.15 Bias Probability vs. STD Number of Configurations - World 2 200
6.16 Bias Probability vs. Total Execution Time - World 2 201
6.17 Best CONNECT Result for World 3 - Goal Bias 1 205
6.18 Best EXTEND Result for World 3 - Goal Bias 10 206
6.19 Bias Probability vs. Number of Configurations - World 3 207
6.20 Bias Probability vs. STD Number of Configurations - World 3 208
6.21 Bias Probability vs. Total Execution Time - World 3 209
6.22 Best CONNECT Result for World 4 - Goal Bias 60 213
6.23 Best EXTEND Result for World 4 - Goal Bias 20 214
6.24 Bias Probability vs. Number of Configurations - World 4 215
6.25 Bias Probability vs. STD Number of Configurations - World 4 216
6.26 Bias Probability vs. Total Execution Time - World 4 217
6.27 Bias Probability vs. Number of Configurations - World 5 220
6.28 Bias Probability vs. Total Execution Time - World 5 221
6.29 Best Result for World 5 222
6.30 2D RRT k-Nearest Example 228
6.31 2D RRT k-Nearest Example 229

List of Tables

5.1 Blitz Joints 102
5.1 Blitz Joints 103

6.1 Summary of Evaluation Criteria 173
6.2 Qualitative Inspection Ranking 174
6.3 Result Matrix World 0 175
6.3 Result Matrix World 0 176
6.4 Result Matrix World 1 187
6.4 Result Matrix World 1 188
6.5 Result Matrix World 2 195
6.5 Result Matrix World 2 196
6.6 Result Matrix World 3 203
6.6 Result Matrix World 3 204
6.7 Result Matrix World 4 211
6.7 Result Matrix World 4 212
6.8 Result Matrix World 5 219

List of Algorithms

1 Jacobian 40

2 Using Jacobian Transpose with Incremental Simulator 41

3 PID 47

4 PID with Integral Limit 48

5 Classic RRT [Lavalle, 1998] 73

6 3D Euclidean Fast Approximation [Ritter, 1990] 79

7 2D Euclidean Fast Approximation [Ritter, 1990] 80

8 RRT-EXTEND 8 [Kuffner Jr. and Lavalle, 2000; Lavalle and Kuffner Jr., 2000] 83

9 EXTEND 8 [Kuffner Jr. and Lavalle, 2000; Lavalle and Kuffner Jr., 2000] 84

10 RRT-CONNECT 8 [Kuffner Jr. and Lavalle, 2000; Lavalle and Kuffner Jr., 2000] 85

11 CONNECT 8 [Kuffner Jr. and Lavalle, 2000; Lavalle and Kuffner Jr., 2000] 85

12 Pseudo Random Number Generator 160

13 Pseudo Random Number Generator in Sphere 162

14 Random Configuration 164

15 Random Sphere Configuration 165

16 Random Humanoid Configuration 165

17 k-Nearest Move 227

Acknowledgments

I would like to thank my advisor and examining committee member Dr. Jacky Baltes for your friendship, support, and advice over the years from undergraduate to present time. I would also like to thank my advisory and examining committee member Dr. John Anderson for your feedback, insight, and attention to detail which has proven to be invaluable. Another thank you to my advisory committee member Dr. James Young. Finally a special thanks goes out to my parents and family for encouraging me to further my education.

This thesis is dedicated to my parents.

None of my accomplishments could have been achieved without you.

Chapter 1

Introduction

Motion planning is done by humans every day and typically without much thought. Imagine a simple task you perform every morning such as making a cup of coffee. Making a cup of coffee requires many motion plans. For example consider the first task to get a cup from the cupboard. You must first walk over to the cupboard and position your body such that the distance from your shoulder to the cupboard handle is less than your arm length. The required sequence of motions for your body to achieve this position from your initial position can be called a motion plan. A good motion plan would be quick, require little effort, and avoid any part of your body bumping into obstacles in the kitchen. For each subsequent movement a motion plan is required. In order to complete the task you would determine a motion plan to move your hand to grasp the cupboard handle, move your hand to open the cupboard door, move your hand to grasp the cup, and so forth. You may even decide

to use both hands; one to open and close the cupboard door and the other to grasp the cup. For humans this can be called a simple task, but in reality there are many complex systems at work. The human body has a significant number of possible joint movements (Degrees of Freedom (DOF)), and many ways that the same end goal can be achieved through different combinations of joint movements. Determining motion plans is a extremely difficult problem for an autonomous robot despite the use of modern technology because current hardware does not provide equivalent sensory and movement capabilities to a human. There is still much improvement to be made in both the hardware and software to develop and execute motion plans as effectively as humans.

In robotics, motion planning refers to the organization of robot motions in a specific sequence to achieve a goal. A robot motion in the simplest sense can be thought of as a translation or rotation. The actual execution of a robot motion is much more complicated because the *dynamics* and *differential constraints* of the robot must be considered. When considering dynamics in robot motion planning, the effect of external forces on the robot motion play a role in the motion plans chosen. On the other hand, when considering differential constraints in robot motion planning, the velocities of the robot at any given moment play a role in the motion plans chosen. Motion planners typically ignore the dynamics and differential constraints of the robot assuming that a secondary planner called a trajectory planner can solve for each transition between motion steps during the execution of the motion plan [LaValle,

2006].

Robot motion planning is a cross-disciplinary problem because it involves both hardware and software components. The Mechanical Engineering, Electrical Engineering, and Computer Science fields all must contribute in order to make advancements in solving the robot motion planning. As new sensor and servo technologies emerge, software algorithms must adapt to make use of different feedback.

1.1 How hard is the robot motion planning problem?

The complexity class that the robot motion planning problem falls into is as follows:

The motion planning problem for robots is a form of the *general mover's problem* an extension of the *classical mover's problem* also known as the *piano mover's problem*. The piano mover's problem is to determine how to move a piano from one point to another without collision. The general mover's problem replaces the piano with a generic object where the moving "object may have multiple polyhedra freely linked together at various distinguished vertices" [Reif, 1979]. A robot would have many masses linked together by joints. A DOF for a robot is a controllable joint that defines the configuration of the robot. The number of DOF defines the dimension of the moving object. The general movers problem was shown to be Polynomial

Space (PSPACE)-hard by Reif [1979]. A problem that is PSPACE-hard can be solved with a Turing machine with a polynomial amount of memory and unlimited time. As the number of DOF increases the runtime follows for deterministic solutions.

In addition to solving the general mover's problem, the concept of Kinodynamic Motion Planning introduced by Donald et al. [1993] considers both kinematic and dynamic constraints as well. Kinodynamic motion planning has been shown to be Non-deterministic Polynomial-time (NP)-hard in three dimensions. A problem that is NP-hard is at least as hard as any NP-problem, it may be even harder.

1.2 Why is it important to solve the robot motion planning problem?

The increasing complexity of robots has brought on a whole new set of challenges for robotics research. The cost of building small scale robots has decreased considerably, which facilitates active research in robotics. An increase in the number of DOF and a desire for autonomy in uncertain environments with real-time requirements leaves much room for improvement in the current popular motion planning algorithms.

Efficient robot motion planning is one significant piece of the puzzle in making robots useful and allowing them to operate in the same environment as humans. Having robots that can operate in the same environment as humans along side hu-

mans is the ideal situation since it eliminates the need for specialized environments that allow the robots to work. These specialized environments can be expensive and require large amounts of space to isolate them from humans to prevent injury.

The motion planner must use information from other complex systems such as vision, localization, mapping systems but it is just as important as any other part of the puzzle. All of these complex systems combined together make robots useful.

1.3 How can the robot motion planning problem be solved?

Since the motion planning problem for robots is PSPACE-hard [Reif, 1979], finding paths quickly for robots with large DOF in uncertain environments requires sacrificing planner optimality for query response time [Kavraki et al., 1995; Kuffner et al., 2002]. A path that is good enough but not optimal allows for fast think and react cycle time. Reducing the think and react cycle time is a critical task for robots that interact with the real world. Deterministic complete planners are well defined but are known to be intractable for robots with large DOF and real-time requirements in even simple domains. Sample-based probabilistically complete planners (will find solution if one exists but cannot determine if one does not exist) have showed promise returning sub-optimal results in reasonable time. Sample-based planners can be classified in two general groups: roadmap and tree [Tsianos et al.,

2007]. The main difference between roadmap and tree planners is the underlying data structure. A roadmap planner typically stores the map in a graph and requires two phases to return an answer [Hsu et al., 2007]. The first learning phase builds the map and the second query phase determines the plan with a graph search algorithm such as Dijkstra's algorithm which is described in Section 2.14.1.1. The learning phase is an expensive operation but in a static environment the roadmap can be used for all queries, therefore the cost of the learning phase is only incurred once. In a dynamic environment the cost of the learning phase would be too large to react to changes in the environment since the roadmap must be rebuilt before each query [Jaillet and Simeon, 2004]. Roadmap planners are discussed in more detail in Chapter 3. A tree planner stores only the necessary data to find a solution to the query in a tree. The root of the tree represents the start state and the tree is built incrementally until the end state is reached. The answer to the query is simply a branch of the tree from the root (start) to a leaf (end) node, therefore the answer is found in a single query [Sucan and Kavraki, 2012]. Building the tree is not an expensive operation which makes tree planners ideal for dynamic environments where plans may become invalid quickly. Tree planners are discussed in more detail in Chapter 4. The focus of the research to be conducted is on sample-based tree planners because the goal is to solve the motion planning problem in dynamic environments with real-time requirements.

A RRT is a data structure for planning problems where the nodes of the tree

are generated randomly and typically biased towards un-explored areas [Lavalle, 1998]. From the name, it is clear that the RRT can be categorized as a sample-based tree. There are many elements of a RRT that can be implemented in various ways. Different implementations behave and perform better or worse for specific problems. The RRT is the basis of the motion planner presented in this thesis. Instead of using a single implementation of a RRT many combinations of different element implementations are explored in order to analyze how each behaves and performs for specific types of problems. The key elements of a RRT are sample bias, nearest neighbor, distance metric, and collision detection. These key elements are discussed in detail in Chapter 4. Each element effects the performance of the RRT in different ways. For example the sample bias method has a direct impact on how well the tree covers the environment while the nearest neighbor method only limits how fast random configurations can be connected to the tree.

Motion planning algorithms determine continuous collision free paths from an initial position to goal position. The motion plan is constrained by the physical characteristics of the robot and the environment. Robots with many DOF, redundant manipulators, and real-time requirements increase the difficulty of the motion planning problem. The goal of my thesis is to develop a planner able to consider kinodynamic constraints, find or approximate the IKs solution, and perform collision detection quickly and reliably in order to generate feasible motions plans.

Simple robots such as a wheeled robots on an even terrain are dynamically stable

in every possible configuration. Complex robots such as humanoid robots are not dynamically stable in every possible configuration, and therefore they can fall over. This is, of course, undesirable. A solution strategy is presented to determine for complex robots if a configuration will be dynamically stable.

1.4 Summary of Contributions

The thesis presented makes many contributions to the motion planning problem for robots. These include current techniques to solving the robot motion planning problem, using a sample-based tree algorithm combined with an incremental simulator, and an extensive analysis of the different elements that constitute a sample-based tree algorithm; namely the Rapidly Exploring Random Tree (RRT). My motion planner was also tested in both simulation and the real world, which demonstrates that the simulated robot and world models can be translated and applied to the real world.

By combining a sample-based tree algorithm with an incremental simulator, the motion planner not only generates a plan but tests the feasibility of each step in real-time as it is generated using a model of the robot and world. The plan can then be executed on a real robot with greater confidence that the plan is valid.

Many other solutions to the motion planning problem for robots have been proposed but have only been demonstrated in simulation. For example variations of sample-based tree planners were proposed in Lavalle [1998]; Lavalle and Kuffner Jr.

[2000]; Kuffner Jr. and Lavalle [2000]; Lindemann and LaValle [2004]; Zucker et al. [2007] but were only evaluated in simulation with models that are always dynamically stable (cannot fall over) and fictitious environments. My thesis research is demonstrated in both simulation and the real world. Not all solutions translate well from simulation to the real world, therefore demonstrating applicability to the real world is also a significant contribution.

1.5 Thesis Organization

The thesis is organized and presented in the following chapters:

Background

The purpose of the Background chapter is to provide sufficient information to the reader on robotics and motion planning since the problem at hand requires a breadth of knowledge. A wide array of concepts are presented with examples. It is not necessary to immediately read this chapter. If you are familiar with robotics and motion planning, you could skip to the Related Work chapter. If you are unfamiliar with robotics and motion planning, then it is a good place to start. The Background chapter is back referenced in later chapters when the concepts are related and should be understood. If the reader wants more information on the topic at hand, they can always come back to this chapter.

Related Work

A survey of the current popular robot motion planning techniques are presented

to provide a broad picture of the evolution of solution strategies and how they are applicable to specific problems.

Rapidly Exploring Random Tree (RRT)

The concept of the RRT is presented and related work to date is discussed.

Implementation

The specific details of the implementation of my solution strategy are given. In particular the implementation for a real humanoid robot and its equivalent simulated model, other simulated robot models, simulated and real environments, and the motion planner.

Evaluation

My approach is evaluated through experimentation. This chapter describes the experiment purpose, setup, and criteria. The results of the experiment are presented with analysis and observations. Finally, modifications and optimizations to my approach are suggested.

Conclusion

Final thoughts, future work and discussion.

1.6 Terminology

The terminology used in this thesis is described as required and background information is given in a orderly manner throughout. All acronyms used are defined in Appendix A. All terms that may require additional explanation are defined in

Appendix B. All concepts used are defined in Chapter 2.

Chapter 2

Background

In order to solve the motion planning problem for robots it is necessary to be familiar with a wide array of topics in robotics. Before proceeding to the following chapters it is recommended that the reader understand the topics in this chapter. This chapter aims to provide sufficient background information to readers with little to no robotics knowledge.

The chapter is organized as follows: first the state space representation that is commonly used in robot motion planning is introduced. Next coordinate systems, angle representations, and transformations are described in order to aid the reader in understanding how robotics kinematics problems are solved. In contrast to the kinematics, dynamics are introduced with the concept of the inverted pendulum and Polygon of Support (POS) in relation to balancing. The remainder of this chapter covers various topics such as Proportional-Integral-Derivative (PID), Near-

est Neighbor (NN), Voronoi diagrams, graph theory, incremental simulations, and collision detection are covered because they are necessary to understand the solution strategy I propose in the remainder of this thesis.

2.1 Configuration Space (CSPACE)

An essential ingredient to solving planning problems is a unique understanding of the state space. The state space consists of every possible combination of inputs and the respective outputs. For a large number of inputs the size of the state space yields a combinatorial explosion of states [LaValle, 2006]. For example, a Selective Compliance Articulated Robot Arm (SCARA) has a fixed base and 4 DOF, it can reach any 3D coordinate in its workspace. SCARA are a commonly for assembly robots. The CSPACE for this type of manipulator would consist of all possible arm configurations and the respective 3D coordinates of the tool. Imagine each of the 4 joints has a range of [0.0,180.0] in 0.1 degree increments. This yields a state space of 1800^4 for a relatively simple robot. Finding the optimal solution would require searching through the state space for a set of states which arrive at the goal state from the initial state and maximize the heuristic function. For example if a robot has a limited power supply, the heuristic function might rank solutions that use less power higher. In general searching through the entire state space is infeasible for large state spaces especially if a solution must be found in real-time. A solution might be considered acceptable if the power consumption does not exceed a fixed

threshold although it may not be the optimal solution.

The most commonly used state space representation for robot motion planning is known as a Configuration Space (CSPACE) introduced by Lozano-Perez [1980]. A robot in CSPACE is reduced to a single n-dimensional point (configuration) moving through space, where n is the number of DOF. The CSPACE represents the set of all transformations that can be applied to the robot. The Configuration Free (CFREE) set denotes the set of all robot configurations that are not in collision with obstacles. The Configuration Obstacle (COBS) set is the compliment of the CFREE set, therefore the COBS set denotes the set of all robot configurations that are in collision with obstacles. A valid path from a robot's initial configuration to goal configuration would consist of configurations which all belong to the CFREE set.

2.2 Task Space (TSPACE)

The Task Space (TSPACE) set is a subset of the CSPACE [Liegeois, 1977]. Instead of considering all possible robot configurations, some joints are fixed using some information about the desired goal configuration. For example if the goal configuration requires the tool of a robotic manipulator to be in a specific orientation, a few joints can be fixed to provide the desired orientation then the TSPACE set consists of all possible combinations of the remaining joints. The TSPACE set is typically used in feedback control systems and is usually called Task Space Control in related literature [Shkolnik and Tedrake, 2009].

2.3 Cartesian Coordinates

Cartesian coordinates are a method of specifying points in space. Each point is given numerical coordinates that specify the location of the point on a set of planes that are perpendicular to one another [Wright et al., 2007]. Where the planes intersect is called the origin. Most people are familiar with 2D coordinates. In robotics coordinates are specified in 3D.

A 2D coordinate system has two perpendicular planes with axes typically labelled x, y as shown in Figure 2.1. A 2D coordinate has the numerical form (x, y).

A 3D coordinate system has three perpendicular planes with axes typically labelled x, y, z. A 3D coordinate has the numerical form (x, y, z).

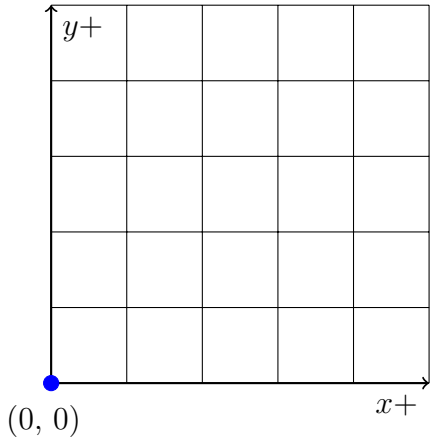


Figure 2.1: 2D Cartesian Coordinates

There are a few useful rules for two common 3D coordinate systems. There is no standard for 3D coordinate systems in robotics. The coordinate system chosen

is a matter of opinion and there is no advantage to any except for commonality. The following rules use your thumb, index finger, and middle finger to remember the relationship between axes and their direction [Corral, 2008]. The thumb is positive X, the index finger is positive Y, and the middle finger is positive Z. When pointing these fingers, one can also visually understand the affect of rotations on the coordinate system by turning their hand by the appropriate rotation.

2.3.1 Left Hand Rule

To use the Left Hand Rule one would, using their left hand, position their thumb, index finger, and middle finger to point in the direction as shown in Figure 2.2 [Corral, 2008]. The default coordinate system of DirectX uses the left hand rule, but can be configured to use other coordinate systems if desired.

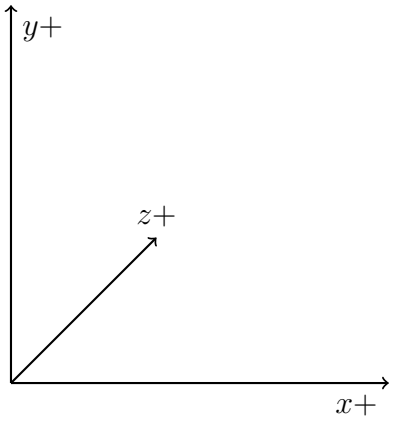


Figure 2.2: Left Hand Rule

2.3.2 Right Hand Rule

To use the Right Hand Rule one would using their right hand position their thumb, index finger, and middle finger to point in the direction as shown in Figure 2.3 [Corral, 2008]. The default coordinate system of OpenGL and XNA uses the right hand rule, but can be configured to use other coordinate systems if desired. The coordinate system used in this thesis uses the right hand rule.

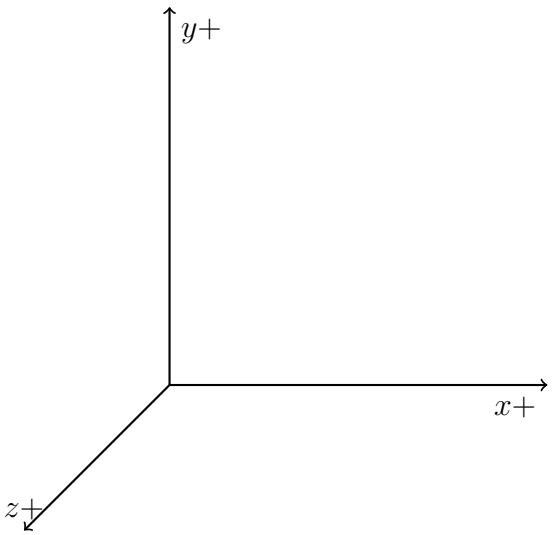


Figure 2.3: Right Hand Rule

2.4 Homogeneous Coordinates

Homogeneous coordinates is another method of specifying points in space. When using Cartesian coordinates, each point has one unique definition, while using homo-

geneous coordinates, each point has infinite definitions. Recall that 3D coordinates have the numerical form (x, y, z). Given a real number not equal to zero w, homogeneous coordinates have the numerical form (xw, yw, zw, w). For example if $w = 1$ then Cartesian coordinates (x, y, z) is $(x, y, z, 1)$ in homogeneous coordinates.

Using homogeneous coordinates can be thought of as defining 3D coordinates as three axes and the origin. The uses of homogeneous coordinates becomes apparent in later sections when defining reference frames and using matrix multiplication to perform translation and rotation.

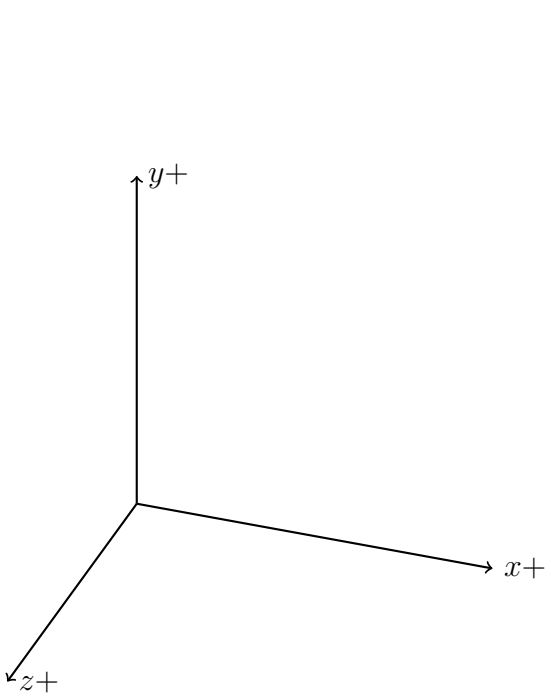
2.5 Representing Orientation by Angles

In robotics applications it is often necessary to represent orientation by angles. For example specifying joint angles, frame orientations, and the end effector or tool orientation. The following section describes conventions for defining angles of 3D objects or reference frames.

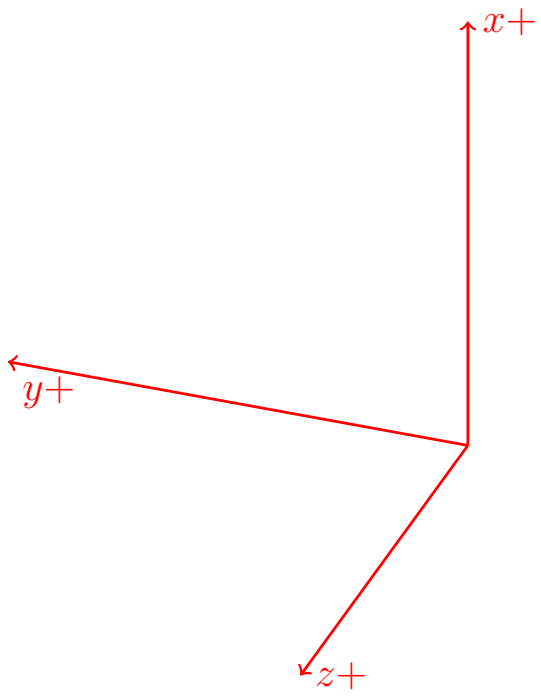
2.5.1 Euler

Euler angles are the most common method of representing orientation for industrial robots [Ang and Tourassis, 1987]. They also have many other applications such as specifying orientation of aircraft, gyroscopes, and computer graphics to name a few. Any orientation can be defined by combining three rotations in a specific order. The three rotations can be thought of as the rotation about each axis X, Y, Z

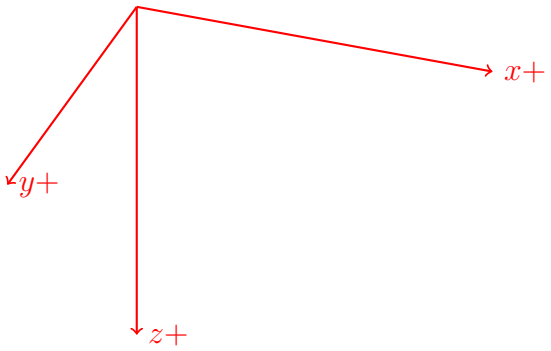
typically denoted R_x, R_y, R_z. They are also sometimes referred to as roll, pitch, and yaw due to their application to aircraft. The order of the rotations matter: applying the rotations roll-pitch-yaw and yaw-pitch-roll will produce different results. This can be visualized using the Right Hand Rule: apply the same rotations in different order and watch what it does to your coordinate system. There is no standard for the order in which to apply the rotations in robotics. The order chosen is a matter of opinion and there is no advantage to any except for commonality [Schneider and Eberly, 2002; Eberly, 2014]. An example of Euler rotations is shown in Figure 2.4. In the example, the figure depicts how a 90° R_x, R_y, and R_z rotation affects the Cartesian coordinate system.



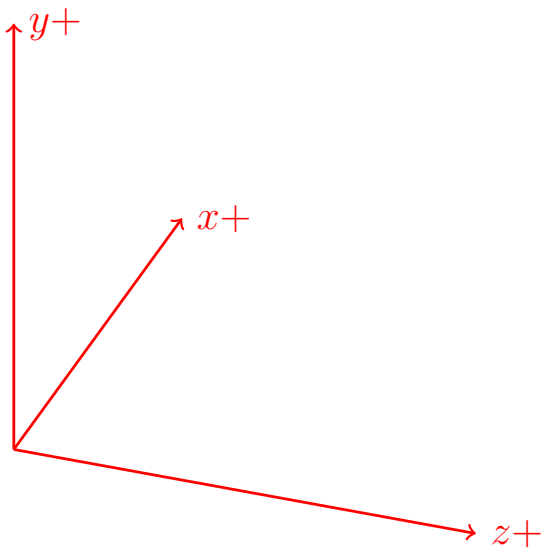
(a) $R_x = 0°, R_y = 0°, R_z = 0°$



(b) $R_x = 90°, R_y = 0°, R_z = 0°$



(c) $R_x = 0°, R_y = 90°, R_z = 0°$



(d) $R_x = 0°, R_y = 0°, R_z = 90°$

Figure 2.4: Euler Angles - R_x, R_y, R_z

One of the main advantages of using Euler angles is that they are simple to comprehend. The main disadvantage is the potential for ambiguity, since order matters, and the possibility of *gimbal lock* sometimes referred to as a *singularity*. A gimbal is a pivoted support that allows rotation about a single axis [Grassia, 1998]. Imagine that R_x, R_y, and R_z each correspond to a gimbal, if two of the gimbals are parallel then a DOF is lost. Similarly one case where this is apparent is when $R_y = 0$ since $cos(0) = 1$ and $sin(0) = 0$ then the rotation matrix for R_y becomes the identity matrix thus losing a DOF because the identity matrix has no affect on rotation.

2.5.2 Quaternions

Quaternions is a different method of representing angles that are widely used in computer graphic applications [Shoemake, 1985]. Quaternions work on the principle of Euler's rotation theorem, i.e. that any rotation or sequence of rotations can be represented by a single rotation about a Euler axis. The Euler axis is denoted by a 3D vector such as (x_i, y_j, z_k) and the rotation by an angle θ. The general form of a rotation quaternion is given in Equation 2.1

Unlike Euler angles, Quaternions are not ambiguous and cannot suffer from gimbal lock [Grassia, 1998]. However, they are not simple to comprehend for humans.

$$q = [x_i \sin\theta/2, y_j \sin\theta/2, z_k \sin\theta/2, \cos\theta/2] \tag{2.1}$$

2.6 Transformation Matrices

Transformation matrices are used extensively in computer graphics for moving objects (position and orientation), rendering scenes, changing the camera field of view (FOV), 3D projection, and animation to just name a few uses [Bloomenthal and Rokne, 1994]. They are also used frequently in robotics [Murray et al., 1994]. For example in computer graphics they can be used to animate an articulated stick figure in order to depict movement of the limbs in a realistic manner. The same concepts could be applied to a robot with similar links and joints as the articulated stick figure. The focus of this section is on transformation matrix topics that apply to robotics only, topics such as scaling that only apply to computer graphics will not be discussed.

2.6.1 Notation

In this section, matrix entries are labelled m_{ik} where i is the row and k is the column as shown in Equation 2.2. θ_x, θ_y, and θ_z represent rotations about the associated axis.

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \tag{2.2}$$

2.6.2 Rotation Matrices

It is not obvious why 4x4 matrices are used, but if you recall homogeneous coordinates as described in Section 2.4, points are represented in the form (x, y, z, w). In our case the homogeneous coordinate w is always 1 for the transformation matrices. By doing this it allows us to combine translation transformations together with rotation transformations.

One rotation matrix is specified for each of the Euler angles. The rotation matrices for R_x, R_y, and R_z are given in Equations 2.3, 2.4, 2.5 respectively [Selig, 1992; Bajd, 2010]. It is possible to combine the three rotation matrices with simple matrix multiplication. There are six possible permutations, and the order matters. Two common permutations (yaw-pitch-roll and roll-pitch-yaw) are discussed in the following section.

$$Rx(\theta_x) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x & 0 \\ 0 & \sin\theta_x & \cos\theta_x & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.3}$$

$$Ry(\theta_y) = \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.4}$$

$$Rz(\theta_z) = \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 & 0 \\ \sin\theta_z & \cos\theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.5}$$

2.6.2.1 Combining Rotation Matrices

In order to achieve a single rotation matrix for yaw, pitch, and roll the matrices given in Equations 2.3, 2.4, 2.5 can be combined by performing simple matrix multiplication. The combination of yaw and pitch obtained by matrix multiplication is given in Equation 2.6 and the combination of yaw, pitch, and roll obtained by matrix multiplication is given in Equation 2.7 [Schneider and Eberly, 2002; Eberly, 2014].

$$Rx(\theta_x)Ry(\theta_y) = \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y & 0 \\ \sin\theta_x \sin\theta_y & \cos\theta_x & -\sin\theta_x \cos\theta_y & 0 \\ \cos\theta_x - \sin\theta_y & \sin\theta_x & \cos\theta_x \cos\theta_y & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.6}$$

$$Rx(\theta_x)Ry(\theta_y)Rz(\theta_z) = \begin{bmatrix} \cos\theta_y \cos\theta_z & \cos\theta_y - \sin\theta_z & \sin\theta_y & 0 \\ \sin\theta_x \sin\theta_y \cos\theta_z + \cos\theta_x \sin\theta_z & \sin\theta_x \sin\theta_y - \sin\theta_z + \cos\theta_x \cos\theta_z & -\sin\theta_x \cos\theta_y & 0 \\ \cos\theta_x - \sin\theta_y \cos\theta_z + \sin\theta_x \sin\theta_z & \cos\theta_x \sin\theta_y \sin\theta_z + \sin\theta_x \cos\theta_z & \cos\theta_x \cos\theta_y & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.7}$$

Likewise, for roll, pitch, and yaw, they are combined by performing simple matrix multiplication. The combination of roll and pitch obtained by matrix multiplication is given in Equation 2.8 and the combination of roll, pitch, and yaw obtained by matrix multiplication is given in Equation 2.9 [Schneider and Eberly, 2002; Eberly, 2014]. As stated in Section 2.5, the order of Euler angle rotations matter. This is apparent once again if the matrices in Equation 2.7 and 2.9 are compared since they are clearly different.

$$Rz(\theta_z)Ry(\theta_y) = \begin{bmatrix} \cos\theta_z \cos\theta_y & -\sin\theta_z & \cos\theta_z \sin\theta_y & 0 \\ \sin\theta_z \cos\theta_y & \cos\theta_z & \sin\theta_z \sin\theta_y & 0 \\ -\sin\theta_y & 0 & \cos\theta_y & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.8}$$

$$Rz(\theta_z)Ry(\theta_y)Rx(\theta_x) = \begin{bmatrix} \cos\theta_z \cos\theta_y & -\sin\theta_z \cos\theta_x + \cos\theta_z \sin\theta_y \sin\theta_x & \sin\theta_z \sin\theta_x + \cos\theta_z \sin\theta_y \cos\theta_x & 0 \\ \sin\theta_z \cos\theta_y & \cos\theta_z \cos\theta_x + \sin\theta_z \sin\theta_y \sin\theta_x & \cos\theta_z - \sin\theta_x + \sin\theta_z \sin\theta_y \cos\theta_x & 0 \\ -\sin\theta_y & \cos\theta_y \sin\theta_x & \cos\theta_y \cos\theta_x & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.9}$$

2.6.2.2 Extracting Euler Angles from Rotation Matrix

The Euler angles can be extracted from the rotation matrix [Groover, Weiss, and Nagel, 1986]. To extract the Euler angles from Equation 2.9, the Equations 2.10, 2.11, and 2.12 can be used. To extract the Euler angles from Equation 2.7, the Equations 2.13, 2.14, and 2.15 can be used.

$$Rz(\theta_z) = \arctan \frac{m21}{m11} \tag{2.10}$$

$$Ry(\theta_y) = \arctan \frac{-m31}{m11 \cos\theta_z + m21 \sin\theta_z} \tag{2.11}$$

$$Rx(\theta_x) = \arctan \frac{m13 \sin\theta_z - m23 \cos\theta_z}{m22 \cos\theta_z - m12 \sin\theta_z} \tag{2.12}$$

$$Rx(\theta_x) = \arctan \frac{m23}{m33} \tag{2.13}$$

$$Ry(\theta_y) = \arcsin -m13 \tag{2.14}$$

$$Rz(\theta_z) = \arctan \frac{m12}{m11} \tag{2.15}$$

For example we can extract the Euler angles from the rotation matrix if we plug in $R_x = 20°, R_y = 35°, R_z = 90°$ into Equation 2.9 then we can extract the

Euler angles using Equations (2.12, 2.11, 2.10) as demonstrated below in Equations (2.16, 2.17, 2.18, 2.19).

$$Rz(\theta_z)Ry(\theta_y)Rx(\theta_x)) = \begin{bmatrix} 0 & -0.94 & 0.34 & 0 \\ 0.82 & 0.20 & 0.54 & 0 \\ -0.57 & 0.28 & 0.77 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.16}$$

$$Rz(\theta_z) = \arctan \frac{0}{0.82} \tag{2.17}$$

$$Ry(\theta_y) = \arctan \frac{0 \cos \theta_z + 0.82 \sin \theta_z}{0.57} \tag{2.18}$$

$$Rx(\theta_x) = \arctan \frac{0.20 \cos \theta_z + 0.94 \sin \theta_z}{0.34 \sin \theta_z - 0.54 \cos \theta_z} \tag{2.19}$$

2.6.3 Translation Matrices

Suppose you wish to translate (x, y, z) by (d_x, d_y, d_z). All that must be done is add the individual components $(x + d_x, y + d_y, z + d_z)$. This can be expressed as a

translation matrix as shown in Equation 2.20 Bajd [2010].

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2.20}$$

2.7 Frames

Frames are local Cartesian coordinate systems that can be attached to arbitrary points [Cheng, 2010]. Frame are useful when you wish to specify the coordinates of a point relative to another point instead of in the global Cartesian coordinate system. Given three points we can calculate a frame. The calculated frame is a Cartesian coordinate with an orientation specified in Euler angles.

Given three robot points Origin, X, and Y. We take the Cartesian coordinates x, y, z of each. All rotations on the three points are completely ignored.

$o = [O_x, O_y, O_z]$

$x = [X_x, X_y, X_z]$

$y = [Y_x, Y_y, Y_z]$

We then calculate the signed relative distance along each axis. Depending on which side the X and Y points are on in comparison to the Origin it will change what positive X and Y are for the frame, in other words the handedness of the coordinate system is affected by how the X and Y points are chosen.

$a = x - o$

$b = y - o$

We then need to find three mutually orthogonal Euclidean vectors with our Origin that pass through X and Y. In layman's terms this gives us three directed line segments $(X+, Y+, Z+)$ starting from the origin that are perpendicular to one another. This can be done by calculating the cross product as shown below.

$i = ||a||$

$k = i \text{ x } b$

$k = k/||k||$

$j = k \text{ x } i$

An orthogonal matrix can be created by taking mutually orthogonal Euclidean vectors and using them as the rows of the matrix as shown in Equation 2.21. This homogeneous transformation matrix can be used to transform a frame point or local Cartesian coordinates, i.e. $(0, 0, 1)$ into global Cartesian coordinates with simple matrix multiplication.

$$\begin{bmatrix} i_x & j_x & k_x & 0 \\ i_y & j_y & k_y & 0 \\ i_z & j_z & k_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.21}$$

We can also express the transformation matrix or frame as a point with an orientation. From Section 2.6.2, we can extract the Euler angles out of the orthognal

matrix given in Equation 2.21. The frame Cartesian coordinates (x, y, z) would just be the coordinates of the origin point.

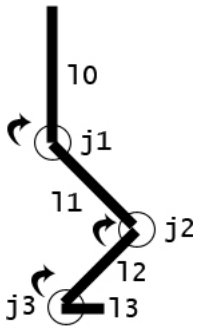
2.8 Kinematics

The kinematics of a humanoid robot allow us to determine how it will move without considering individual masses and the external forces acting on them [Spong et al., 2005]. In my work we are concerned with two kinematic problems in particular: Forward Kinematics (FK) and Inverse Kinematics (IK). As the names imply, the respective problems are the opposites of one another. Each problem is useful in its own unique way. Solving a FK problem is easier than solving an IK problem for a number of reasons, as we will see in Sections 2.8.2 and 2.8.3.

A humanoid robot can be modelled as a set of joints connected by links [Spong et al., 2005]. There are many different types of joints, but we will only consider revolute joints. This is without loss of generality because it is not difficult to model other joints in a similar manner. Revolute joints rotate on a single axis, therefore each revolute joint represents a single DOF. Each link has an associated length and is connected by two joints, which means there is always one more link than there are joints. Joints are labelled from $(1, ..., n)$ and links from $(0, ..., n)$. If joint i moves link i moves and so forth. Generally the *n-th* joint is connected to what is called the end-effector or tool and link 0 is fixed and does not move when any of the joints move.

For the purposes of introducing kinematics, we will consider a 4 link model of a humanoid robot to contrast with Spong et al. [2005] examples of robot manipulators. The kinematic model's joints and links are (j_1, j_2, j_3) and (l_0, l_1, l_2, l_3) respectively. Once again, all joints are revolute and the lengths of each link are as shown in Figure 2.5. The end-effector is the foot of our humanoid robot, so we can see how the foot will move irrespective of individual masses and the external forces acting on them. We will also only consider two dimensional coordinates for simplicity.

A simple robot model consists of links connected by joints as depicted in Figure 2.5. In order to solve the FK problem, a coordinate frame must be attached to each link as depicted in Figure 2.6. Coordinate frame $o_i x_i y_i z_i$ will be attached to link i and the coordinate frame of l_0 is the base frame. There is also a homogeneous transformation matrix T_j^i which represents the position and orientation of frame j relative to frame i. Although it is possible to attach the coordinate frames arbitrarily and still find the correct solution, it is better to have a standard convention of selecting coordinate frames that allows for comparison of results between colleagues working on the same problem. The Denavit Hartenberg (D-H) convention is a commonly used method for selecting frames of reference [Spong et al., 2005]. In this method, coordinate frames are selected using two rules. The first rule states that the x-axis x_i is perpendicular to the z-axis z_{i-1}; the second rule states that the x-axis x_i intersects z-axis z_{i-1}. Spong et al. [2005] show that four parameters (link length a_i, link twist α_i, link offset d_i, and joint angle θ_i) are enough to rep-



Link	Length	Connected to	Analogous to
l0	10cm	j1	Torso
l1	8cm	j1,j2	Thigh
l2	8cm	j2,j3	Shin
l3	2cm	j3	Foot
Joint	**Type**	**Connected to by**	**Analogous to**
j1	Revolute	j2,l1	Hip
j2	Revolute	j3,l2	Knee
j3	Revolute	n/a	Ankle

Figure 2.5: 4 link kinematic model of humanoid robot.

resent the homogeneous transformation matrix when these rules are satisfied. Once each homogeneous transformation matrix is known, it is possible to calculate the position and orientation of coordinate frame i relative to the base frame by right multiplying homogeneous transformation matrices $(T_0, ..., T_{i-1})$ where T_i is defined by Equations 2.22 and 2.23 [Spong et al., 2005].

2.8.1 Denavit Hartenberg (D-H) Parameters

D-H convention [Denavit and Hartenberg, 1955] represents each homogeneous transformation matrix as a product of four other transformations given four parameters [Spong et al., 2005]. The four parameters are:

- Link length: a_i The length of the connecting structure between two joints. In the cases where a joint is connected to the end effector it may be the length of the tool.

- Link twist: α_i The mounting orientation of the joint. For example the joint could be mounted at angle.

- Link offset: d_i The mounting offset of the joint. For example the joint's axis of rotation may not be exactly where the two links are connected.

- Joint angle: θ_i The angle of the joint.

The only two relevant parameters for my work are link length and joint angle, since I am only concerned about revolute joints. Spong et al. [2005] give us Equation 2.22 and 2.23. I have simplified these equations based on the example kinematic model of the humanoid robot given in Figure 2.5. This kinematic model only has revolute joints and is given in two dimensional coordinates, therefore my homogeneous transformation matrix can be simplified because α_i and d_i are equal to zero. After multiplying Equation 2.22 we obtain the simplified Equation 2.24. The last column of transformation matrix T_i are the coordinates for coordinate frame i relative to the base frame - or simply put, the coordinates of the end effector.

$$Rot_{z,\theta_i} Trans_{z,d_i} Trans_{x,a_i} Rot_{x,\alpha_i} = T_i =$$

$$\begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_i & -\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.22}$$

$$T_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.23}$$

$$T_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i & a_i\sin\theta_i \\ 0 & 0 & 1 \end{bmatrix} \tag{2.24}$$

In Figure 2.6, the coordinate frames are attached using D-H convention. Now that each homogeneous transformation matrix is known it is possible to calculate the position and orientation for coordinate frame i relative to the base frame by right multiplying homogeneous transformation matrices $(T0, ..., T_{i-1})$. The last column of the transformation matrix T_i are the coordinates for coordinate frame i relative to the base frame.

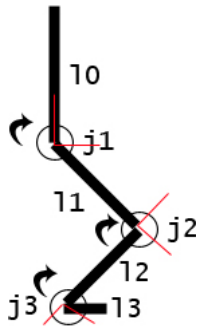


Figure 2.6: 4 link kinematic model of humanoid robot with coordinate frames attached.

2.8.2 Forward Kinematics (FK)

Given Equation 2.24 the FK is solved by right multiplying homogeneous transformation matrices as shown in Equation 2.25. Using a simplified model of a robot, the only parameters necessary to solve the FK are link lengths and joint angles.

$$T_i = T_0 T_1 T_2 ... T_{i-1} \tag{2.25}$$

2.8.3 Inverse Kinematics (IK)

The Inverse Kinematics (IK) problem for a robot manipulator is to find the joint angles given the goal position and orientation of the end effector. There is no general closed form solution to the IK problem for robot manipulators with greater than 6 DOF [Spong et al., 2005], therefore numerical solutions or hill climbing methods are used in such situations [Selig, 1992]. Finding a solution to the IK problem

requires solving many non-linear equations simultaneously, which is a difficult task in mathematics. Unlike the FK problem, the IK problem may not always have a solution, or a unique solution. Even if a solution exists, it may be hard to obtain. In fact, manipulators are often designed with simplification of the IK problem in mind so that techniques such as kinematic decoupling can be used [Ali et al., 2010]. Kinematic decoupling allows the position and orientation of the end effector to be solved separately. Kinematic decoupling is only possible if the design of the kinematic chain supports position and orientation separation. An example of such a design is a robot manipulator with a spherical wrist.

Physical systems fall into two groups: holonomic and non-holonomic. In a holonomic system, the controllable DOF are equal to the total DOF. In a non-holonomic system, the controllable DOF are less than the total DOF. For a non-holonomic robot, the goal configuration of the end effector depends on the trajectory from the initial configuration, unlike a holonomic robot, where the trajectory is irrelevant. Redundant manipulators have more DOF than the dimension of the work space position vector, but can be either be holonomic or non-holonomic. Having a redundant manipulator can present more challenges when solving the IK problem because of singularities, which are caused my multiple configurations that produce the same end effector configuration.

2.8.3.1 Jacobian Matrix

One method of numerically solving the IK problem is by using a Jacobian matrix [Buss, 2004]. A Jacobian matrix consists of entries that are first-order partial derivatives. A partial derivative of a multi-variable function represents how changing a single variable affects the function when the other variables are kept constant. The Jacobian matrix is a linear approximation of the multi-variable function. For a robotic manipulator, the partial derivatives represent how each DOF affects the end effector position and orientation. The multi-variable function represents the FK, therefore the Jacobian matrix is an approximation of the FK. In most literature, the notation for the Jacobian matrix is J. Since J approximates the FK, it follows that J^{-1} is an approximation for the IK. An example of a Jacobian matrix is given in Equation 2.26 for a 3 DOF robotic manipulator. Each column represents a joint and each row a component of the tool position (x, y, z) and orientation (R_x, R_y, R_z).

$$J = \frac{\partial t}{\partial \theta_i} = \begin{bmatrix} \frac{\partial tx}{\partial \theta_0} & \frac{\partial tx}{\partial \theta_1} & \frac{\partial tx}{\partial \theta_2} \\ \frac{\partial ty}{\partial \theta_0} & \frac{\partial ty}{\partial \theta_1} & \frac{\partial ty}{\partial \theta_2} \\ \frac{\partial tz}{\partial \theta_0} & \frac{\partial tz}{\partial \theta_1} & \frac{\partial tz}{\partial \theta_2} \\ \frac{\partial trx}{\partial \theta_0} & \frac{\partial trx}{\partial \theta_1} & \frac{\partial trx}{\partial \theta_2} \\ \frac{\partial try}{\partial \theta_0} & \frac{\partial try}{\partial \theta_1} & \frac{\partial try}{\partial \theta_2} \\ \frac{\partial trz}{\partial \theta_0} & \frac{\partial trz}{\partial \theta_1} & \frac{\partial trz}{\partial \theta_2} \end{bmatrix} \tag{2.26}$$

In order to solve the IK problem with the Jacobian matrix two things must

be done: first the Jacobian matrix must be calculated, and second the Jacobian matrix must be inverted. Inverting the Jacobian matrix is problematic because not all matrices are invertible. Calculation of the Jacobian matrix is discussed in Section 2.8.3.2 and alternatives to inverting the Jacobian matrix are discussed in Sections 2.8.3.3 and 2.8.3.4.

2.8.3.2 Calculating the Jacobian Matrix

The Jacobian matrix can be determined analytically or numerically. Buss [2004] give us Equation 2.27 to analytically determine each entry of the Jacobian matrix for a revolute joint. In the equation, θ_j represents the angle of joint j, t_i is the tool position of tool i, v_j is the unit vector point along the axis of rotation of joint j, and p_j is the position of joint j.

$$\frac{\partial t_i}{\partial \theta_j} = v_j \times (t_i - p_j) \tag{2.27}$$

The entries of the Jacobian matrix can be numerically determined if the FK can be solved using the transformation matrices as described in Section 2.8.2. Given the current state of the robotic manipulator (joint angles), the FK is solved and saved for reference. Each joint can then be changed in turn by a small amount $\Delta\theta_i$ and the FK solved again. The difference in position and orientation from the before and after FK solutions yields the entries for the Jacobian matrix as shown in Equation 2.28. Pseudocode for numerically determining the Jacobian matrix is given in Algorithm 1.

$$\frac{\partial t}{\partial \theta_i} approximately = \frac{\Delta t}{\Delta \theta_i} = \begin{bmatrix} \frac{\Delta tx}{\Delta \theta_0} & \frac{\Delta tx}{\Delta \theta_1} & \frac{\Delta tx}{\Delta \theta_2} \\ \frac{\Delta ty}{\Delta \theta_0} & \frac{\Delta ty}{\Delta \theta_1} & \frac{\Delta ty}{\Delta \theta_2} \\ \frac{\Delta tz}{\Delta \theta_0} & \frac{\Delta tz}{\Delta \theta_1} & \frac{\Delta tz}{\Delta \theta_2} \\ \frac{\Delta trx}{\Delta \theta_0} & \frac{\Delta trx}{\Delta \theta_1} & \frac{\Delta trx}{\Delta \theta_2} \\ \frac{\Delta try}{\Delta \theta_0} & \frac{\Delta try}{\Delta \theta_1} & \frac{\Delta try}{\Delta \theta_2} \\ \frac{\Delta trz}{\Delta \theta_0} & \frac{\Delta trz}{\Delta \theta_1} & \frac{\Delta trz}{\Delta \theta_2} \end{bmatrix} \tag{2.28}$$

2.8.3.3 Jacobian Transpose

Since all matrices are not invertible, an alternative is to use the transpose instead of the inverse as shown in Equation 2.29 [Buss, 2004]. In the equation, α is a scalar and e is a vector that represents the error in the desired tool position and orientation. The transpose J^T is obviously not equal to J^{-1}, however Buss [2004] provide a proof that for a sufficient small α the error vector e is reduced. By iteratively using Equation 2.29 with a small α value the error vector e will eventually be reduced to zero. Pseudocode for using the Jacobian transpose with an incremental simulator to solve the IK problem is given in Algorithm 2.

$$\Delta\theta = \alpha J^T e \tag{2.29}$$

Algorithm 1 Jacobian

procedure JACOBIAN
    $prevTool \leftarrow Tool$ ▷ Save the current position and orientation of the tool. Uses FK to determine tool position and orientation.
    for $i \leftarrow 0, n$ **do** ▷ Where n is the number of joints.
        $joint[i] \leftarrow joint[i] + \epsilon$ ▷ Change joint angle by small amount epsilon.
        $curTool \leftarrow Tool$ ▷ Retrieve the current position and orientation of the tool. Uses FK to determine tool position and orientation.
        $deltaTool \leftarrow (curTool - prevTool)/\epsilon$ ▷ Calculate change in tool position and orientation caused by change to joint[i].
        $J[i] \leftarrow deltaTool$ ▷ Save Jacobian entry for joint[i], how changing joint[i] affects x, y, z, rx, ry, rz of tool position and orientation.
        $joint[i] \leftarrow joint[i] - \epsilon$ ▷ Restore joint angle.
    end for
    return J
end procedure

2.8.3.4 Jacobian Pseudo Inverse

Another alternative to inverting the Jacobian is using the pseudo inverse, also known as the Moore-Penrose inverse [Buss, 2004]. Unlike the inverse, the pseudo inverse can be determined for any matrix. The equation for the Jacobian pseudo

Algorithm 2 Using Jacobian Transpose with Incremental Simulator

1: **procedure** MOVE($targetTool$)

2: $curTool \leftarrow Tool$ ▷ Retrieve the current position and orientation of the tool. Uses FK to determine tool position and orientation.

3: $errorTool \leftarrow targetTool - curTool$ ▷ Calculate the error in tool position and orientation.

4: $errorTool \leftarrow Clamp(errorTool, max)$ ▷ Limit magnitude of error for time step.

5: **while** $errorTool > \epsilon$ **do**

6: $JT \leftarrow JacobianTranspose()$ ▷ Calculate Jacobian Transpose.

7: **for** $i \leftarrow 0, n$ **do** Where n is the number of joints.

8: $deltaJoint[i] \leftarrow (JT[i] * errorTool) * \alpha$ ▷ Where

9: alpha is significantly small to prevent overshoot.

10: $joint[i] \leftarrow joint[i] + deltaJoint[i]$ ▷ Set desired joint[i] angle.

11: **end for**

12: $IncrementalStep()$ ▷ Perform incremental simulator step.

13: $curTool \leftarrow Tool$ ▷ Retrieve the current position and orientation of the tool. Uses FK to determine tool position and orientation.

14: $errorTool \leftarrow targetTool - curTool$ ▷ Calculate the error in tool position and orientation.

15: $errorTool \leftarrow Clamp(errorTool, max)$ ▷ Limit magnitude of error for time step.

16: **end while**

17: **return** $targetReached$

18: **end procedure**

inverse is given in Equation 2.30 [Meredith and Maddock, 2004]. The Jacobian pseudo inverse can be used similarly to the transpose as shown in Equation 2.31. In the equation, e is a vector that represents the error in the desired tool position and orientation.

$$J^{\dagger} = (J^T J)^{-1} J^T \tag{2.30}$$

$$\Delta\theta = J^{\dagger} e \tag{2.31}$$

2.9 Polygon of Support (POS) and Zero Moment Point (ZMP)/Center of Pressure (COP)

For mobile robots, motion planning is restricted to movements that can be realized with a particular robot's physical attributes. Wheeled robots have had considerable attention because of their inherent dynamic stability during movement. Although dynamic stability is desirable, wheeled robots sacrifice mobility on uneven terrain such as stairs [Fujiwara et al., 2004]. Humanoid robots are much more suitable to navigate environments such as stairs that are designed for humans. However, humanoid robots, unlike wheeled robots, can fall over. The fact that humanoid robots can fall over further complicates the motion planning problem because the motion plan must be validated to ensure all steps will be dynamically stable. This

validation in itself is a difficult problem.

A common task that a humanoid robot might perform during a motion plan is walking. It is quite easy for a humanoid robot to fall over while walking on uneven terrain such as stairs. In much of the literature, walking is referred to as biped locomotion for humanoid robots. Biped locomotion is a periodic gait cycle composed of two periods, stance and swing repeated continuously [Ayyappa, 1997]. The gait cycle of humanoid robot must change under different circumstances. For example when moving from concrete to grass or from a level surface to an incline, the gait cycle must change. A humanoid robot is in the stance period for approximately 62 percent of a gait cycle [Ayyappa, 1997]. If you assume that the gait cycle begins in the stance period and both feet are touching the ground (double support) with the left foot in front of the right foot, the first half of the gait cycle will end in double support with the right foot in front of the left foot, as depicted in Figure 2.7. A humanoid robot is in the swing period for approximately 38 percent of a gait cycle [Ayyappa, 1997]. During the swing period only one foot is touching the ground (single support) and the other is above the ground “swinging" forward. The front foot of the preceding double support phase becomes the single support of the swing period. The second half of the gait cycle will continue from the double support phase of the first half of the gait cycle, and swing the opposite foot ending in the original double support configuration (right foot in front of the left foot). As the velocity of biped locomotion increases from walking to running, the time spent in the double

support phase decreases to zero. Two feet never touch the ground at the same time while running, only alternating single support (left/right).

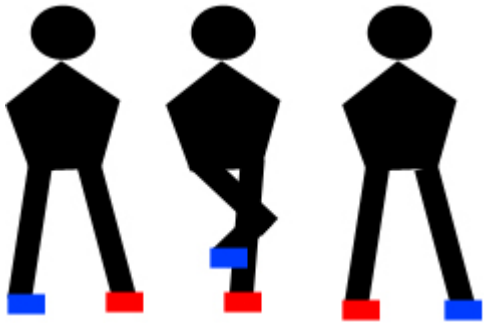


Figure 2.7: First half of gait cycle

The following concepts are presented to understand a method that guarantees that a humanoid robot's biped locomotion is dynamically stable throughput the gait cycle:

The Polygon of Support (POS) is the convex hull of all contact points with the ground [Vukobratovic et al., 2006] . The Zero Moment Point (ZMP) Vukobratovic et al. [2006], also known as the Center of Pressure (COP) Pratt [2000a] is the point where the sum of all moments are equal to zero or the distance-weighted average location of individual pressures on the foot. Although these two terms are used synonymously, there is in fact a subtle difference. The COP exists as long as the foot is in contact with the ground, and the ZMP does not exist if it lies outside of the POS. If the ZMP is within the POS for a given phase of a gait cycle, the phase is balanced. In Figures 2.8 and 2.9, the bottom of a humanoid robot's feet

are depicted from the top down (black). The convex hull (blue) of all contact points (red) is also depicted. The feet are the only contact points with the ground during biped locomotion. It is clear from Figures 2.8 and 2.9 that the POS of the double support phase is larger than the single support phase. The double support phase of a gait cycle is intuitively easier to balance than single support phase due to this size advantage. A larger POS makes it easier to ensure the ZMP lies within it.

Controlling the ZMP - that is, ensuring it lies within the POS - is sufficient to achieve balanced biped locomotion on a humanoid robot [Bagheri et al., 2006; Ha et al., 2007; Tang et al., 2003].

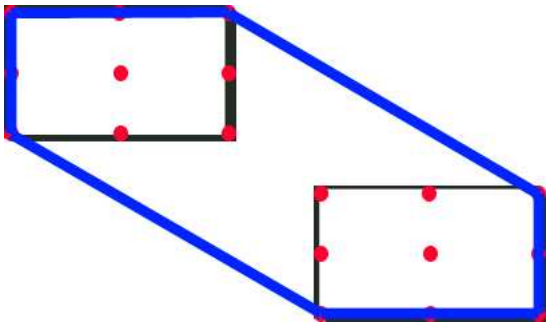


Figure 2.8: Double support POS

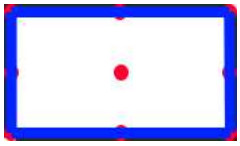


Figure 2.9: Single support POS

2.10 Inverted Pendulum

The inverted pendulum is a well known dynamics problem [Pratt, 2000b]. An inverted pendulum consists of a mass m above a base with a pivot point attached to a rod with length l. If the mass is to the left of the pivot point, the mass accelerates backward. If the mass is to the right of the pivot point, the mass accelerates forward. The problem requires the control to maintain the mass in a upright position. Solutions to the inverted pendulum and many variations of the problem are well defined. Pratt [2000b] has shown that the inverted pendulum model can be adapted to model the dynamics of a humanoid robot.

2.11 Proportional-Integral-Derivative (PID)

A PID controller is a well known algorithm in control theory. It is estimated that a PID controller is used to solve 90 to 95 percent of all control problems [Ledin]. It can be used in systems where the set point error can be directly related to the output of the controller. The are three gains that are used to tune the controller. The first is called the proportional gain, and as the name suggests it is a constant multiplier

that provides a response proportional to the error. The second is called the integral gain, it is a constant multiplier used on the sum of the error. The last gain is called the derivative gain, once again as the name suggests it is a constant multiplier used on the change in error. The PID controller algorithm is given in Algorithm 3.

The integral gain is applied to the sum of error accumulated at every iteration of the PID controller. If there is continuous error, a phenomenon called integral wind-up can occur. The accumulation of continuous error can result in overshooting the set point. This is an undesired behavior of the PID controller. This can happen if the system is blocked, for example if the PID controller is controlling the speed of a toy car and it is up against a wall. Typically to prevent integral wind-up the integral part is limited, as shown in Algorithm 4.

Algorithm 3 PID

1: **procedure** PID

2: $error_i \leftarrow target - current$

3: $deltaError_i \leftarrow error_i - error_{i-1}$

4: $sumError_i \leftarrow \sum_{i=0}^{n} error_i$

5: $output_i = (KP * error_i) + (KI * sumError_i) + (KD * deltaError_i)$

6: **return** $output_i$

7: **end procedure**

Algorithm 4 PID with Integral Limit

1: **procedure** PID
2: $error_i \leftarrow target - current$
3: $deltaError_i \leftarrow error_i - error_{i-1}$
4: $sumError_i \leftarrow \sum_{i=0}^{n} error_i$
5: **if** $sumError_i > LIMIT$ **then**
6: $sumError_i \leftarrow LIMIT$
7: **end if**
8: $output_i = (KP * error_i) + (KI * sumError_i) + (KD * deltaError_i)$
9: **return** $output_i$
10: **end procedure**

2.12 Nearest Neighbor (NN)

The Nearest Neighbor (NN) search problem in CSPACE is defined as follows: given a set S of configurations and a specific configuration p in the set, the nearest neighbor problem is to determine the closest configuration q to p [Muja and Lowe, 2009]. NN search methods typically use space partitioning algorithms as opposed to the naive approach. The naive approach calculates the distance between every configuration in the set, therefore it has a worst case complexity of $O(n)$ where n is the number of configurations. For a large set of configurations this is not feasible if the NN must be computed in real-time [Nene and Nayar, 1997]. There are some known space partitioning approaches that can execute in logarithmic time. For example,

space partitioning algorithms such as those that use a k-dimensional (k-d) tree data structure [Zhou et al., 2008] have an average complexity of $O(log_2 n)$ [Nene and Nayar, 1997]. Although it may seem that there is no point in comparing the naive approach to a space partitioning algorithm, the naive approach uses less memory because no complex data structure is required. The naive approach can outperform space partitioning algorithms as the number of dimensions increases [Weber et al., 1998].

An effective NN search algorithm is critical for both roadmap and tree based planners. The most popular NN search methods use k-d trees to solve the problem, which usually perform better than other search methods in euclidean space [Yershova and LaValle, 2007]. For roadmap planners, connecting vertices requires an NN search method, while tree planners must similarly connect a random configuration to the nearest configuration in the tree. The NN search is used many times during a planner execution, therefore the NN search method must be efficient.

2.13 Voronoi Diagram

The Voronoi diagram is a fundamental data structure in computational geometry [Aurenhammer, 1991]. A Voronoi diagram can be constructed using a NN method. Given n seed sites in a plane, the Voronoi diagram is constructed by partitioning the plane into regions that are convex polygons that contain every point that is closer to a specific seed site than any of the other seed sites. The NN method is used to

determine which seed site the points are the closest to. An example of a Voronoi diagram in 3D is given in Figure 2.10.

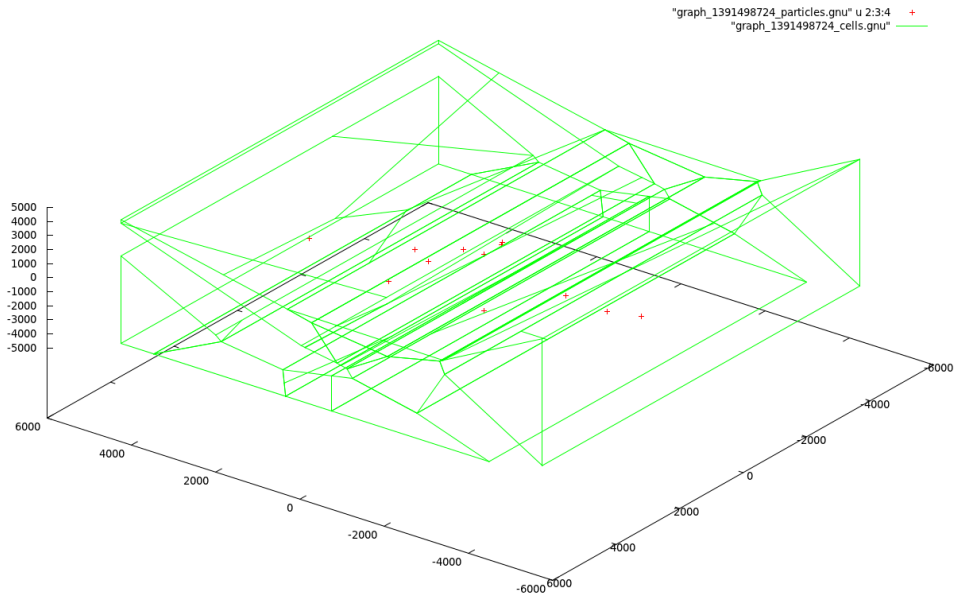


Figure 2.10: Voronoi Diagram in 3D

2.14 Graph Theory

The solution strategy that I propose in the remainder of this thesis generates a data structure that contains the solution. Methods to extract the solution from the data structure are well defined in the graph theory domain and are not the focus of this research. However, in order to understand the data structure discussed in Chapter 4 and how to extract the motion plan from the data structure, this section

discusses basic concepts in graph theory.

A graph is a set of vertices connected by edges. The edges can be bi-directional or uni-directional and have an associated weight (cost) for traversing the edge [Brassard and Bratley, 1996].

A common problem once a graph is defined is finding the shortest (or lowest cost) path through it, given a start and end vertex. Two algorithms for solving the shortest path problem are given in Section 2.14.1

In the graph shown in Figure 2.11, there are four vertices $(v0, v1, v2, v3)$ and five uni-directional edges $(e0, e1, e2, e3, e4)$ with associated weights $(5, 3, 2, 5, 1)$ respectively. If the start vertex is v0 and the desired end vertex is $v3$, then there are a number of possible paths with associated costs $(v0, v1, v3) = 7$, $(v0, v2, v3) = 8$, $(v0, v2, v1, v3) = 6$. The cost is the sum of weights for each edge traversed. As you can see even though a path must visit more vertices, it can have a lower cost.

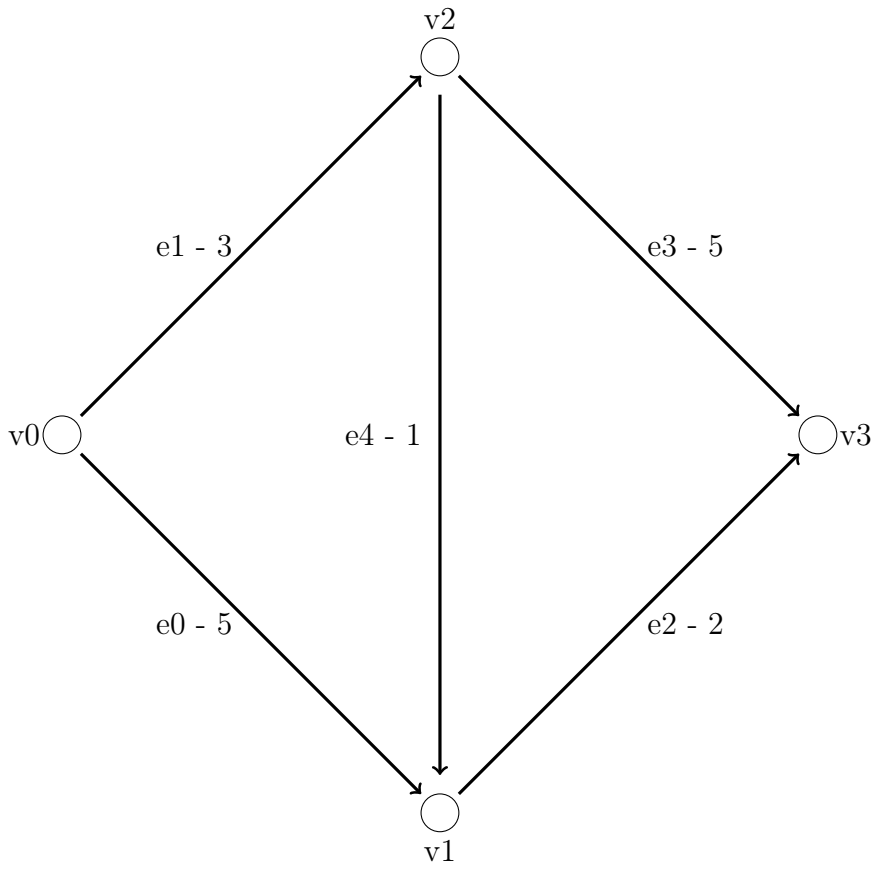


Figure 2.11: Example Directed Weighted Graph

2.14.1 Shortest Path Problem

2.14.1.1 Dijkstra's Algorithm

Dijkstra's algorithm is a greedy algorithm for solving the shortest path problem for a weighted graph [Brassard and Bratley, 1996]. The time complexity of Dijkstra's algorithm is $O(|V|^2)$, but can be improved by using specialized data structures. The algorithm maintains two sets of vertices, S and C. S contains all the vertices that

have already been inspected in the graph. C contains all other vertices in the graph. An array D is also maintained that holds the length of the shortest path to each vertex in the graph. Initially the starting vertex is removed from C and added to S. D is updated as each vertex is inspected. Each vertex is removed from C greedily (smaller weight first), inspected, and added to S in turn. Vertices are inspected by checking if its neighbors (vertices connected by an edge) have a smaller weight, and if it does D is updated to reflect the shorter path between the two vertices. When the algorithm completes C is an empty set and S contains all vertices in the graph. The shortest path from the start to end vertex can then be easily extracted from D.

2.14.1.2 A* Algorithm

The A* algorithm, like Dijkstra's algorithm, is also greedy. It is for the most part the same as Dijktra's algorithm except for the addition of a heuristic function that is considered in addition to the cost. If a heuristic function can be tailored for a specific graph layout, then A* algorithm can perform better than Dijkstra's algorithm.

2.14.2 Trees

A tree is a common data structure used that gets its name from the manner it which it looks as well as the way it is built [Mehlhorn and Sanders, 2007]. Every tree has a root node which is the topmost node. The tree is built by adding branches from the root node to other nodes. A node without any branches is considered a leaf node.

An example of a tree is shown in Figure 2.12. There are numerous variants of the tree data structure that attempt to optimize searching or space by incorporating rules to guide the insertion of nodes: for example binary and red-black trees Mehlhorn and Sanders [2007]. The variant of the tree used mostly depends on the application, since there are advantages and disadvantages to all variants. A tree can be thought of as a specialized graph as well, where nodes are vertices and branches are edges.

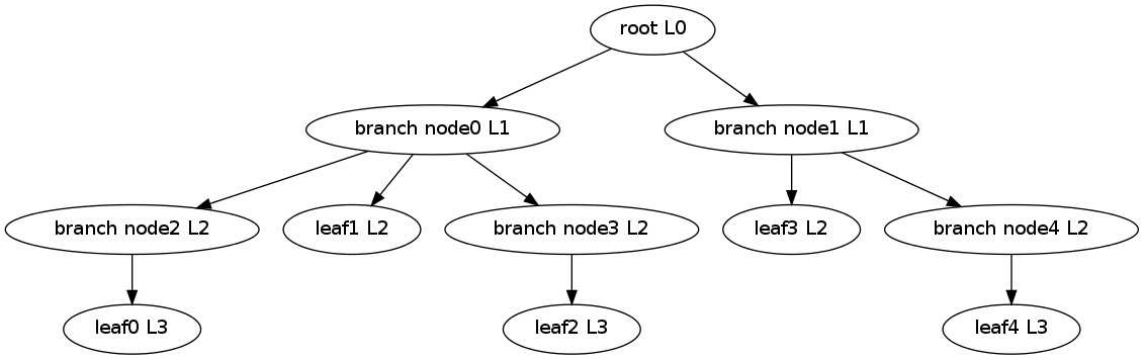


Figure 2.12: Tree

Trees can searched with breadth-first or depth-first search algorithms [Mehlhorn and Sanders, 2007]. RRTs do not even require searching to extract the path. Given the leaf node or goal configuration, all parent nodes including the root or initial configuration make up the path, because of the way the tree is constructed the parents are simply to determine. The time complexity of breadth-first search is $O(|V| + |E|)$. The time complexity of depth-first search if used to traverse the entire the tree is $O(|E|)$.

2.15 Incremental Simulator

In the solution strategy presented in my thesis, the motion planner is coupled with an incremental simulator as described in Section 5.3.

An incremental simulator computes the current state after a set of inputs has been applied over a period of time. Given state $x(t)$ a function is defined which solves $x(t + \delta t)$. Recently Sucan et al. [2008] have successfully combined sample-based tree planners with a physics engine. The physics engine was used as an incremental simulator to handle kinodynamic constraints and compute state $x(t + \delta t)$.

2.16 Collision Detection

In this thesis, collision detection is used to determine if a particular robot configuration lies completely within the CFREE set (no collision) or if it partly lies in the COBS set (collision).

Collision detection checks if two or more objects have collided with one another. Beyond detecting the fact that a collision has occurred, the algorithm should also detect the precise points at which contact was made.

One method of performing collision detection is by using bounding objects. This method of collision detection uses a simpler object that bounds the actual object. The simpler object is chosen such that it is easier to check for collision. Some examples of bounding objects are spheres [Kavan and Žára, 2005], boxes [Koziara

and Bicanic, 2007], and polygons [Basch et al., 2004].

The advantage of using bounding spheres as shown in Figure 2.14 is that they are very fast. All that must be done is to compute the radius of a sphere that bounds the object, but this only needs to be done once per object. When using bounding spheres there is a collision between two objects if the distance from one bounding sphere's origin to the other is less than the sum of the two bounding spheres' radii. What this means is that the two object's bounding regions overlap, therefore it is considered a collision. An example of bounding spheres in 2D is given in Figure 2.13. In the example, $d = 4$ is greater than $r1 + r2 = 3$, therefore there is no collision. One major disadvantage of using bounding spheres is that spheres do not bound all object shapes with minimal empty volume. Undesired empty volume creates false positives. A potential solution to the empty volume problem is to reduce the empty volume by generating many small spheres, as shown in Figure 2.15, instead of one big sphere.

One advantage of using bounding boxes as shown in Figure 2.16 is that they bound some objects with less empty volume than spheres, but this is also true for spheres for certain objects. Using bounding boxes is also slower the bounding spheres because bounding boxes must be calculated at every iteration since the orientation of object changes the bounding box.

Finally, bounding polygons, as shown in Figure 2.17, define bounds on objects with minimal empty volume. If you wish to add margin to the polygon so that near

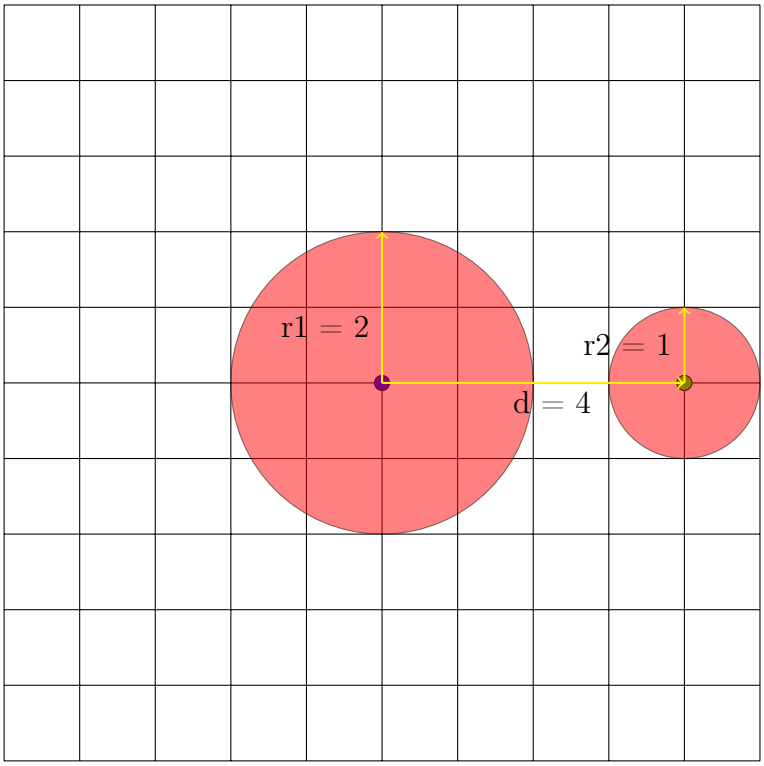
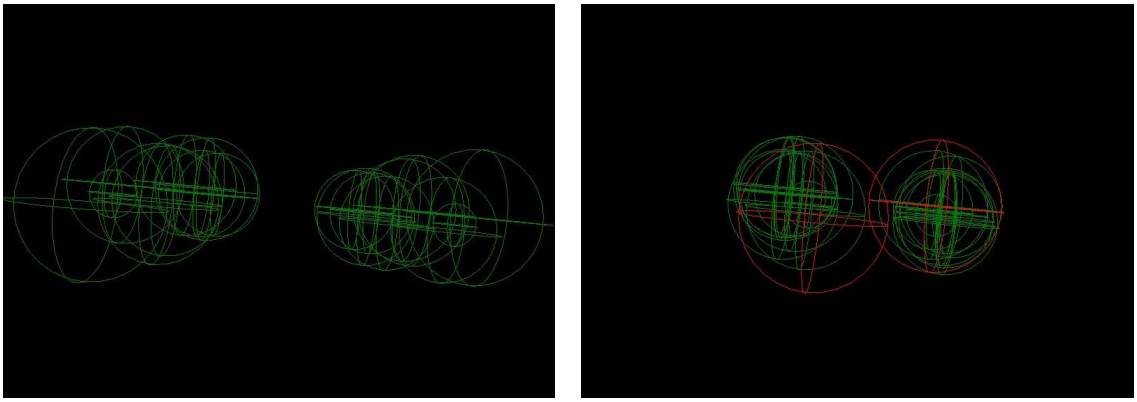


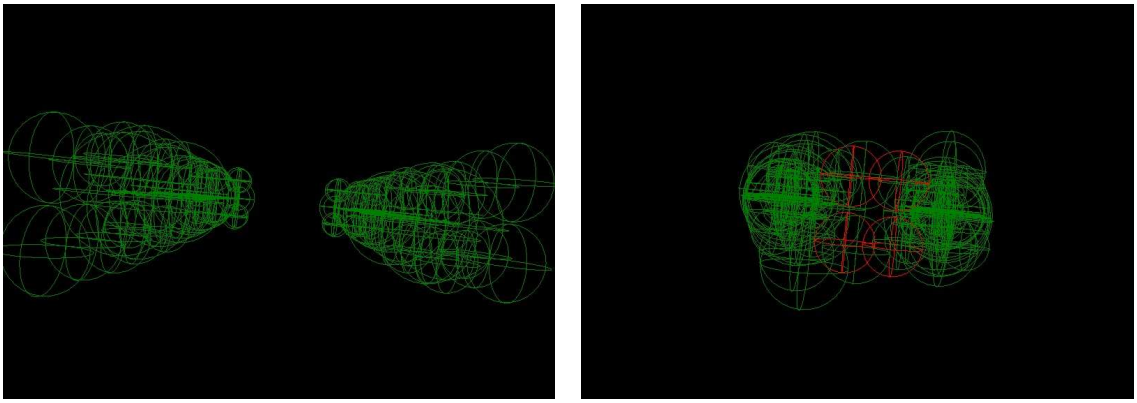
Figure 2.13: Bounding Spheres Example in 2D

collisions are detected early, the margin can be controlled precisely. For example, a 50mm margin around the entire robot can be specified. The major disadvantages of using bounding polygons is that it is very slow since intersections between many triangles must be calculated at every iteration.



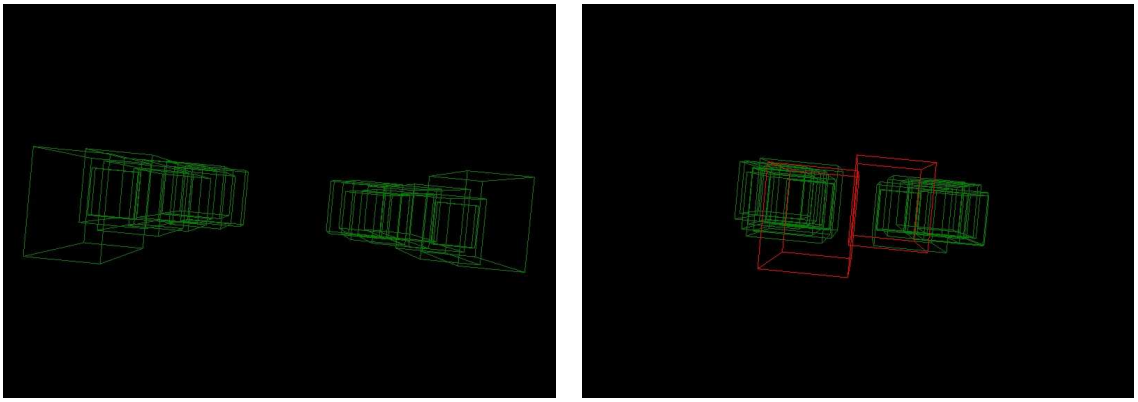
(a) Bounding Spheres, No Collision (b) Bounding Spheres, Collision

Figure 2.14: Bounding Spheres



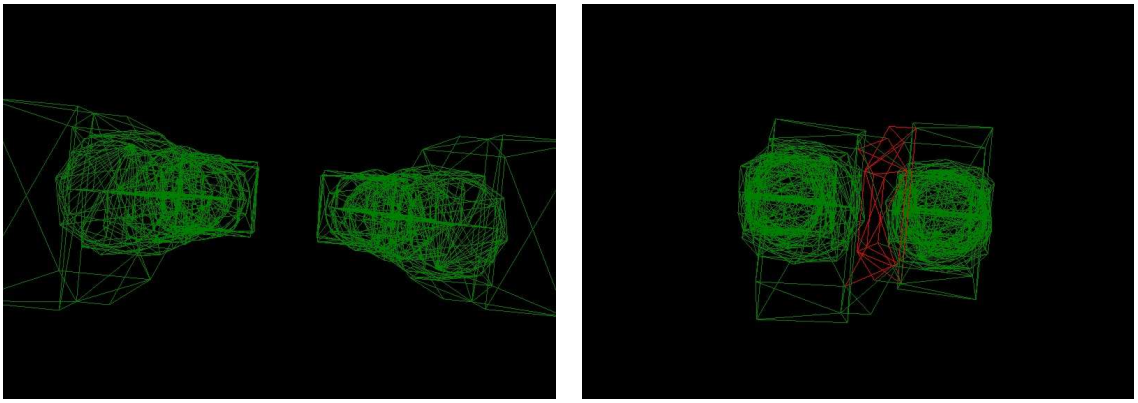
(a) Bounding Multi-Spheres, No Collision (b) Bounding Multi-Spheres, Collision

Figure 2.15: Bounding Multi-Spheres



(a) Bounding Boxes, No Collision (b) Bounding Boxes, Collision

Figure 2.16: Bounding Boxes



(a) Bounding Polygons, No Collision (b) Bounding Polygons, Collision

Figure 2.17: Bounding Polygons

2.17 Summary

A wide array of topics in robotics were covered in this chapter in order to provide sufficient background information to readers with little to no robotics knowledge. These topics are referenced throughout the remainder of this thesis when they are applicable. Now that the basics have been covered, a survey of the current popular robot motion planning techniques are presented. The survey aims to provide a broad picture of the evolution of solution strategies and how they are applicable to specific problems.

Chapter 3

Related Work

There have been various attempts to solve the robot motion planning problem. The following is an account of the most notable techniques developed thus far. This chapter provides insight into the methodologies attempted to solve the robot motion planning problem thus far. The advantages and disadvantages of each methodology are discussed in addition to how they relate to my thesis work.

3.1 Randomized Potential Fields

Potential fields use the metaphor of magnetic field or gas spreading [Vaščák, 2007]. Potential fields have been used to solve many types of problems such as obstacle avoidance and path planning in the past. Khatib [1986] first applied potential fields to real-time robotic manipulator collision avoidance. The idea behind potential fields

is that obstacles repel and the goal/open space attract thus pulling the robot toward the goal. The robot avoids collisions with obstacles by a repulsive force between them. This repulsive force is the negative gradient of the potential field [Hwang and Ahuja, 1992]. One of the biggest problems with using potential fields is that it is very easy to get trapped in local minima [Ge and Cui, 2000]. For example, the robot can be attracted to an area initially because the potential is high. If this area does not contain the goal, the robot can become trapped if no surrounding areas have a higher potential. Another problem is that definition of the *potential function* can be difficult and domain specific. The potential function defines the repulsive forces of obstacles and the attractive forces of the goal/open space. The relation between potential fields and my thesis work is the need for a real-time collision avoidance methodology.

Randomized Potential Fields have been applied to robot motion planning [LaValle, 2006]. The randomized potential field planner attempts to overcome the local minima problem of standard potential fields by performing random walks when it appears to be trapped. This begins to allude to a probabilistic approach for solving the robot motion planning problem, as used in my solution strategy. There remains one major problem with randomized potential fields that cannot be ignored. As with standard potential fields, the assignment of potential requires many heuristic functions and cannot be easily adapted to different robots without tweaking all of the heuristic functions. The goal of my research is to design a motion planner that is applicable

to many different robots without the need for tedious customization for each robot.

3.2 Probabilistic Road Map (PRM)

Sample-based roadmap planners, such as the Probabilistic Road Map (PRM) introduced by Kavraki et al. [1996], must first build a map (learning phase) then search for a solution (query phase). The learning phase creates the PRM by randomly generating configurations in CFREE which are then connected to nearby configurations. There are many different ways to generate random configurations and connect them. Geraerts and Overmars [2002] present an in-depth analysis of the different techniques used in variants of the PRM. The learning phase does not consider the initial or goal configuration. A map for the entire CSPACE is created. This is a waste of time if a specific task must be accomplished, as opposed to navigating the entire environment. The map also becomes invalid if the environment changes. For this reason, roadmap planners work best for holonomic robots in static environments where the map can be queried multiple times once generated because of the cost incurred by the learning phase [Lavalle, 1998; Lavalle and Kuffner Jr., 2000]. The PRM is a multi-query planner whereas the solution strategy presented in my thesis is a single-query planner that does not require a learning phase. My solution strategy, on the other hand, only considers the portions of the environment necessary to find a suitable motion plan.

The underlying data structure of a PRM is an undirected graph where nodes

represent configurations and edges represent paths. The initial and goal configuration are inputs to the query phase. The graph can be efficiently traversed using already well defined methods in graph theory such as the Dijkstra or A* search algorithm described in Section 2.14.1. The underlying data structure of my solution strategy is a tree instead of a graph. The tree also does not require a complex algorithm to extract the motion plan once it is generated.

One advantage of using PRMs is that there are very few parameters and heuristics required for implementation [Kavraki et al., 1996]. This makes them a better choice than randomized potential field planners. A major disadvantage of using PRMs is the number of connections that must be made between configurations which is not a simple task [Lavalle, 1998]. My solution strategy does not require as many connections between configurations which reduces the time to find a suitable motion plan. For a large CSPACE the learning phase is an expensive operation because of the logic required to make connections. In an uncertain environment with real-time requirements, the learning phase (think) does not allow sufficient time to react and the map cannot be reused if the environment changes rapidly.

3.3 Ariadne's Clew Algorithm

Ariadne's Clew algorithm [Mazer et al., 1996] is a similar algorithm to the PRM algorithm, but it does not explore the CSPACE. Instead it explores the trajectory space. The trajectory space is different than the CSPACE and the TSPACE discussed

in Sections 2.1 and 2.2. The trajectory space represents the whole path of the robot with a single point. The point coordinates are all of the sequential movements for the robot [Mazer et al., 1996]. For example, instead of a 3D point (x, y, z) in Cartesian coordinates, a trajectory space point's coordinates might be (forward, left, forward, right). To find a path, a single point in the trajectory space must be found.

The Ariadne's Clew algorithm, like the PRM, uses two steps: search (query phase) and explore (learning phase). The explore step collects information about the free space with increasing resolution. In the explore step, landmarks are placed in the search space that have a known path from the initial configuration. Exploration attempts to uniformly place the landmarks in the search space by placing them as far as possible from one another. The explore step is essentially map building, similar to that done in a PRM.

The search step checks if the goal can be reached. This is the same as the query phase in the PRM. However, the search step is executed multiple times in parallel with the explore step. This means that the Ariadne's Clew algorithm can find a path before the entire map is generated. This can lead to a slight performance increase over the PRM since the PRM requires the entire map to be built. Another benefit of being able to execute the search step in parallel with the explore step is that it lends to parallel computing [Mazer et al., 1996]. Use of parallel computing techniques can achieve further performance gains. The design of robot motion planning algorithms for parallel computing is an interesting topic. My thesis work, however, does not

attempt to address parallel computing implementations.

3.4 Flexible Binary Space Partitioning (BSP)

Cell decomposition methods such as Binary Space Partitioning (BSP) break down the CSPACE into free and blocked cells recursively starting with one large cell that is considered to be mixed, free, and blocked space. The CSPACE can be broken down using different space partitioning methods, such as quad-tree, oct-tree, and binary spacing partitioning. The main difference between these spacing partitioning methods is the number of partitions created. The quad-tree creates four, oct-tree eight, and binary two. As you can imagine, the partitions could be made simply by slicing one cell into n equal pieces. Although this may be efficient, it does not use useful information that could be beneficial for path planning. Entropy can be used to intelligently select partition points that generate better partitions and smaller trees than standard partitioning methods [Baltes and Anderson, 2003]. The information gain is determined by the relative proportion of free and blocked areas. Once the free cells are known, a planner can use the tree output from the flexible BSP algorithm to generate a collision-free path. A major disadvantage of BSP is that it does not directly facilitate the CSPACE representation. In order to generate CFREE, configurations would have to be generated that lie within the free cells. For this reason, this approach would not be suitable for robots with large DOF and real-time requirements. My thesis work, like flexible BSP, attempts to make use of information

about the world to make better decisions during the motion planning process.

3.5 Summary

Various methodologies that attempt to solve the robot motion planning problem have been presented in this chapter. One major advancement in robot motion planning has not yet been discussed. The Rapidly Exploring Random Tree (RRT) and variants thereof are discussed in the next chapter in much more detail than the related work presented above, since the RRT is the basis of the my solution strategy.

Chapter 4

Rapidly Exploring Random Tree (RRT)

Lavalle [1998] introduced the Rapidly Exploring Random Tree (RRT), a data structure for planning problems with non-holonomic and differential constraints. A RRT is a tree composed of quasi uniformly distributed random configurations rooted at the initial configuration of the robot. The random configurations in the tree are only seemingly uniformly distributed because they are typically generated with pseudo Random Number Generators (RNGs). Use of a real RNG would make the random configurations truly uniformly distributed. An example of a RRT and its evolution is shown in Figure 4.1. In the example, the initial configuration (blue), goal configuration (green), random configurations (red), and configurations on a path from the initial to goal configuration (yellow) are depicted. In a tree, every node except the

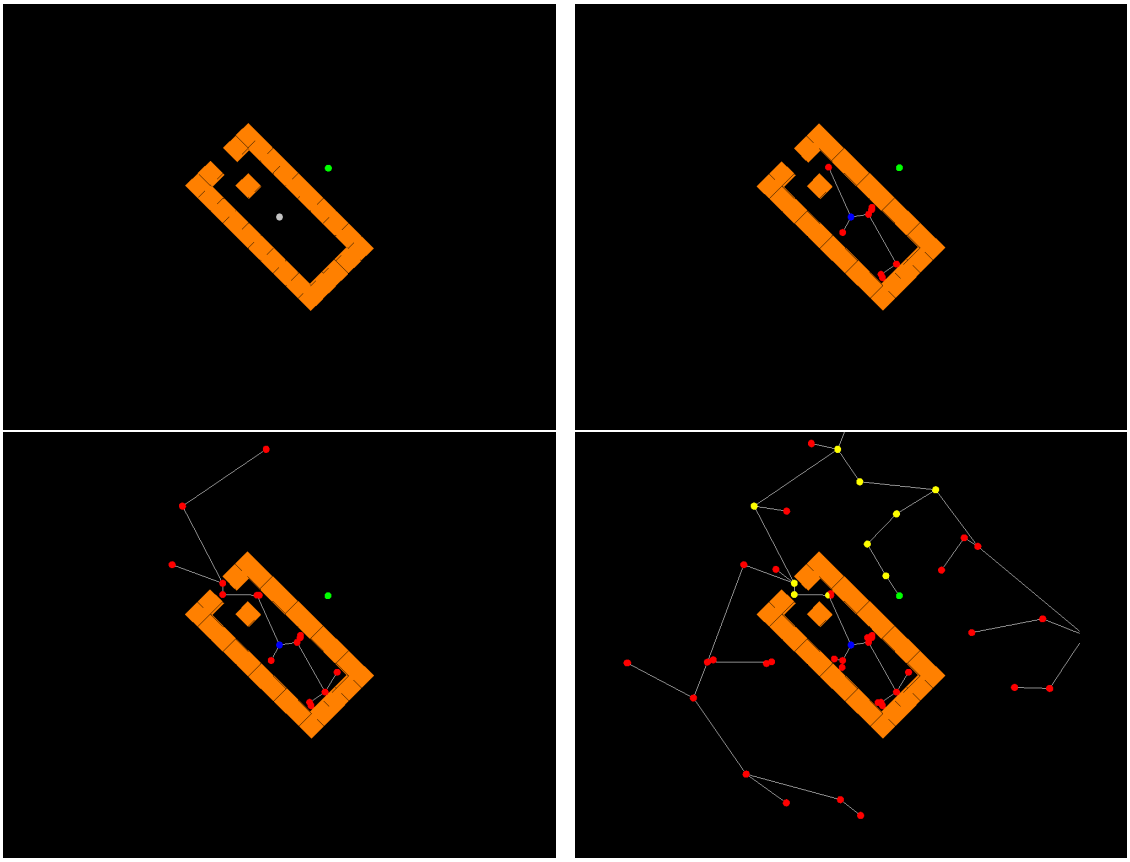


Figure 4.1: Rapidly Exploring Random Tree (RRT)

root has exactly one parent node (see Section 2.14.2 for more details). The random configurations in a RRT are connected in this fashion. Each random configuration or node in the RRT belongs to the CFREE set. Every edge between nodes in a RRT represents a path which also belongs to the CFREE set. The goal is to generate a leaf node which is the goal configuration.

The RRT data structure itself can be used as an algorithm for motion planning

because the data structure translates directly to the CSPACE state space representation. A path in the tree from the root (initial configuration) to leaf (goal configuration) is a motion plan. The RRT algorithm terminates once the goal configuration is achieved, therefore no unnecessary work is done. A RRT does not require a learning phase like the PRM described in Section 3.2. Only a single phase or query is required to determine a motion plan. This makes the RRT data structure a viable option for robot motion planning in dynamic environments with real-time requirements.

Variants of the RRT data structure as a motion planner, such as RRT-CONNECT has been shown to be probabilistically complete [Lavalle and Kuffner Jr., 2000]. However Karaman and Frazzoli [2011] proved that the probability of the classic RRT construction algorithm converging to an optimal solution is in fact zero. When a problem has a solution, a probabilistically complete algorithm's probability of finding a solution goes to one as the runtime approaches infinity [Berenson and Srinivasa, 2010]. In other words, the algorithm will eventually converge to a solution. In this thesis, the goal is not necessarily to find the optimal solution in terms of a heuristic function. The main concern is finding a sub-optimal solution that is good enough as quickly as possible. Karaman et al. [2011] suggest another variant of the RRT which is discussed in Section 4.5.9 that guarantees the optimal solution will be found if a solution exists.

Constructing a RRT requires a few fundamental components. At each iteration of the construction algorithm a random configuration is generated that can potentially

be used as a step in the motion plan. It is important that the RRT explore unexplored areas. To do this there must be a method to bias random configurations towards the unexplored areas (sample bias method). Each random configuration is connected to the existing tree by selecting the nearest configuration in the tree (NN method). To do this a distance metric d such as Euclidean distance must be defined to measure the distance between configurations (distance metric method). It is also important that the path between two configurations lies completely in CFREE or in order words is collision free (collision detection method). Pseudocode with comments for a RRT in the context of a CSPACE is given in Algorithm 5 [Lavalle, 1998].

In Algorithm 5, the first step of building the RRT is to set the root of the tree. The root of the tree is the robot's initial configuration. This is the simplest step in constructing the RRT. The next step is to start adding random configurations to the tree. This step could be repeated until the goal configuration is connected to the tree, however this could potentially cause the algorithm to never terminate if no solution exists. A solution to infinite execution time is to only repeat the add random configuration step n times. Care must be taken in selecting n to prevent premature termination of the algorithm when a solution does exist. You could choose n based on the maximum time you wish the RRT construction to take. Adding random configurations to the tree is done in a few steps: first the random configuration is generated. If the random configuration should be biased, then a configuration that is biased is returned. The sample bias method is discussed in more detail in Section 4.1.

Generation of the random configuration is robot dependent. For example, if a robot has 3 DOF then a configuration consists of the joint angles for each of the three joints. To generate a random configuration for this robot a random joint angle within the angle limits is selected for each joint. Once the random configuration is generated, the NN in current tree is found using the NN and distance metric method. The NN method is discussed in more detail in Section 4.2 and the distance metric method in Section 4.3. For the first random configuration, the NN is obviously the robot's initial configuration since this is the only other configuration in the tree at the time. After the NN is found, a move is attempted from the NN to the random configuration. One of three things may occur during the attempted move between the two configurations. Either the move is blocked by an obstacle, which is determined by using the collision detection method, or the move is not physically possible with the robot, or the move can be performed by the robot. The collision detection method is discussed in more detail in Section 4.4. If the move can be performed by the robot then the random configuration is simply added to the tree by connecting it to the NN. If the move is not possible then a new configuration that was generated by the movement up until it could no longer get any closer to the random configuration is added to the tree by connecting it to the NN.

RRTs share many of the same benefits as PRMs, such as being easy to implement, but also have many notable advantages of their own. For example, expansion of the RRT is biased towards unexplored regions of state space and with no sample bias

Algorithm 5 Classic RRT [Lavalle, 1998]

1: **procedure** BUILD($init_c, n, \delta t$)
2: $init_c \leftarrow RRT.init()$ ▷ Robot initial configuration.
3: **for** $i \leftarrow 0, n$ **do**
4: $rand_c \leftarrow RandomConfiguration()$ ▷ Uses sample bias method.
5: $near_c \leftarrow NearestNeighbor(rand_c, RRT)$ ▷ Uses kd-tree.
6: $motion \leftarrow Move(rand_c, near_c)$ ▷ Motion that minimizes distance between two configurations.
7: $new_c \leftarrow NewConfiguration(near_c, motion, \delta t)$ ▷ Configuration generated by motion from $near_c$.
8: $RRT.addConfiguration(new_c)$ ▷ Adds configuration to the tree.
9: $RRT.addEdge(near_c, new_c, motion)$ ▷ Path between two configurations when motion is performed.
10: **end for**
11: **return** RRT
12: **end procedure**

method the distribution of nodes is consistently close to a uniform sampling distribution which leads to consistent behaviour. With a uniform sampling distribution, the probability of selecting any particular configuration is the same. The RRT is always connected and has minimal edges because a configuration is only added to the tree if it can be connected to an existing configuration in the tree. An RRT is

not a map. A good map requires all valid paths to be defined in the map. Since the RRT does not contain all valid paths, one of the most notable advantages of the RRT is the fact that it does not require connections between all configuration pairs. Making connections between configuration pairs is an expensive operation in building a PRM [Lavalle, 1998].

4.1 Sample Bias

The sampling method of an RRT generates a random configuration in the CSPACE. Sampling refers to selection of a subset of configurations from all possible configurations (population). With no sample bias in the sampling method, the classic RRT with uniform sampling has an inherent Voronoi bias [Lavalle and Kuffner Jr., 2000; Lindemann and LaValle, 2004]. The concept of Voronoi diagrams was introduced in Section 2.13, recall that a Voronoi cell contains every point in which that distance from the center of the cell is less than any other cell. To understand why the classic implementation of an RRT has a Voronoi sample bias, consider how the RRT is expanded: when a random configuration is generated it is connected to the tree by selecting the NN. The NN selected can be considered the center of a Voronoi cell. Larger Voronoi cells occur on the “frontier" of the tree, and so larger cells are more likely to be selected for expansion. In a RRT configurations with larger Voronoi regions are more likely to be chosen. While a Voronoi bias is good for exploration of the CSPACE it does not help the RRT converge to the goal configuration quickly

while still allowing the RRT to branch to unexplored regions.

In order to help the RRT converge to the goal configuration faster, a bias to the sampling method can be introduced. The sample bias method is applied to the sampling method with probability n and is so called the *bias probability*. The bias probability chosen can affect the performance of the RRT construction algorithm. For example by creating cases where the RRT construction algorithm gets trapped in local minima. As stated in Section 4.5.1 one type of sample bias could be to choose the goal configuration based on the bias probability.

A near goal sample bias method could be employed that selects a configuration near the with probability n. A radius can be used for the randomly chosen biased configurations such that they are constrained within a sphere as described in Section 5.3.1.

A sort of counter-intuitive sample bias method is to select a configuration in COBS based on the bias probability as stated in Section 4.5.10. It would seem common sense to always pick configurations in CFREE, but exploration around obstacles can enable the motion planner to find paths when a robot must travel close to the obstacles in order to reach the goal configuration.

Another method to bias the samples is to use k-NN instead of just the NN. A move would be attempted from each of the k-NN to the random configuration and the new configuration with the best quality would be selected [Urmson and Simmons, 2003]. This type of sample bias method requires another heuristic to determine which

new configuration is the best. The sample bias is no longer completely Voronoi-like since a specific configuration is not continuously selected for expansion. Instead a region or k configurations can be selected for expansion.

The type of sample bias chosen may be domain specific. Different sample bias methods may perform better than others for specific problems. If your motion planner can be tailored for a specific domain and does not need to handle a wide variety of problems, then domain specific knowledge can be used in the sample bias method.

4.2 Nearest Neighbor (NN)

The NN method for finding the nearest configuration in the RRT to random configurations that are generated is as described in Section 2.12. The NN method is tightly coupled to the distance metric discussed in Section 4.3. Careful choice of a distance metric method is critical in how effective and fast the NN method is. For example a distance metric method that is computationally expensive will drastically impact the performance of the NN method since the distance metric method is executed many times throughout a NN query. Information about the configuration for the specific state space could be used to filter what data is relevant to the distance metric.

4.3 Distance Metric

The Distance Metric is used to determine which configurations are close to one another. How the “closeness" of configurations is defined can vary for different robots and domains. For example, it could be defined as how far the tool positions are from one another, or how much each of the joint angles differ, or how much power it takes to move from one configuration to another, etc. The distance metric used can be tailored for the robot and domain. All that matters is that the distance metric equates to a measurable quantitative value.

The following are some potential distance metric functions. As stated already, the inputs to the functions can vary. The inputs could be Cartesian coordinates, joint angles, power consumption, etc.:

Euclidean distance, derived from Pythagoras' formula, is the straight line distance between two points. Euclidean distance can be used for n-dimensional spaces. The general formula is as follows [Cha, 2007]:

$$d(a,b) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2} \tag{4.1}$$

Squared Euclidean distance in Equation 4.2 [Deza and Deza, 2009] is the same as Euclidean distance in Equation 4.1 but the square root is omitted for execution

time performance gains.

$$d(a,b) = \sum_{i=1}^{n} (a_i - b_i)^2 \tag{4.2}$$

Manhattan distance, also known as city block or taxicab distance, is the shortest distance between two points if one could only move in through a grid layout. The Manhattan distance can also be thought of as the sum of absolute error. Manhattan distance can be used for n-dimensional spaces. The general formula is as follows [Cha, 2007]:

$$d(a,b) = \sum_{i=1}^{n} |a_i - b_i| \tag{4.3}$$

Implementation of Equation 4.1 for 3D distances on a computer can be done using an approximation that reduces the execution time. The implementation is called a fast approximation to 3D Euclidean distance [Ritter, 1990]. The implementation uses fixed point arithmetic and there are no multiplications or divisions at all. Pseudocode for the implementation is given in Algorithm 6. The maximum error of the approximation from the real Euclidean distance is +/- 13%.

Implementation of Equation 4.1 for 2D distances on a computer can be approximated on a octagon to reduce the execution time [Ritter, 1990]. Pseudocode for the implementation is given in Algorithm 7. The approximation is always larger by at most 12% than the real Euclidean distance, the approximation is never smaller.

Algorithm 6 3D Euclidean Fast Approximation [Ritter, 1990]

1: **procedure** DISTANCE(dx, dy, dz)

▷ Convert reals to scaled integers

2: $maxc \leftarrow |dx \ll 10|$

3: $medc \leftarrow |dy \ll 10|$

4: $minc \leftarrow |dy \ll 10|$

▷ Sort

5: **if** $maxc < medc$ **then**

6: $swap(maxc, medc)$

7: **end if**

8: **if** $maxc < minc$ **then**

9: $swap(maxc, minc)$

10: **end if**

▷ Compute 1/4 of med & min in 1 step.

11: $medc \leftarrow medc + minc$

12: $maxc \leftarrow maxc + (maxc \gg 2)$

13: **return** $(real)(maxc \gg 10)$

14: **end procedure**

Algorithm 7 2D Euclidean Fast Approximation [Ritter, 1990]

```
1: procedure DISTANCE(x1, y1, x2, y2)
2:     ix ← x1 − x2
3:     iy ← y1 − y2
4:     t ← 0
5:     ix ← (ix < 0? − ix : ix)                          ▷ Absolute value
6:     iy ← (iy < 0? − iy : iy)                          ▷ Absolute value
                                                         ▷ Swap ix and iy
7:     if ix < iy then
8:         ix ← ix ^ iy
9:         iy ← iy ^ ix
10:        ix ← ix ^ iy
11:    end if
12:    t ← iy + (iy ≫ 1);
                                                         ▷ (123 * ix + 51 * iy) / 128
13:    return (ix − (ix ≫ 5) − (ix ≫ 7) + (t ≫ 2) + (t ≫ 6))
14: end procedure
```

4.4 Collision Detection

The collision detection method I employ in my approach is as described in Section 2.16. If a motion planner can be tailored for a specific domain and does not need to handle a wide variety of problems, then domain specific knowledge can be used to select an appropriate collision detection method. For example if the domain consists of simple objects then this may influence the type of bounding objects used by the collision detection method. This will in turn improve the speed of the collision detection method.

4.5 Related Work

Since the concept of the RRT was introduced there have been many adaptations of the RRT. In this section a few notable adaptations of the RRT are presented and discussed.

4.5.1 RRT-Goal Bias

The RRT-Goal Bias adaptation modifies Algorithm 5 above to introduce a different sample bias and thereby changes the inherent Voronoi bias. The sample bias method is to choose the goal configuration with probability n called the bias probability. With a small bias probability the RRT converges to the goal configuration much faster than the classic RRT [Lavalle and Kuffner Jr., 2000]. Too large of a bias

probability can lead to the RRT getting trapped in local minima so care must be taken in selection of the bias probability.

4.5.2 RRT-Goal Zoom

The RRT-Goal Zoom adaptation modifies Algorithm 5 above to introduce a different sample bias method and also modifies the sampling range. Instead of using the entire CSPACE as the sampling range, a region around goal is used as the sampling range with probability n. The size of the region around the goal is controlled by the current closest RRT configuration. The number of samples around the goal increases as the RRT gets closer to the goal [Lavalle and Kuffner Jr., 2000]. This is similar to the incremental sample range idea proposed in Section 5.3.1. The RRT-Goal Zoom can suffer from the same local minima problem as the RRT-Goal Bias and the Randomized Potential Field planner discussed in Section 3.1.

4.5.3 RRT-EXTEND

The RRT-EXTEND algorithm is given in Algorithm 8 [Kuffner Jr. and Lavalle, 2000; Lavalle and Kuffner Jr., 2000]. The EXTEND algorithm differs from the classic RRT in that the move towards the random configuration from the NN is performed for n time steps, or until the NN is reached, or an obstacle is blocking the path.

Algorithm 8 RRT-EXTEND 8 [Kuffner Jr. and Lavalle, 2000; Lavalle and Kuffner Jr., 2000]

1: **procedure** BUILD($init_c, n, \delta t$)
2: $init_c \leftarrow RRT.init()$ ▷ Robot initial configuration.
3: **for** $i \leftarrow 0, n$ **do**
4: $rand_c \leftarrow RandomConfiguration()$ ▷ Uses sample bias method.
5: $EXTEND(RRT, rand_c)$
6: **end for**
7: **return** RRT
8: **end procedure**

Algorithm 9 EXTEND 8 [Kuffner Jr. and Lavalle, 2000; Lavalle and Kuffner Jr., 2000]

1: **procedure** EXTEND($RRT, rand_c$)
2: $near_c \leftarrow NearestNeighbor(rand_c, RRT)$ ▷ Uses kd-tree.
3: $motion \leftarrow Move(rand_c, near_c)$ ▷ Motion that minimizes distance between two configurations.
4: $new_c \leftarrow NewConfiguration(near_c, motion, \delta t)$ ▷ Configuration generated by motion from $near_c$.
5: **if** $new_c \neq near_c$ **then**
6: $RRT.addConfiguration(new_c)$ ▷ Adds configuration to the tree.
7: $RRT.addEdge(near_c, new_c, motion)$ ▷ Path between two configurations when motion is performed.
8: **if** $new_c = rand_c$ **then**
9: **return** $REACHED$ ▷ If new_c and rand_c are equal then a movement can be performed between the two configurations.
10: **else**
11: **return** $ADVANCED$ ▷ If new_c and rand_c are not equal then a partial movement can be performed between the two configurations.
12: **end if**
13: **else**
14: **return** $TRAPPED$ ▷ If new_c and near_c are equal then a movement could not be performed, therefore the path between the configurations is blocked.
15: **end if**
16: **return** RRT
17: **end procedure**

Algorithm 10 RRT-CONNECT 8 [Kuffner Jr. and Lavalle, 2000; Lavalle and Kuffner Jr., 2000]

1: **procedure** BUILD($init_c, n, \delta t$)
2: $init_c \leftarrow RRT.init()$ ▷ Robot initial configuration.
3: **for** $i \leftarrow 0, n$ **do**
4: $rand_c \leftarrow RandomConfiguration()$ ▷ Uses sample bias method.
5: $CONNECT(RRT, rand_c)$
6: **end for**
7: **return** RRT
8: **end procedure**

Algorithm 11 CONNECT 8 [Kuffner Jr. and Lavalle, 2000; Lavalle and Kuffner Jr., 2000]

1: **procedure** CONNECT($RRT, rand_c$)
2: **repeat**
3: $result \leftarrow EXTEND(RRT, rand_c)$
4: **until** $result \neq ADVANCED$
5: **return** $result$
6: **end procedure**

4.5.4 RRT-CONNECT

The RRT-CONNECT algorithm is given in Algorithm 10 [Kuffner Jr. and Lavalle, 2000; Lavalle and Kuffner Jr., 2000]. The CONNECT algorithm is similar to the EXTEND algorithm but there is no restriction of n time steps. Without this restriction the CONNECT algorithm can generate longer paths with fewer calls to the NN method. CONNECT works best for holonomic planning problems while EXTEND works best for non-holonomic planning problems.

Many interesting examples in simulation are presented by Kuffner Jr. and Lavalle [2000] such as a docking maneuver for a fully orientable satellite model, piano moving, a Programmable Universal Machine for Assembly (PUMA) 6 DOF robotic arm, a human finding and using a hammer in a virtual world, and playing a game of virtual chess. No real world examples were presented.

4.5.5 RRT-Bi-directional

The concept behind RRT-Bi-directional, as the name implies is to grow two trees [Lavalle and Kuffner Jr., 2000]. One tree is rooted at the initial configuration of the robot and the other from the goal configuration. At every iteration an attempt is made to join the two trees. If the two trees are joined then the RRT halts since this means a solution has been found. The RRT-Bi-directional is analogous to a bi-directional search of a list. Variants of RRT-Bi-directional algorithm based on RRT-EXTEND and RRT-CONNECT were presented. A version where the two trees can

use varying EXTEND/CONNECT methods was also given. An example of where the RRT-Bi-directional algorithm may be advantageous is when you have a robotic manipulator where not only are the final position and orientation are important but the exact arm configuration must be specified for the goal configuration. Starting a tree from the goal configuration ensures that the part of the plan near the goal configuration meets the criteria to maintain the specified arm configuration, whereas if one were to only start from the initial configuration the random configurations generated near the goal may not meet the criteria of the final arm configuration.

4.5.6 Jacobian Transpose RRT (JT-RRT)

As stated in Section 2.8.3, there is no general closed form solution to the IK problem for robot manipulators with greater than 6 DOF. Recall from Section 2.8.3.1 that the Jacobian is a matrix of partial derivatives that specify how each joint affect the tool position and orientation. The algorithm presented by Vandeweghe et al. [2007] uses the Jacobian Transpose (JT) to bias the tree growth once a random configuration is within a specified distance of the goal configuration. The time steps are kept small since the JT represents instantaneous change of the tool. The solution was demonstrated with a 7 DOF manipulator. As a result of combining the JT with the RRT, the motion plan generated also produces viable numerical solution to the IK problem without the need to analytically solve the IK for the manipulator.

4.5.7 Multipartite RRT (MP-RRT)

The Multipartite RRT (MP-RRT) introduced by Zucker et al. [2007] is geared towards environments with dynamic obstacles. Instead of throwing away useful information and completely re-planning whenever the environment changes, parts of the previous valid RRT are kept. The MP-RRT maintains a forest of disconnected sub-trees that all lie in CFREE which are RRTs that became disconnected as the environment changed since invalid configurations and edges are pruned. These disconnected sub-trees are not connected to the initial configuration. When the next plan is required an RRT is built with a sample bias method that attempts to connect the root of one of the disconnected sub-trees. By re-using the old disconnected RRTs that are still mostly valid unless the environment changes drastically, a considerable amount of time is saved when building a new RRT that is valid for the current state of the environment.

4.5.8 RRT-Blossom

The RRT-Blossom adaptation is tailored for highly constrained environments such as those with narrow passages [Kalisiak and van de Panne, 2006]. RRT-Blossom expands on and improves the RRT-Collision Tendency (CT) variant. The RRT-CT variant keeps track of failed move attempts (edges) in the tree. This information is used to prevent redundant failures. It is also used in the sample bias method by allowing selection of configurations with lower “collision tendency". RRT-Blossom

adds a local receding but non-regressing flood fill mechanism into the RRT construction algorithm [Kalisiak and van de Panne, 2006]. What this means is that expansion can move further (recede) from the goal configuration but prevents exploration of space already explored (regression). The local flood fill part refers to how individual configurations are expanded to aid in exploration of the space near to the configuration.

4.5.9 RRT*

Karaman and Frazzoli [2011] proved that although the classic RRT construction algorithm is probabilistically complete, the probability of classic RRT construction algorithm converging to an optimal solution is zero. Karaman [Karaman and Frazzoli, 2011; Karaman et al., 2011] proposed a variant RRT* that exhibits *asymptotic optimality* and probabilistic completeness. Asymptotic optimality provides almost-sure convergence to the optimal solution. The RRT* achieves asymptotic optimality by refining the motion plan during construction of the RRT.

To understand how RRT* differs from the classic RRT construction algorithm given in Algorithm 5. Consider how the edges in the tree are created. If a move is possible from the NN to the random configuration, then an edge is added to the RRT with the NN as the parent of the random configuration. Similarly if a move is not possible from the NN directly to the random configuration, then an edge is added to the RRT with the NN as the parent of the configuration generated by the movement.

In the RRT*, instead of simply creating this edge a heuristic function is used to pick the best of the k-NNs to determine the parent. Karaman et al. [2011] use the cost of the trajectory as the heuristic function. In addition to this, configurations in the tree that are near to the new configuration added to the tree are evaluated for cost to determine if their parents should be changed to the new configuration. This is optimization is called re-wiring by Karaman et al. [2011].

4.5.10 Obstacle-Based RRT (OB-RRT)

Contrary to the RRT-Goal Bias construction algorithm, which introduces a sample bias method that chooses the goal configuration, Obstacle-Based RRT (OB-RRT) uses information about the obstacles in the sample bias method [Rodriguez et al., 2006]. An interesting idea presented by Rodriguez et al. [2006] is the use of a fixed orientation for the robot when moving through narrow passages. If the movement through a narrow passage can be completed with a simple sliding motion such as a translation, then using a fixed orientation greatly simplifies the problem. Variable orientation is only necessary when complex movements are required to navigate narrow passages. Rodriguez et al. [2006] suggest many different obstacle based and non-obstacle based sample bias methods as listed below:

- Random robot position and same orientation as the NN.
- Random obstacle position and random orientation.

- Random obstacle position and same orientation as the NN.
- Rotation followed by extension. The NN is rotated to align with the random configuration first, then a translation is performed until the random configuration is reached or a collision occurs.
- Trace obstacle with random orientation. A random configuration is first attempted to be added to the tree with the class RRT construction algorithm. If an obstacle is hit basic then the NN is extended in the direction of the obstacle with a random orientation.
- Trace obstacle with same orientation. This is the same as above but the orientation of the NN is used instead of a random orientation.
- Trace CSPACE obstacle. The obstacle boundary is calculated by ray tracing instead of relying on the collision detection method.
- Medial axis push. The medial axis provides a compact representation of shapes [Dey and Zhao, 2003]. The medial axis push method is the same as trace obstacle with random orientation except, after pushing towards the obstacle the configuration is pushed back towards the medial axis of the CSPACE. This in affect moves the configuration away from COBS back closer to CFREE.

4.5.11 Task Space RRT (TSPACE-RRT)

The Task Space RRT (TSPACE-RRT), as the name implies, builds the RRT in the TSPACE instead of CSPACE [Shkolnik and Tedrake, 2009]. For more information on the TSPACE see Section 2.2. By using the TSPACE, construction of the RRT occurs in a lower dimensional space since the TSPACE is a subset of the CSPACE. Performance gains can be realized for high dimensional CSPACEs. For example, instead of considering the joint angles of the whole robot one may only consider the hand or arm. Generating random configurations in the CSPACE allows for probabilistic completeness while sampling in TSPACE allows for faster more direct exploration.

4.5.12 Closed-Loop RRT (CL-RRT)

The Closed-Loop RRT (CL-RRT) is a variant of RRT that uses a path-tracking control loop [Luders et al., 2010]. CL-RRT was demonstrated on a vehicle at the Defense Advanced Research Projects Agency (DARPA) DARPA Urban Challenge (DUC). The CL-RRT approach maintains two trees. One tree represents the simulated trajectory and the second tree represents the actual trajectory achieved. The goal of the control loop is try to match the actual trajectory to the simulated trajectory since the vehicle will not always track the path as expected.

4.5.13 Particle RRT (pRRT)

Particle RRT (pRRT) attempts to address the problem of planning with uncertainty [Melchior and Simmons, 2007]. Uncertainty can come from many places such as unreliable sensor feedback or inconsistent robot movements. The pRRT approach builds on existing work done with *particle filters* [Arulampalam et al., 2002]. A particle filter contains many particles, each of which is a weighted estimate of the agent's pose. After an action (e.g. a left, right, forward, or backward rotation or translation) the pose estimate of each particle is updated based on the motion model, then the weights are updated based on the sensor feedback [Bagot et al., 2008]. The best particle is the weighted average of all particles.

The pRRT approach is similar to the particle filter in that when adding random configurations to the tree, multiple likely conditions are simulated. The likelihood of reaching a configuration can be calculated similar to the best particle as described above. This likelihood can then be used in the sample bias method to favour paths that the robot is more likely to achieve.

4.5.14 Heuristically-guided RRT (hRRT)

The classic RRT construction algorithm does not take into account the cost of the motion plan. Since the cost of the motion plan is not considered, motion plans extracted from the RRT are typically sub-optimal. The Heuristically-guided RRT (hRRT) attempts to bridge this gap by introducing a heuristic function based

on the quality of the motion plan and the size of the Voronoi region [Urmson and Simmons, 2003]. The Voronoi bias from the classic RRT construction algorithm is not completely removed because exploration of the CSPACE is still desirable. Although the heuristically-guided technique worked well in the experiments performed by Urmson and Simmons [2003], it still suffered from undesirable behaviour caused by large Voronoi regions in narrow passages. As a solution to this problem, Urmson and Simmons [2003] suggested using the k-NNs to allow smaller Voronoi regions with a better quality motion plan to be expanded instead of the larger Voronoi region. Urmson and Simmons [2003] also suggests that k can be iteratively increased until the quality measure of the motion plan meets reaches an acceptable level.

4.5.15 Exploring/Exploiting Tree (EET)

The Exploring/Exploiting Tree (EET) to function like a potential field, exploiting high potential areas in the environment while reaping the benefits of the RRT for exploration [Rickert et al., 2008]. The potential field like behaviour allows for exploitation of useful information. By mixing the potential field like behaviour with the RRT, the motion planner becomes probabilistically complete allowing for exploration when necessary. Rickert et al. [2008] argue that using some form of exploitation is the only way to minimize the required exploration of the state space in order to find a solution.

4.6 Other Applications

RRTs have been applied in some other fields unrelated to robot motion planning. These other applications are not readily apparent. One such example is for insertion of a flexible steerable needle into biopsy tissue to deliver treatment [Xu et al., 2008]. Given a target (injection site), Xu et al. [2008] are trying to find an entry point and path to the target. By using the RRT and flexible steerable needle, injection sites that were previously impossible to use are now achievable.

4.7 Summary

The basic RRT data structure, construction algorithm, and application to robotic motion planning were presented. The RRT is the basis of my solution strategy for robot motion planning. The next chapter will describe how my variant of RRT for robot motion planning is implemented, both in simulation and on a physical robot.

Chapter 5

Implementation

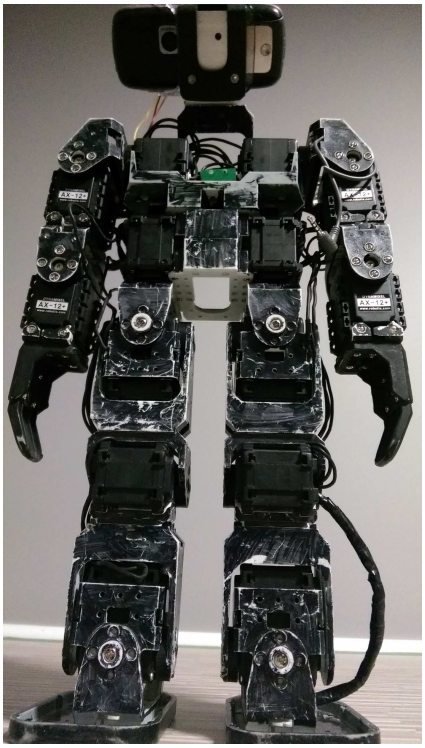
Having described the use of RRTs for robot motion planning, this chapter will cover the implementation of my approach to robot motion planning. The basis of my approach is the RRT. To ultimately evaluate the performance of my approach (Chapter 6), a decision must be made as to whether to use simulation or the real world. Simulation allows for more trials to be run with many more variations of robots and testing environments because they can be easily constructed, while a physical robot in the real world serves to ensure that the simulated model translates to actual performance. My work will be evaluated in both of these realms, and so the discussion of the implementation of my work is divided similarly.

5.1 Real World

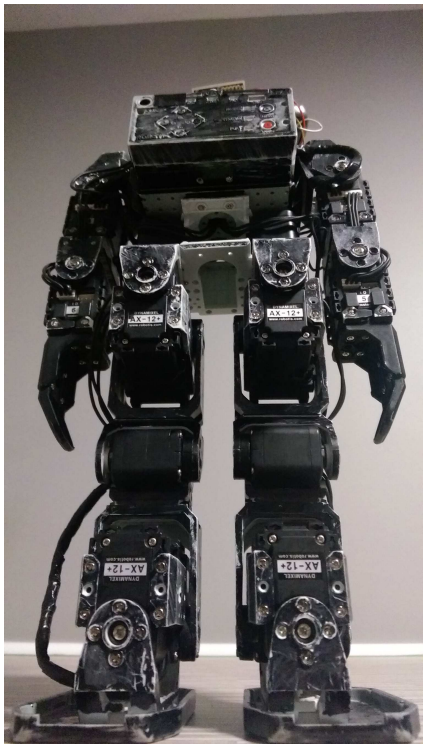
The purpose of performing some experiments in the real world is to demonstrate the accuracy of the world and robot models. Without verifying and validating the models in the real world, there is no way of knowing if the solution strategy will work in the real world or whether it only works in simulation.

5.1.1 Humanoid Robot

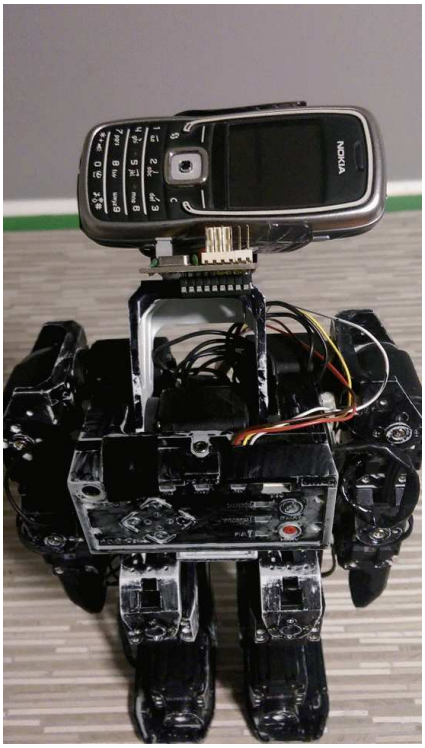
The humanoid robot I used to embody my work is shown in Figure 5.1. The humanoid robot is referred to as *Blitz* throughout the remainder of this thesis. Blitz is a custom modified robot based on Robotis' Biolod kit [Robotis, a,b] and has been used in both previous academic research [Bagot et al., 2008] and in many competitions, including Federation of International Robot-soccer Association (FIRA) 2007 HuroCup in San Francisco USA, and RoboCup 2007 in Atlanta USA, Euroby 2008 in Linz Austria, FIRA 2008 HuroCup in Qingdao China, and RoboCup 2008 in Suzhou China



(a) Blitz View 1



(b) Blitz View 2



(c) Blitz View 3

(d) Blitz View 4

Figure 5.1: Humanoid Robot - Blitz

This robot has nineteen DOF (three in each arm, five in each leg, two in the torso, and one in the neck) as shown in Figure 5.3. When describing the axis of motion [Pla, 2008] affected by a joint in a humanoid robot, it is useful to consider the human body cut into three planes, called the *cardinal planes* as shown in Figure 5.2. The first plane is called the lateral (saggital) plane which divides the body into the left and right halves. Motion in the lateral plane is forward and backwards. The second plane is called the frontal (coronal) plane, which divides the body into the front and back halves. Motion in the frontal plane is left and right. The third plane is called the traverse (horizontal) plane, which divides the body into the top and bottom halves. Motion is the traverse plane turns/twists the body. Motion in the planes can be depicted by rotation of the plane. For Blitz, one DOF in the arm effects motion in the lateral plane, while the other two effect motion in the frontal plane. Three DOF in the leg effect motion in the lateral plane and the other two effect motion in the frontal plane. Both DOF in the torso effect motion in the traverse plane. The DOF in the neck effects motion in the lateral plane.

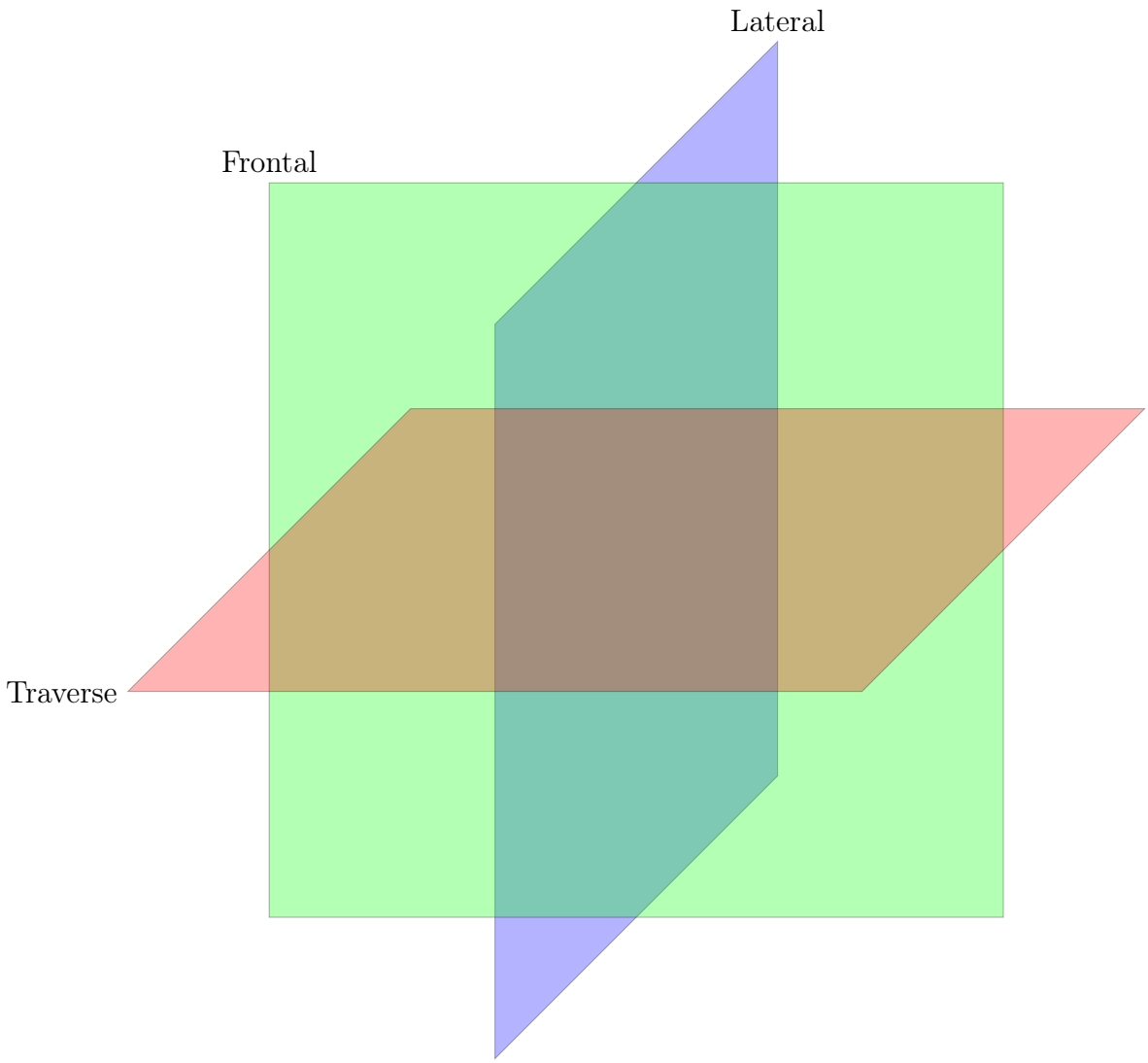


Figure 5.2: Cardinal Planes

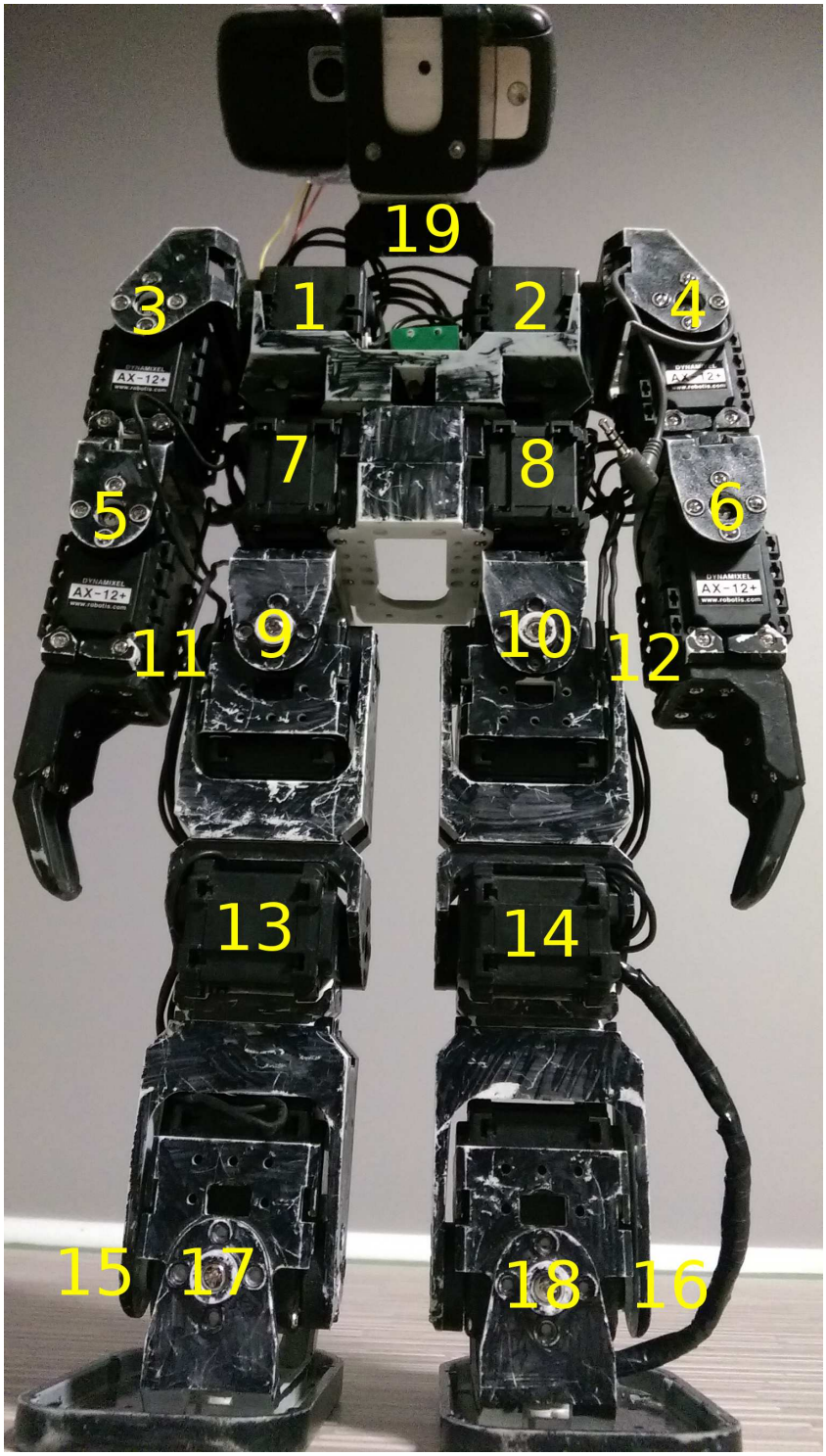


Figure 5.3: Blitz Joints with Servo ID

Each DOF in Blitz is articulated by a Dynamixel AX-12 servo [Robotis, 2006] which is capable of producing 15.29kg.cm of torque at 12V. The servos are daisy chained together using a serial bus. Table 5.1 lists the servos and their respective joint limits. Although all servos are equivalent, some joint angle limits are different due to the servo mounting position, orientation, proximity to other servos, and bracket. The joint angle limits imposed by the physical servo is shown in Figure 5.4. The servos would be considered under-powered at the present time compared to current small size humanoid robots that may have up to 101.97kg.cm of torque at 14.8V in extreme cases (e.g. if using a Robotis Dynamixel EX-106+ servo).

Table 5.1: Blitz Joints

Servo ID	Name	Min Angle	Max Angle
2	Left Shoulder Lateral	0	300
4	Left Shoulder Frontal	0	300
6	Left Elbow Frontal	0	300
1	Right Shoulder Lateral	0	300
3	Right Shoulder Frontal	0	300
5	Right Elbow Frontal	0	300
8	Left Torso Traverse	0	300
7	Right Torso Traverse	0	300

Table 5.1: Blitz Joints

Servo ID	Name	Min Angle	Max Angle
12	Left Hip Lateral	0	300
10	Left Hip Frontal	0	300
14	Left Knee Lateral	0	300
16	Left Ankle Lateral	0	300
18	Left Ankle Frontal	0	300
11	Right Hip Lateral	0	300
9	Right Hip Frontal	0	300
13	Right Knee Lateral	0	300
15	Right Ankle Lateral	0	300
17	Right Ankle Frontal	0	300
19	Neck Lateral	0	300

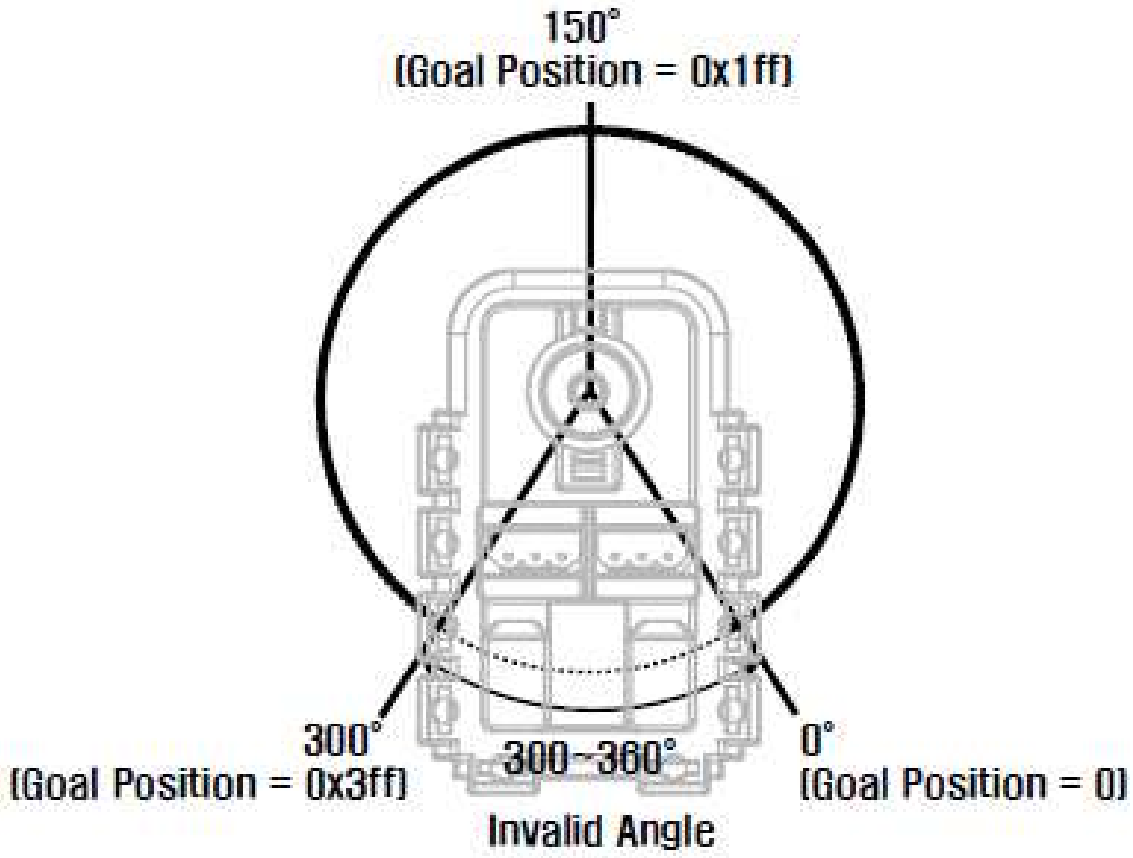


Figure 5.4: AX-12 valid angle range

(from Dynamixel AX-12 Manual Robotis [2006])

Blitz is equipped with an on-board ATMEL AVR ATmega128 micro-controller ATMEL [2006] and a Nokia 5500 cellular telephone (Figure 5.5), which are interfaced by a custom-made Infrared Data Association (IrDA) board containing a Microchip MCP2150 standard protocol stack controller supporting Data Terminal Equipment (DTE) applications. The on-board micro-controller is predominately used for communication with the servos, including but not limited to tasks such as target position

setting, trajectory planning by position interpolation, and servo load checking. The on-board micro-controller is also used for the storage and playback of static motions created by the Motion Editor as shown in Figure 5.6. This is all made possible by custom firmware running on a multi-threaded Real-Time Operating System (RTOS) code named Freezer OS [Baltes et al., 2011]. The micro-controller is situated in what Robotis calls the CM-5, which packages the micro-controller with interface items such as buttons and Light Emitting Diodes (LEDs).

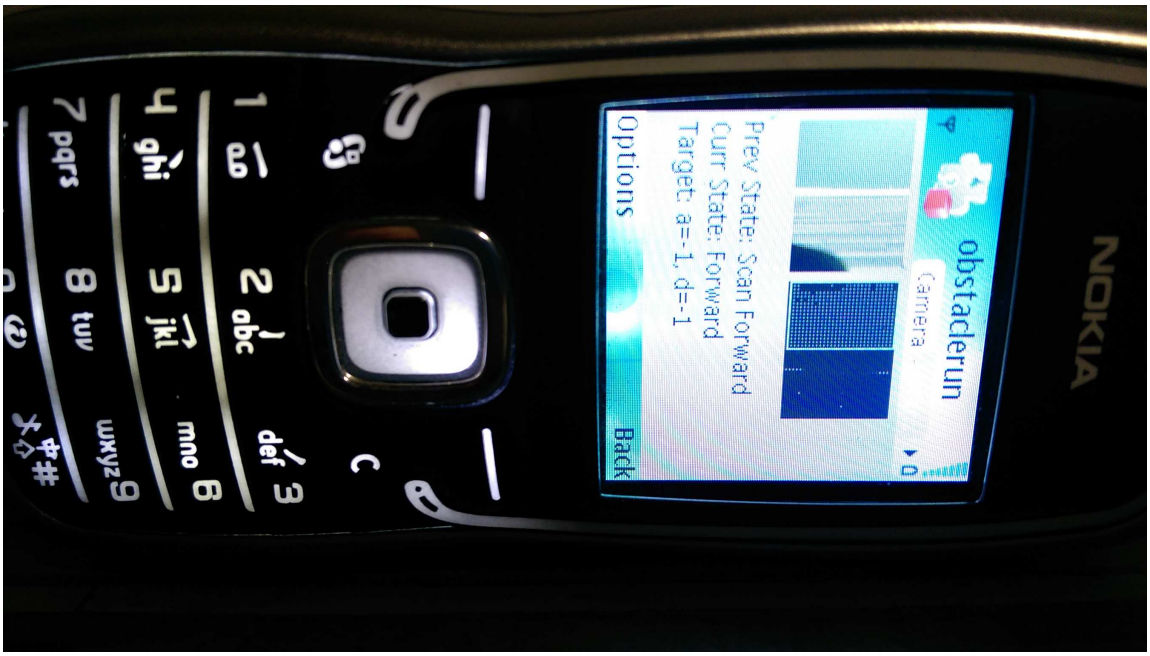


Figure 5.5: Nokia N5500

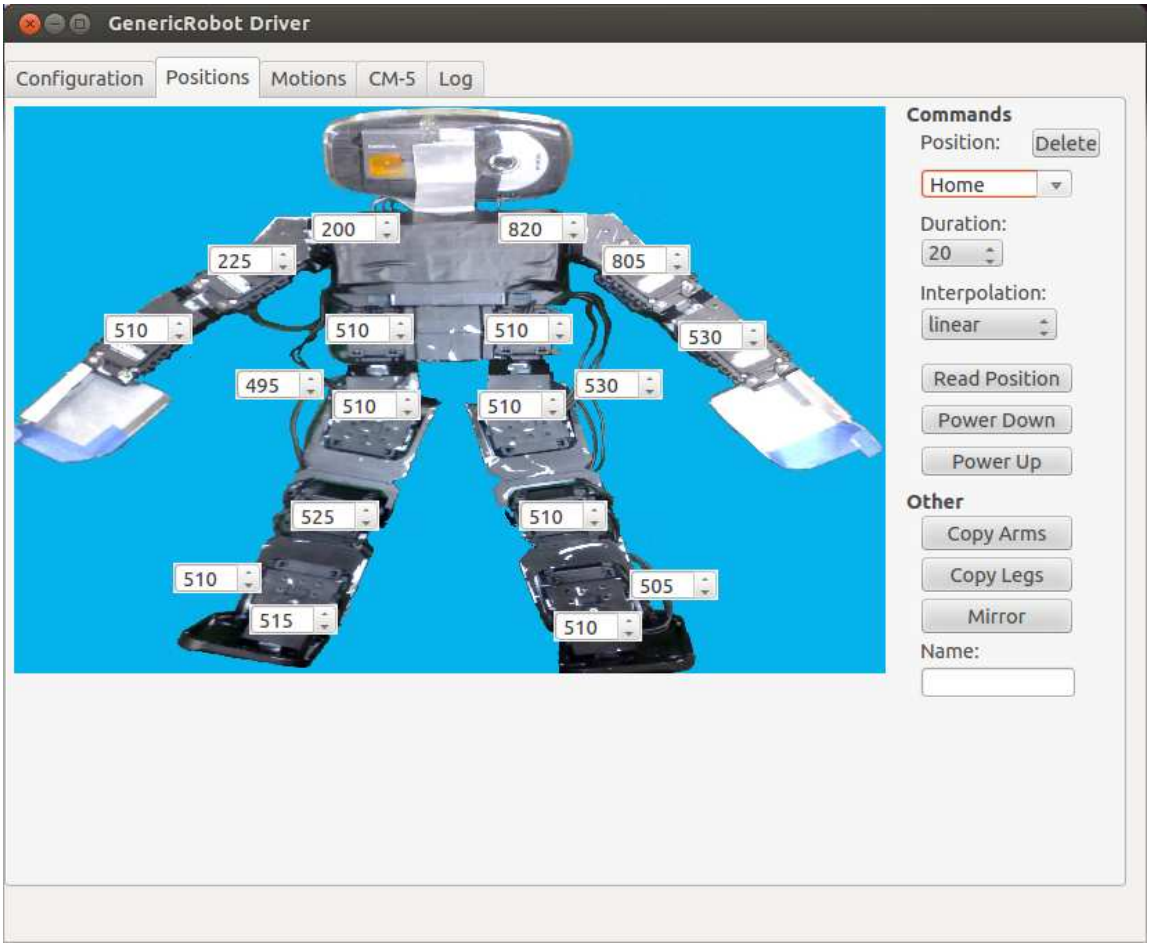


Figure 5.6: Motion Editor

The Nokia 5500 provides a full C++ development environment, robust operating system (SymbianOS 9.1 series 60 release 3.0), camera, communication mediums (Bluetooth and IrDA), an ARM 9 235MHz processor, and a three axis accelerometer (LIS302DL). The Nokia's processor is used for state generation, image processing, sensor data smoothing, localization, mapping and other various application pro-

grams. Everything is powered by a single lithium-ion polymer battery pack except the Nokia 5500, which has its own battery.

There are many complex modules working together on Blitz. The following sections describe how each of these modules work with the motion planner by providing inputs or performing tasks that the motion planner is not responsible for. Figure 5.7 shows a coarse FBD of the modules and their relation to one another.

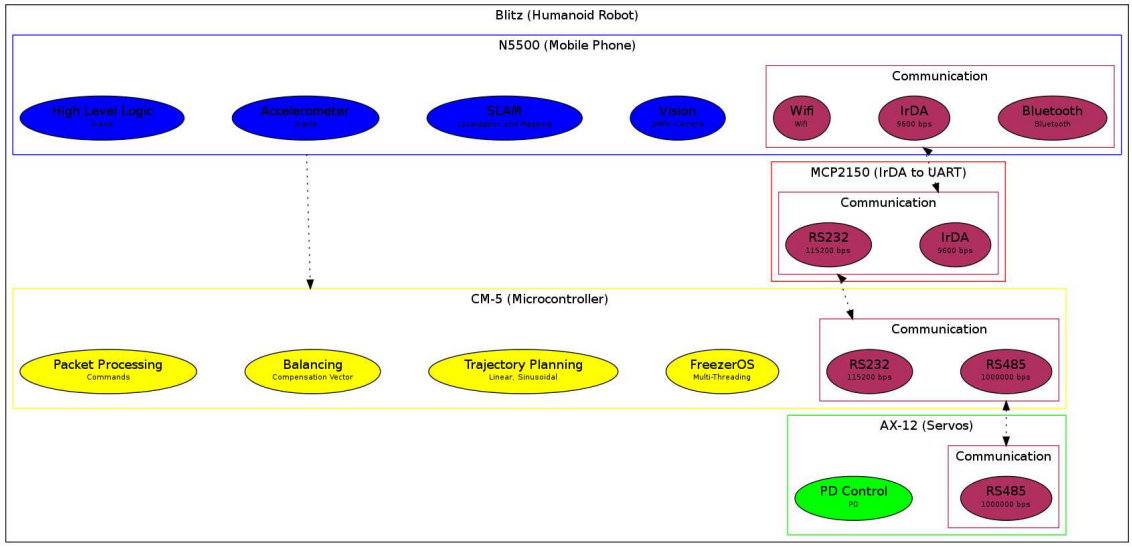


Figure 5.7: Blitz FBD

5.1.2 High Level Logic

The high level logic in Blitz is essentially where all the decision making occurs. The high level logic uses information from lower level modules and makes informed decisions as for what to do next. For example for a robotic soccer player, if a lower

level module such as a vision system detects the soccer ball then the high level logic may decide to move towards the soccer ball. For the motion planner the high level logic will specify the goal configuration, such as intercepting the soccer ball.

The high level logic is implemented as a decision making finite state machine using the concept of the Subsumption Architecture introduced by Brooks [1986]. The states in the finite state machine are organized in hierarchical manner. Parent states can subsume or suppress child states. Figure 5.8 is an example of the finite state machine for a soccer player penalty kick application. Perception is the topmost state, and can subsume or suppress all other states. When the robotic soccer player cannot find the soccer ball, scanning or random walks are performed. Once the soccer ball is found, the robotic soccer player is positioned such that the soccer ball is in reach to kick and oriented towards the target in the goal. Any of the movements required by the high level logic would be handed off to the motion planner to handle.

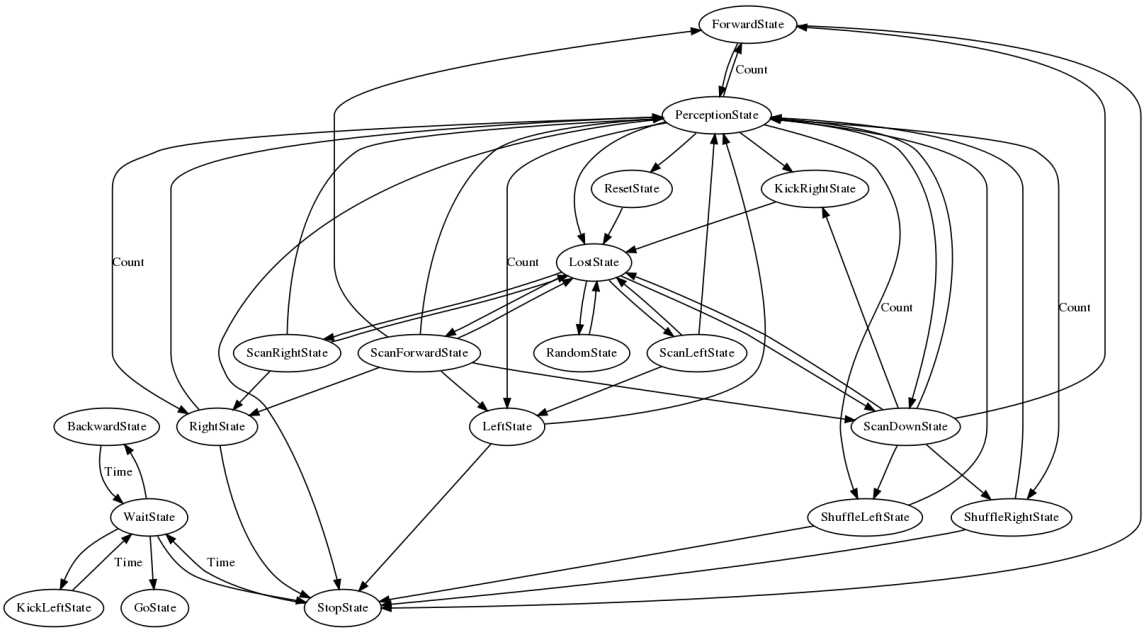


Figure 5.8: State Machine Diagram for Soccer Player Penalty Kick

5.1.3 Vision

The vision system in Blitz is responsible for detection of objects in the robot's line of sight. Object detection provides information about objects such as size, shape, colour, position, and orientation. The information regarding the objects is required by the motion planner for determining what is considered CFREE and what is considered COBS. Unfortunately since the position and orientation information provided by the vision system is relative to the robot because it is calculated by line of sight, a higher level above the vision system is required to determine the absolute position and orientation of the objects in the world. Determining absolute position

and orientation of the objects in the world is described in Section 5.1.4.

The Nokia N5500's camera is used for vision. The algorithm used to detect objects is a floodfill colour based blob detection. The algorithm is similar to the paint bucket tool in many popular image editors. A calibration application was written called SColour that allows for selection of colours that identify objects. For example, different colours can be calibrated for object types such as spheres and cubes. The floodfill algorithm uses the calibrated colours to extract objects from images and can provide feedback such as the 2D coordinates of the centroid of objects, orientation of the objects, and their bounding regions. The vision system is fairly primitive and noisy, but there exists other vision systems that can provide much more accurate feedback such as laser scanners or stereo vision systems.

5.1.4 Localization and Mapping

I developed a novel particle filter for Simultaneous Localization and Mapping (SLAM) in humanoid robots [Bagot et al., 2008], and this is employed in my implementation. The particle filter is used to maintain an estimate of the current robot pose (localization) as the robot moves throughout the world. The robot's estimated pose is used to supplement the relative coordinate input from the vision system. Using the robot's estimated pose, the relative coordinates are converted into absolute coordinates (mapping) which provide the position and orientation of the objects in the world. The motion planner can use the map to determine if the generated ran-

dom configurations are in CFREE or COBS and to perform collision detection when determining if a move is possible between two configurations.

5.1.5 Trajectory Planning

Trajectory planning in Blitz is used to handle the low level details of movements between two configurations. The motion planner determines what configurations to transition between in order to achieve the goal configuration. The motion planner provides input to the trajectory planner. However, it also requires either input from the trajectory planner or basic knowledge of how the trajectory planner works, in order to determine if a move between two configurations is feasible.

Movement between two configurations can be specified with three parameters. The first parameter, *delay*, specifies how long to remain at the initial configuration before starting a movement towards the goal configuration. The second parameter, *time*, is the desired amount of time that it should take to transition between the initial and goal configuration. The third parameter, *interpolation type*, in conjunction with time specifies how to interpolate between the initial and goal configuration. There are two types of interpolation implemented: simple Linear interpolation and Sinusoidal interpolation. Interpolation must be performed for each joint in order to reach configuration two in the desired time. A plot of the Sinusoidal interpolation is given in Figure 5.9.

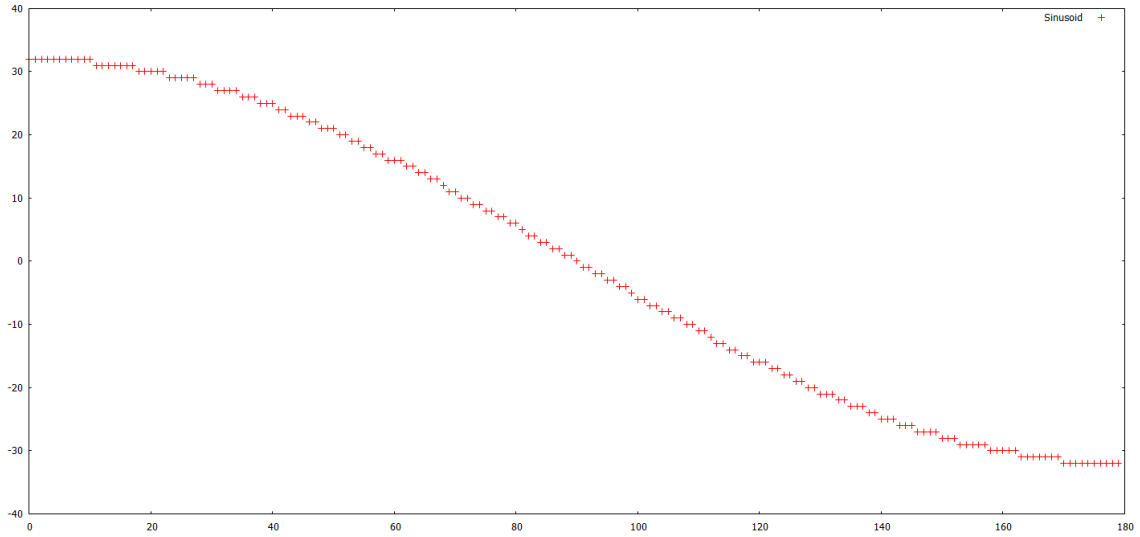


Figure 5.9: Sinusoid

5.1.6 Balancing

The walking gait of Blitz is a static walking gait, meaning it is a sequence of static motions. Static walking gaits are susceptible to any variation in the ground terrain. Variation in the ground terrain can cause the static walking gait to become unstable. Balancing on Blitz is achieved by sending compensation values during the static walking gait. The compensation values are determined by forecasting time series data provided by an accelerometer in the Nokia N5500. In addition to providing information to calculate the compensation values, the accelerometer is used to detect falls.

The forecasting technique used is called exponential smoothing. Exponential smoothing [Nis, 2007] is a form of time-series analysis used extensively in forecasting

(also known as prediction). The exponential portion of the name comes from the method in which data is weighted. As data ages in the time-series, the weight associated with it decreases exponentially. Types of exponential smoothing include single, double, and triple where are each is suited for time-series data without any trends, with trends, and with trends and seasonality, respectively. The equations for single and double exponential smoothing are given by Equations 5.1 and 5.2. The original value is y, while s is the smoothed value.

$$s_t = \alpha y_{t-1} + (1 - \alpha) s_{t-1} \tag{5.1}$$

$$s_t = \alpha y_t + (1 - \alpha)(s_{t-1} + b_{t-1}) b_t = \beta(s_t - s_{t-1}) + (1 - \beta) b_{t-1} \tag{5.2}$$

$$s_t = y_t + (y_t - y_{t-1}) \tag{5.3}$$

The reason for using a forecasting technique to predict accelerometer data is because of the latency incurred during receiving and processing the accelerometer data. The latency is too large to provide the required reaction time. Exponential smoothing provides some prediction and helps deal with noisy data, but the smoothing reduces the sensitivity of the balancing because important peaks where motions are becoming unstable can be remove by the smoothing. A potential solution to this problem is to accentuate peaks by using the rate of change as given in Equation 5.3.

Although the motion planner can discard configurations that are not dynamically stable based on POS, as described in Section 2.9, or by the simulated robot model and

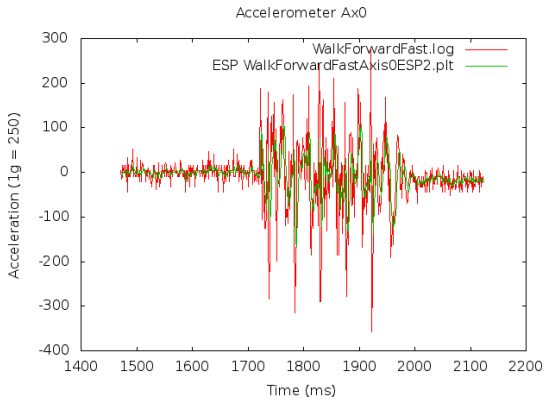
physics engine, this alone does not guarantee the robot will not fall over. Uncertainty in the ground plane consistency (uneven, friction, etc.) may cause some motions or transitions between motions to become unstable even though they were valid given the world and robot models in simulation. An additional level of optimization to the motions determined by the motion planner are required that can only be achieved in an ad-hoc manner such as the compensation values described.

Using the accelerometer data from the N5500, balancing is achieved by detecting if the robot is becoming unstable and calculating a compensation vector to counteract the source of the instability. Since the accelerometer is a 3-axis accelerometer, the g-force can be determined for each of the cardinal planes. For each plane, the compensation vector contains offsets for the main joints that effect motion in the corresponding plane. For example if instability is detected in the lateral plane, then the compensation vector will contain offsets for the main joints that effect motion in the lateral plane. The compensation vector changes the target joint angles that the trajectory planner is working with.

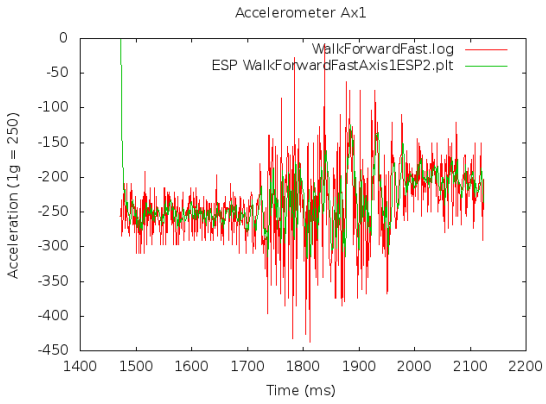
Instability is detected by thresholding the pre-processed accelerometer data. A PID controller (Section 2.11) is then used to calculate the offsets that are placed in the compensation vector. Plots of the accelerometer data and forecasting for each cardinal plane are given in Figure 5.10, 5.11, and 5.12.

There are a number of problems with this method that could easily be improved. First, since the accelerometer is on the N5500, there is an inherent latency issue

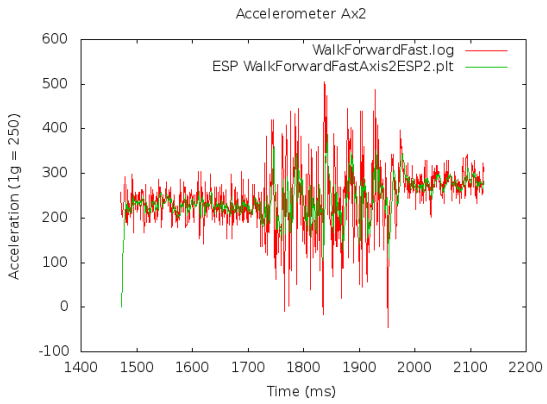
since the compensation vector must be calculated on the Nokia phone and then sent to the CM-5 via infrared, which is slow. This latency issue could easily be solved by attaching a 3-axis accelerometer to one of the AtMega128 Analog-to-Digital Converter (ADC) channels and moving the calculation of the compensation vector onto the CM-5. The latency issue could also be solved by using other forecasting techniques to predict future accelerometer data which limits the affect of the latency as opposed to using the raw data. A second problem is that the accelerometer is on the N5500, which forms the head of the robot and is nowhere near the robot's Center of Gravity (COG). The g-force at the head may be quite different from the g-force at the COG. This placement issue can be solved by placing the accelerometer at the COG or by the use of multiple accelerometers.



(a) Axis 0 (Frontal Plane)

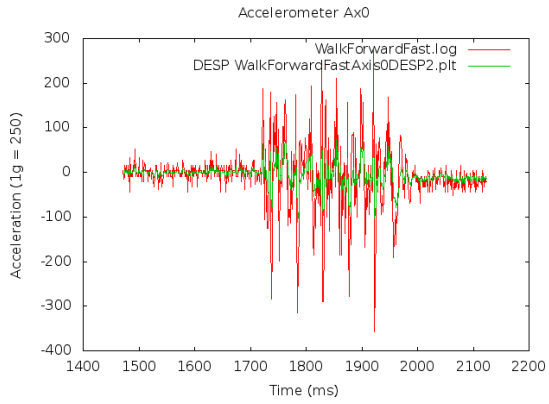


(b) Axis 1 (Traverse Plane)

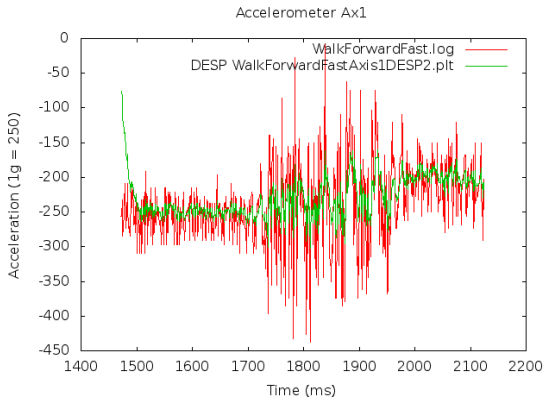


(c) Axis 2 (Lateral Plane)

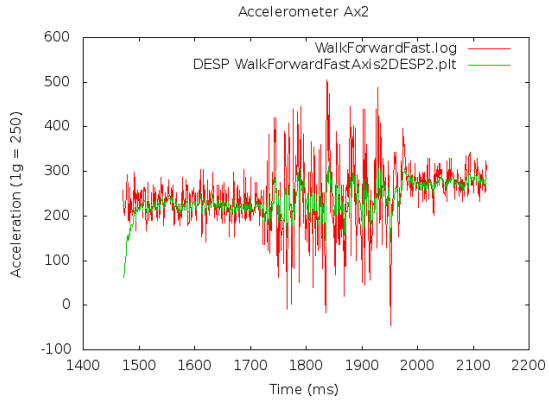
Figure 5.10: Accelerometer Data for each Cardinal Plane with ESP



(a) Axis 0 (Frontal Plane)

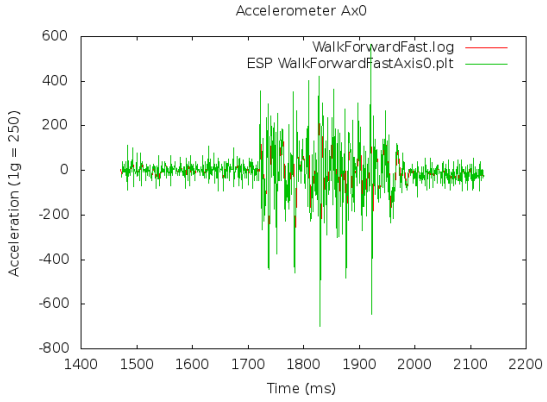


(b) Axis 1 (Traverse Plane)

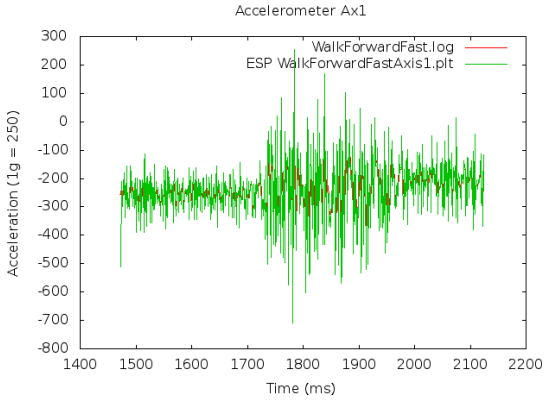


(c) Axis 2 (Lateral Plane)

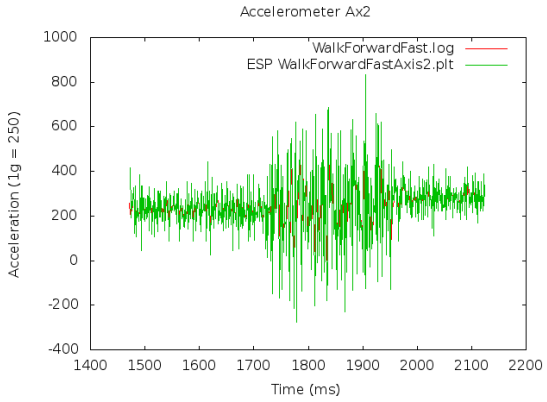
Figure 5.11: Accelerometer Data for each Cardinal Plane with DESP



(a) Axis 0 (Frontal Plane)



(b) Axis 1 (Traverse Plane)



(c) Axis 2 (Lateral Plane)

Figure 5.12: Accelerometer Data for each Cardinal Plane with Rate of Change

5.1.7 Firmware

The AtMega128 on Blitz is using a custom firmware running on a multi-threaded RTOS code named Freezer OS that performs many functions.

There is no floating point co-processor on the AtMega128, therefore floating point calculations are slow. All arithmetic calculations are performed in fixed point to reduce the amount of Central Processing Unit (CPU) resources used. An example of when fixed point calculations are performed is during interpolation for trajectory planning, as described in Section 5.1.5.

Commands are sent from the Nokia N5500 phone to the AtMega128 via serial communication. The command protocol packets have been optimized to reduce the amount of data required to be transferred. A side effect of this optimization is that the resolution of the Dynamixel AX-12 servos has been reduced. The Dynamixel AX-12 servos have a resolution of 2^{10} unique positions. In order to encode the position of each servo in a single byte, the positions sent to the AtMega128 are divided by four and converted to an integer then multiplied by four in the firmware to determine the position. This causes a loss of resolution when the position is not divisible by four. The loss of resolution is negligible since missing a few pulses does not significantly change the servo position for these low grade servos.

Typically micro-controller applications are completely interrupt driven which can be problematic when there are real-time requirements because interrupts can prevent critical tasks from completing on time if care is not taken in the interrupt priority

scheme. Freezer OS allows both interrupts and threads to be used for tasks. Critical tasks can still use an interrupt at the highest priority if necessary.

In order to allow execution of multiple threads, context switching is achieved by saving general and special purpose registers on the thread stack during a timer interrupt. A scheduler determines what the next thread to execute will be and the thread's state is restored by loading the general and special purpose registers from the thread stack. Semaphores for thread synchronization have been implemented for cases when concurrent access to resources are required by the threads. Sleep has also been implemented for delaying threads by marking threads as blocked and performing No Operations (NOOPs) until the time slice is complete.

Interrupt driven context switching for the threads results in time slice based scheduling. Each thread is scheduled for a time slice and the timer interrupt is chosen such that it provides sufficient time for common tasks to complete their work. Time slice scheduling also implements a priority scheme by using a niceness value to prevent starvation of low priority tasks.

The main tasks or threads executing on the Freezer OS for Blitz are two serial communication receiver/transmitter tasks (one for communication between the Nokia N5500 phone and AtMega128 and the other for communication between the AtMega128 and Dynamixel AX-12 servos), a task to parse packets and dispatch commands, and a motion interpolation task to handle any movement requests. Generally command request is handled on its own thread.

The firmware also allows storage of static motions in flash memory that can be executed on demand. A motion can be defined as a sequence of servo positions executed in order, with specified delays between each step and interpolation methods for each step.

The robot state is saved continuously which allows correction vectors to be added to the robot's position quickly. This facilitates the balancing algorithm as described in Section 5.1.6.

5.2 Simulation

My simulated implementation is intended to approximate the physical implementation of Blitz closely: a plan created with the motion planner in simulation should be directly transferable to the real robot. In order to make this possible, it is first necessary for there to be a model of the robot and the world that represent each accurately. These models are used as inputs to the motion planner. For example, when specifying the initial, goal, and random configurations the robot model is used and the obstacles in the environment are represented by the world model. The robot models are also used to help solve the kinematics since their joints and links accurately model where the end effector or tool will be positioned and oriented. For example if the joint angles of the robot model are set to the desired values then the solution to the FK problem can be directly determined by checking the end effector or tool position in simulation because the physics engine solves by kinematics for you

by using the equations as defined in Section 2.8. The robot and world models are both used for collision detection, since they provide the required bounding objects to test for collision as described in Section 2.16. If the simulation did not exist the motion planner could execute offline before attempting to move the robot.

In the subsections that follow, I describe the computer used for simulations, followed by the simulation, models, and motion planner that are made possible by the Motion Simulator.

5.2.1 Computer

The computer used to run simulations has the following configuration:

Component	**Description**
CPU	AMD Phenom(tm) 9850 Quad-Core Processor X4 64 bit
GPU	GeForce 9500 GT/PCIe/SSE2
Memory	8GB RAM

5.2.2 Motion Simulator

The Motion Simulator is a custom C++ application that uses some third party libraries and is written specifically for the research conducted in this thesis. The

purpose of the Motion Simulator is to provide a framework to evaluate the motion planner. First and foremost the Motion Simulator provides one important feature: an incremental simulator that integrates with the motion planner. The incremental simulator is described in more detail in Section 5.2.3.1. The Motion Simulator can be used for rapidly prototyping robot and world models. The Motion Simulator can also be used to visualize the CSPACE and motion planner RRT in either real-time or faster than real-time as the motion planner executes. The Motion Simulator simplifies debugging and conducting experimental trials with different motion planner parameters. The motion planner executes within the Motion Simulator.

The following is a list of features that the Motion Simulator has, in no particular order:

- Incremental simulation.
- Create, save, and load world models.
- Create, save, and load robot models.
- Robot and world models stored in Extensible Markup Language (XML).
- Define the initial configuration.
- Define the goal configuration.
- Configure motion planner parameters.
- Configure simulation parameters.

- Run simulations in batch mode.
- Integration with motor adapter for real world testing.
- Toggle visualization mode on and off.
- Generate and record data relevant for evaluation.
- Generate screenshots and graphs that represent the motion plan.

5.2.3 Third Party Software

Many open source libraries were leveraged in my work that simplify the implementation of the Motion Simulator. The following section describes each. The language of choice for the Motion Simulator is C++, therefore all of the third party libraries are written in C or C++.

The major libraries employed are reviewed in the following subsections.

5.2.3.1 Open Dynamics Engine (ODE)

A means of incremental simulation was required in the Motion Simulator. Recall that a incremental simulator computes the current state after a set of inputs has been applied over a period of time, see Section 2.15 for a refresher. There are many physics engines that can be used as incremental simulators. Different physics engines were evaluated but Open Dynamics Engine (ODE) was chosen in the end. In addition to providing incremental simulation, collision detection and the basic

primitives necessary to model the robots and worlds is provided. ODE version 0.13 was used which is licensed under GNU Lesser General Public License, BSD license, or LGPL license.

ODE is an articulated rigid body simulator [Smith, 2006] that is fast, robust, stable, and highly configurable. ODE allows configuration of parameters that effect dynamics such as gravity and friction coefficients. Kinodynamic constraints can also be enforced on joints. At every time step collision detection is performed which produces a list of contact points. A contact point represents the intersection of two objects in the world. While ODE provides built in collision detection, custom collision detection can be integrated into the physics engine. The IK problem is not solved by ODE, but hill climbing is used to determine if joint configurations meet the goal configuration while satisfying joint limitations.

With ODE, incremental simulation can be performed at different resolutions. The size of the time step can be configured. The smaller the time step, the greater the accuracy of the simulation but the drawback of a small time step is slower simulation. To understand how the size of the time step decreases the accuracy of the simulation, consider an object moving at high velocity 1m/s. If the object is 0.25m away from an obstacle at time t and the simulation is stepped every 500ms, then the object will have collided with the obstacle 250ms before the next time step $t + 1$. If some simulation accuracy can be sacrificed for speed, the simulation can be stepped in a faster than real-time. For the time step size, a balance must be found between

accuracy of the simulation and speed. The time step can typically be made larger (faster than real-time) if the robot and obstacles are slow moving and the obstacles are sparse.

In ODE, rigid shapes are represented by what they call *geometry objects* that store geometrical properties such as size, shape, position, and orientation. Each geometry object is attached to a *rigid body object* that stores dynamic properties such as velocity, acceleration, forces, and mass. The geometry object and rigid body object together constitute the simulated object. The simulated objects can be grouped together in a Space. ODE has three different types of Spaces: Simple, Multi-Resolution Hash Table, and Quadtree [Smith, 2006]. In order to optimize collision detection, ODE performs collision culling on the Space which identifies pairs of geometry objects that may collide instead of performing collision detection between every pair of geometry objects. The Simple Space does not perform collision culling, which means it tests every pair and therefore has a time complexity of $O(n^2)$. The Multi-Resolution Hash Table Space tracks simulated object that overlap in cubical cells in a internal data structure, collision culling is performed which leads to a time complexity of $O(n)$ in sparse domains. The Quadtree Space time complexity is not given but appears to perform better dense domains. The Multi-Resolution Hash Table Space is used in the implementation presented in this thesis. The pairs of geometry objects that are identified as possibly being in collision are passed to a callback function for further analysis to determine the exact contact points between the two geometry objects.

The callback function used by the collision culling algorithm can be specified which allows for customization of the callback function. In the implementation presented in this Thesis the callback function has been customized to allow for enabling a mode where the contacts points are calculated but not generated. Once the contact points are generated, the physics engine will apply the appropriate forces to the geometry objects in collision. In some cases this is not desirable: for example when performing move tests during RRT construction we only need to know if the configurations are in COBS and we do not want to apply collision forces to the robot and world model during this process, since it would use unnecessary resources.

Simulated objects can be constructed in a few different shapes such as spheres, cylinders, and cubes. These simulated objects can then be connected by a few different joint types such as ball, hinge, slider, and Angular Motor (AMotor). For the robots in this thesis, only revolute joints are considered. This accomplished in ODE by combining a ball joint with an AMotor joint. The AMotor can be configured to allow rotation on a specific axis which constricts the ball joint that is free to rotate about any axis.

5.2.3.2 Boost

Boost Graph library is used in the Motion Simulator to provide tree data structures and search algorithms for the motion planner RRT. The generated trees can also be output in the DOT language which are used to show the motion planner

RRT structure.

Boost provides free peer-reviewed portable C++ source libraries [Boost, 2014]. Boost is a set of peer-reviewed open source libraries that work well with the standard C++ library. Boost also focuses on providing libraries that can eventually become part of the new standard C++ library. For a full list of libraries see the Boost documentation. Boost version 1.55 is used, which is licensed under Boost Software License.

5.2.3.3 Approximate Nearest Neighbor (ANN)

ANN is used in the Motion Simulator exclusively for the motion planner RRT constructions. Recall from Chapter 4 that a nearest NN method is required to construct the RRT. When a random configuration is generated, the NN must be determined and an attempt made to connect the two configurations. Also recall from Section 2.12 that k-d trees store configurations in a space partitioning scheme that can significantly improve the NN query time.

ANN [Mount, 2010] is an open source library that can compute exactly or approximately k-Neareset neighbors for any dimension space. Parameters such as distance metric and k-Nearest for the NN search algorithm can be easily configured. ANN version 1.1.2 is used which is licensed under GNU Lesser Public License. ANN is used extensively in the RRT implementation to determine the NN when building the RRT.

5.2.3.4 Graphviz

Dynamically generated graphs can be hard to visualize since a large list of vertices and edges cannot be easily comprehended by a human. Graphviz is used in the Motion Simulator to provide images of the RRTs generated which can provide useful information for debugging and understanding of how different parameters affect the creation of an RRT.

Graphviz is an open source library that can generate images of graphs [Graphviz, 2014]. The Boost library can output a graph to the DOT language [Gansner et al., 2006], then the output can be rendered by the Graphviz library. Graphviz version 2.26.3 which is licensed under Eclipse Public License.

5.2.3.5 Voro++

The standard implementation of an RRT has an inherent Voronoi bias, as discussed in Chapter 4. Voro++ is used in the Motion Simulator to depict Voronoi bias in the standard RRT algorithm. Voro++ is an open source library that can compute and generate images of 3D Voronoi tessellations [Voro, 2014]. Voro++ version 0.4.6 is used, which is licensed under BSD License.

5.2.3.6 Open Graphics Library (OpenGL)

OpenGL is used in the Motion Simulator to render the robot and world models and the motion planner RRT.

OpenGL is an open source Application Programming Interface (API) for rendering 2D and 3D graphics [OpenGL, 2014]. OpenGL version 4.3 is used, which does not require licensing for developers.

5.2.4 Robot Models

Robots can be modelled [Spong et al., 2005] as a set of joints connected by links. There are many different types of joints, but only revolute joints are used in this thesis. This is without loss of generality because it is not difficult to model other joints in a similar manner. Revolute joints rotate on a single axis, therefore each revolute joint represents a single DOF. Each link has an associated length and connects two joints, unless a kinematic chain ends with a joint there is always one more link than there are joints.

The Motion Simulator allows for the links to be created as simple shapes such as spheres, cubes, and cylinders which are given a name, size, mass, position, orientation, colour, and texture. An example of two link definitions are given in Listing 5.1. An algorithm was implemented that can automatically attach joints to the links. The algorithm requires a few parameters in order to attach the links. The first set of parameters are the names of the two links to be attached, one link in the case where the joint will be at the end of a kinematic chain. If a joint is simply attached to the links then the joint is placed such that it connects the origins of $link0$ and $link1$. Obviously this is not the desirable behaviour in all cases. Another set of parame-

ters called dx, dy, and dz to specify the delta from $link0$'s origin to place the joint and connect $link1$'s origin are input into the algorithm. Just as the orientation of a servo will affect the axis of rotation, the mounting orientation of the joint must be specified so three additional parameters drx, dry, and drz that rotate the joint are also input into the algorithm. Finally since revolute joints are only considered, the axis of rotation must be specified. The axis of rotation is specified to the algorithm by three boolean values for x, y, and y. An example of a joint definition is given in Listing 5.2.

There are a few additional parameters for joints that are not used by the algorithm that attaches them to links. An upper and lower limit can be specified that limits the joint's range of motion. When the robot model is for a real robot, two parameters that are used by the Motor Adapter discussed in Section 5.2.4.3 can be specified which are a servo identifier and an offset that is used to help translate the simulated joint angle to the real robot's joint angle.

Listing 5.1: Example XML Link Definition

```
<Object Name="LeftAnkleLateral" Type="9OdeGlCube" Texture=""
    X="-460" Y="630" Z="232.5" Rx="0" Ry="0" Rz="0" lx="400"
    ly="500" lz="320" Mass="100" R="0.1" G="0.1" B="0.1"/>
<Object Name="LeftAnkleLateralBracket" Type="9OdeGlCube"
    Texture="" X="-460" Y="722.5" Z="232.5" Rx="0" Ry="0" Rz
    ="0" lx="420" ly="520" lz="145" Mass="100" R="1" G="1" B
    ="1"/>
```

Listing 5.2: Example XML Joint Definition

```
<Joint Name="LeftAnkleLateralJoint" Type="10OdeGlJoint"
    Link0="LeftAnkleLateral" Link1="LeftAnkleLateralBracket"
    Joint0="LeftAnkleLateral" ServoID="16" ServoOffset="500"
    LowerLimit="-90" UpperLimit="90" Dx="0" Dy="-500" Dz="0"
    DRx="0" DRy="0" DRz="0" Sequence="0" DisableJoint="0"
    DisableLink="1"  XAxis="1" YAxis="0" ZAxis="0"/>
```

5.2.4.1 Sphere Robot

The first simple robot model is for proof of concept. A single 3D point with 6 DOF for translational and rotational velocity along each axis. The model is called the Sphere Model throughout the remainder of this thesis. The Sphere Model is used

in worlds with various static obstacles. 6 revolute joints are attached to the Sphere Model as described in Section 5.2.4. The Sphere Model is shown in Figure 5.13.

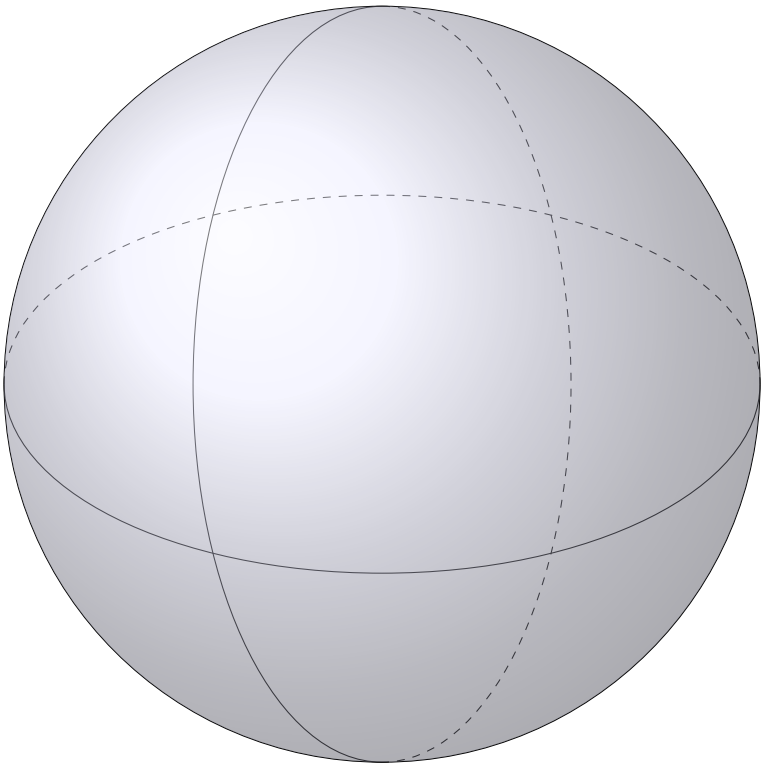


Figure 5.13: Sphere Model

The motion planner requires a method to test a move between two configurations of the Sphere Model in the Motion Simulator. The control of the Sphere Model in the Motion Simulator can be accomplished in a number of different ways;

A simple control strategy for moving the sphere model from one point to another is to determine the equation of a line segment between the two points. For example if the start point A is (1, 0, 1), and the end point B is (10, 0, 5) then the direction

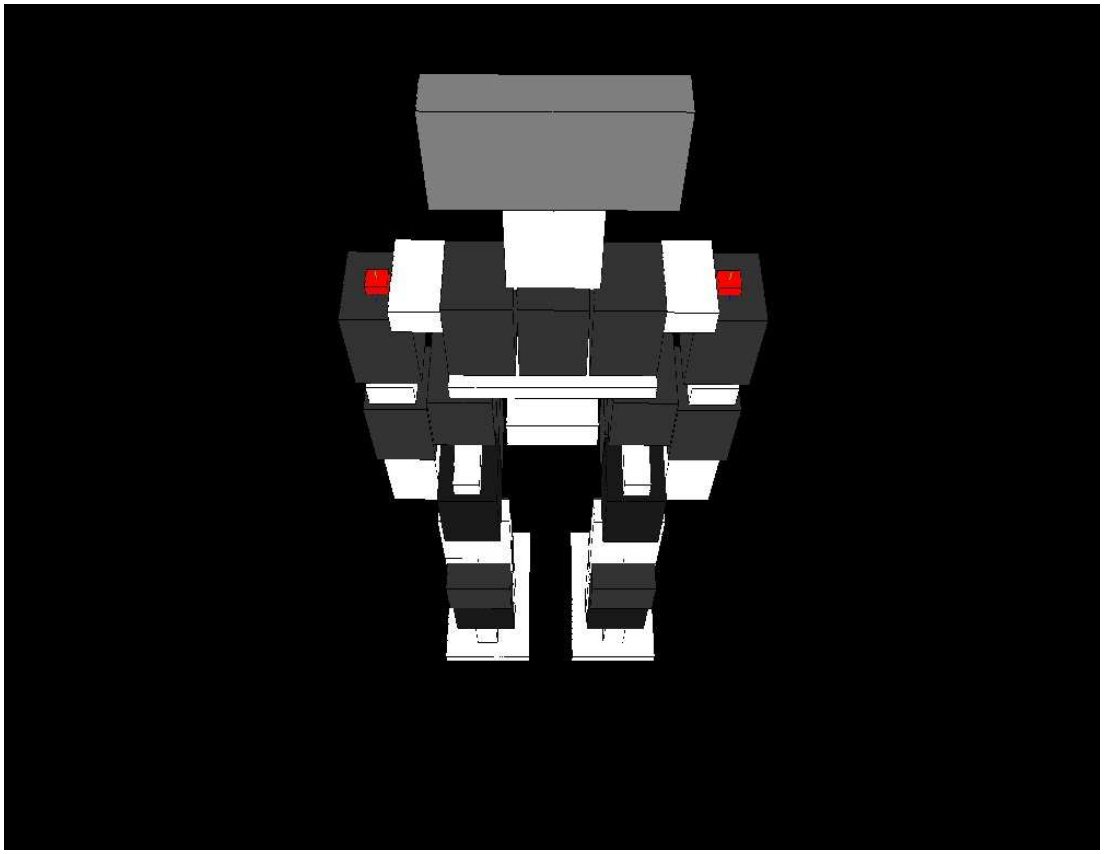
vector D is B - A = (9, 0, 4). To calculate a point on the line segment C = tD + A where t [0, 1]. t = 0 would generate A, and t = 1 would generate B. We can incrementally increase t in order to move the sphere model between A and B performing collision detection at every increment.

Another control strategy would be to rotate one the sphere's axes to point directly at the target using one of the three DOF that control rotation. A velocity or torque could then be added to the axis in order to move the sphere model towards the target. The velocity or torque could be varied with a PID controller to prevent overshooting the target. This solution makes use of the incremental simulator. This control strategy may be slow for determining if moves are feasible since it must be done inline with the incremental simulator time step.

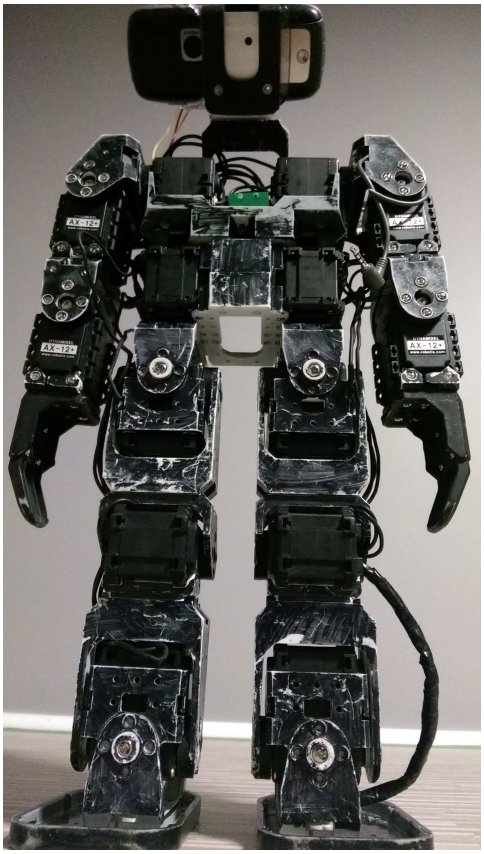
If we did not need to actually move the sphere, but simply only check if a move is feasible. A straight line collision check could be performed. A straight line could be projected from the start point A to the end Point B. The line segment could then be checked for collisions against all obstacles in the world. To simplify collision detection a sphere could be used regardless of the object shape as long as the sphere fully encompasses the object. This method could also be used for other simple robots such as a wheeled robot.

5.2.4.2 Humanoid Robot

The second robot model is of the real robot Blitz described in Section 5.1.1. The model is referred to as the Humanoid Model throughout the remainder of this thesis. Robotis has made Computer-aided design (CAD) models and drawings of all of the joints and links of their robot kit available. Using CAD models and drawings, the dimensions and masses of all the links and joints were input into the model. Nineteen revolute joints are attached to the Humanoid Model as described in Section 5.2.4. Care was taken to ensure all joints rotate at the correct orientation and the desired point between the links. The Humanoid Model is shown with Blitz for comparison in Figure 5.14 and an example of the joints moving is given in 5.15.

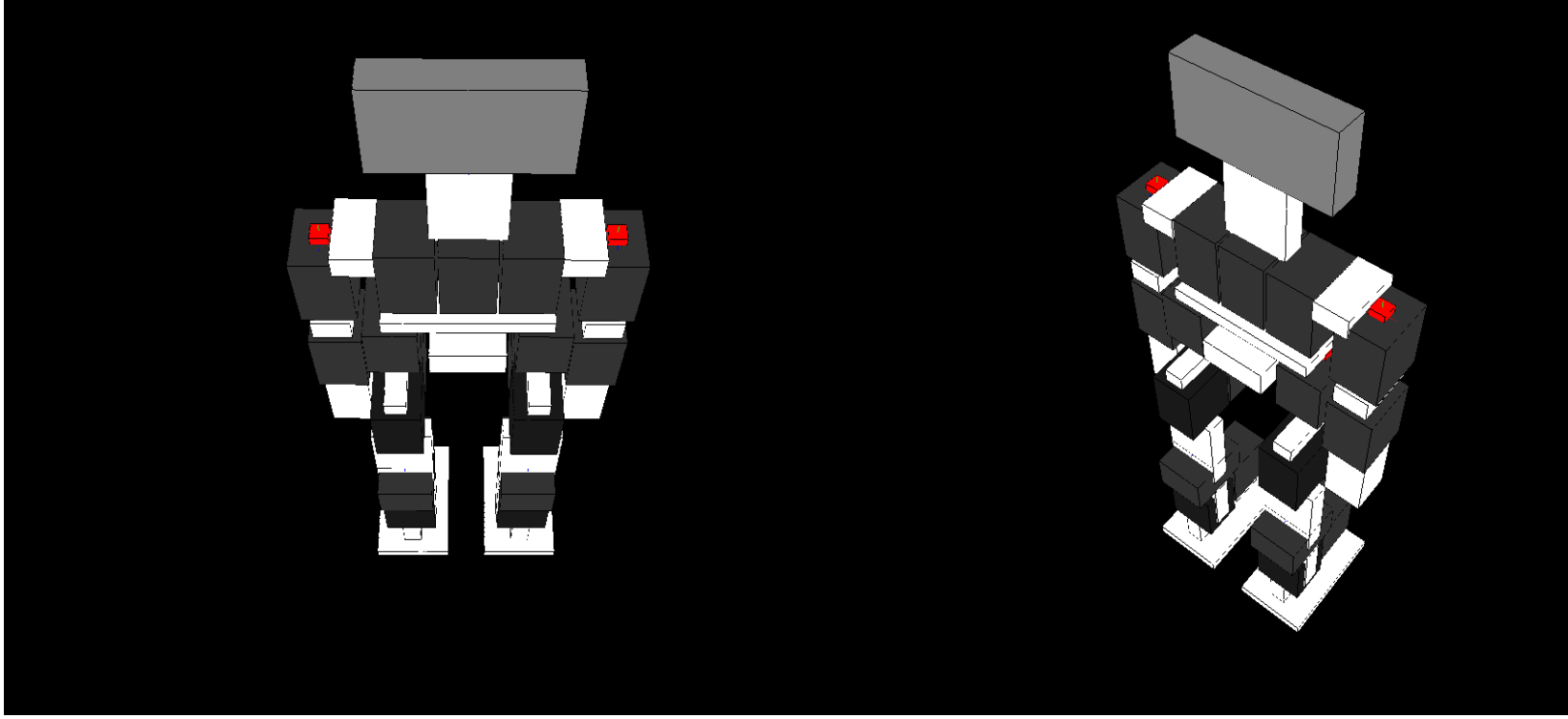


(a) Blitz Robot Model



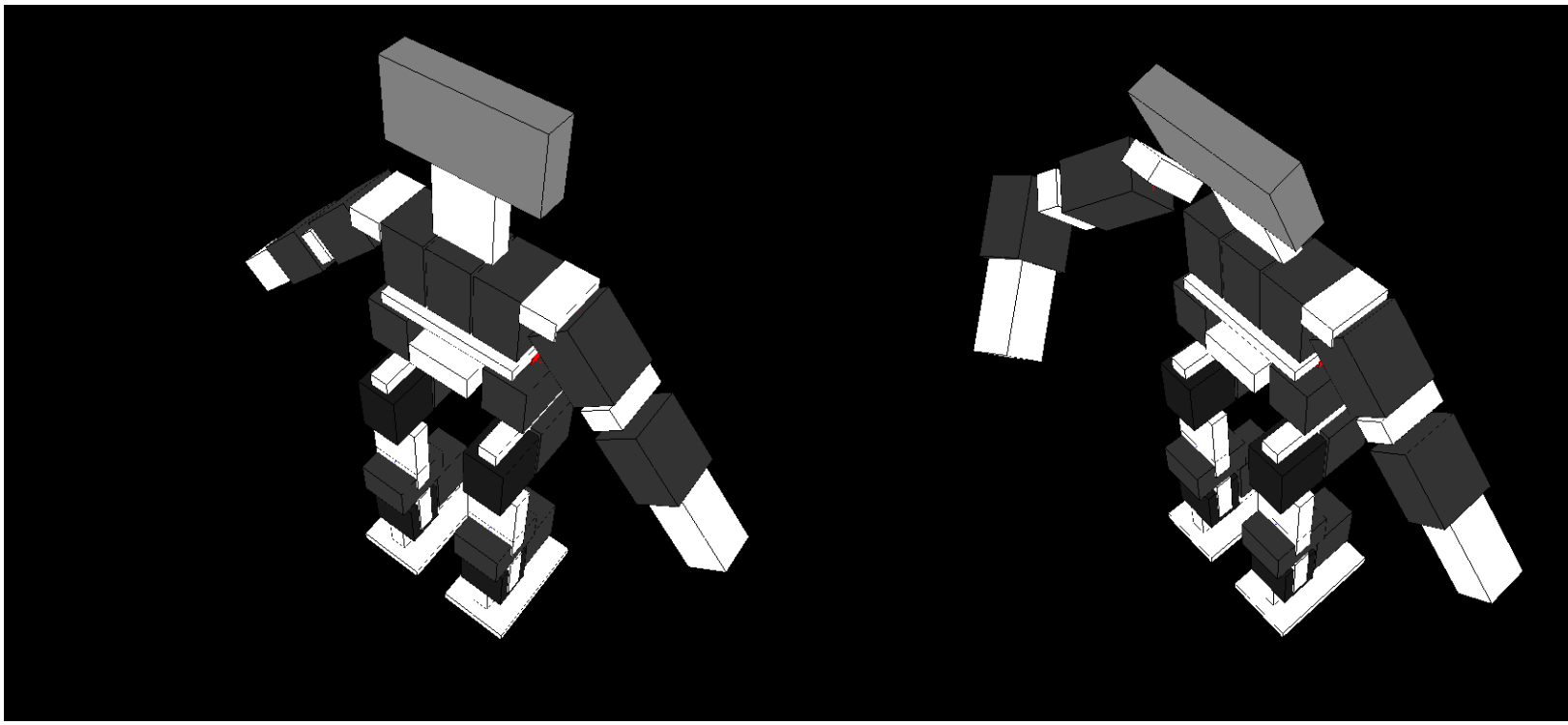
(b) Blitz

Figure 5.14: Blitz Robot Model vs. Blitz



(a) Humanoid Model View 1

(b) Humanoid Model View 2



(c) Humanoid Model View 3

(d) Humanoid Model View 4

Figure 5.15: Humanoid Model

The motion planner requires a method to test a move between two configurations of the Humanoid Model in the Motion Simulator. The control of the Humanoid Model in the Motion Simulator can be accomplished in a number of different ways;

The IK can be solved as described in Section 2.8.3 for each kinematic chain since the link lengths and joint locations are all known. Alternatively the IK can be solved by using a Jacobian technique as described in Section 2.8.3.1 while taking advantage of the incremental simulator. This is accomplished by changing each joint by a small amount in order to see how it changes the end effector or tool position. The FK solution for the end effector or tool position can be extracted directly from the Humanoid Model when the joint angles are changed because simulation maintains the defined joint constraints for the Humanoid Model. Using this method the partial derivatives for the Jacobian are calculated. The Jacobian can then be used to solve the IK.

5.2.4.3 Motor Adapter

The plan generated for the Humanoid Model must somehow be translated to a plan that the physical robot Blitz can use. This is accomplished by what is referred to as a Motor Adapter throughout the remainder of this thesis. A Motor Adapter converts the robot model joint angles to servo positions with a function $f(\theta)$ where θ is the angle of the robot model joint. Each servo type would require a different Motor Adapter because the specification of its position may be unique. For the purpose of this thesis only one Motor Adapter is required since only one servo type is used; the Dynamixel AX-12.

Recall from Section 5.2.3.1 that ball joints with AMotors are used to represent revolute joints. The AMotors are configured for Euler mode in ODE, which means the angle between the two links that the joint connects are automatically calculated. This angle is then used as the joint angle θ and passed to the function $f(\theta)$ that gives us the servo position for Blitz. $f(\theta)$ for each servo type can be determined by using the servo documentation. For example, with a Dynamixel AX-12 servo has 2^{10} positions that can be specified where each position maps to an angle between $[0, 300]^\circ$, therefore the mapping function is a simple linear equation give in Equation 5.4. As stated in Section 5.1.7, the resolution of the Dynamixel AX-12 servo was reduced in order to reduce the packet sizes in the communication protocol which gives us Equation 5.5. Figure 5.16 is the User Interface (UI) for motor control that can be used to test the Motor Adapter in the Motion Simulator. The angle is set on the

model joint and real robot when connected.

$$f(\theta) = (\theta * (2^{10}/300)) + offset \tag{5.4}$$

$$f(\theta) = ((\theta * (2^{10}/300)) + offset)/4 \tag{5.5}$$

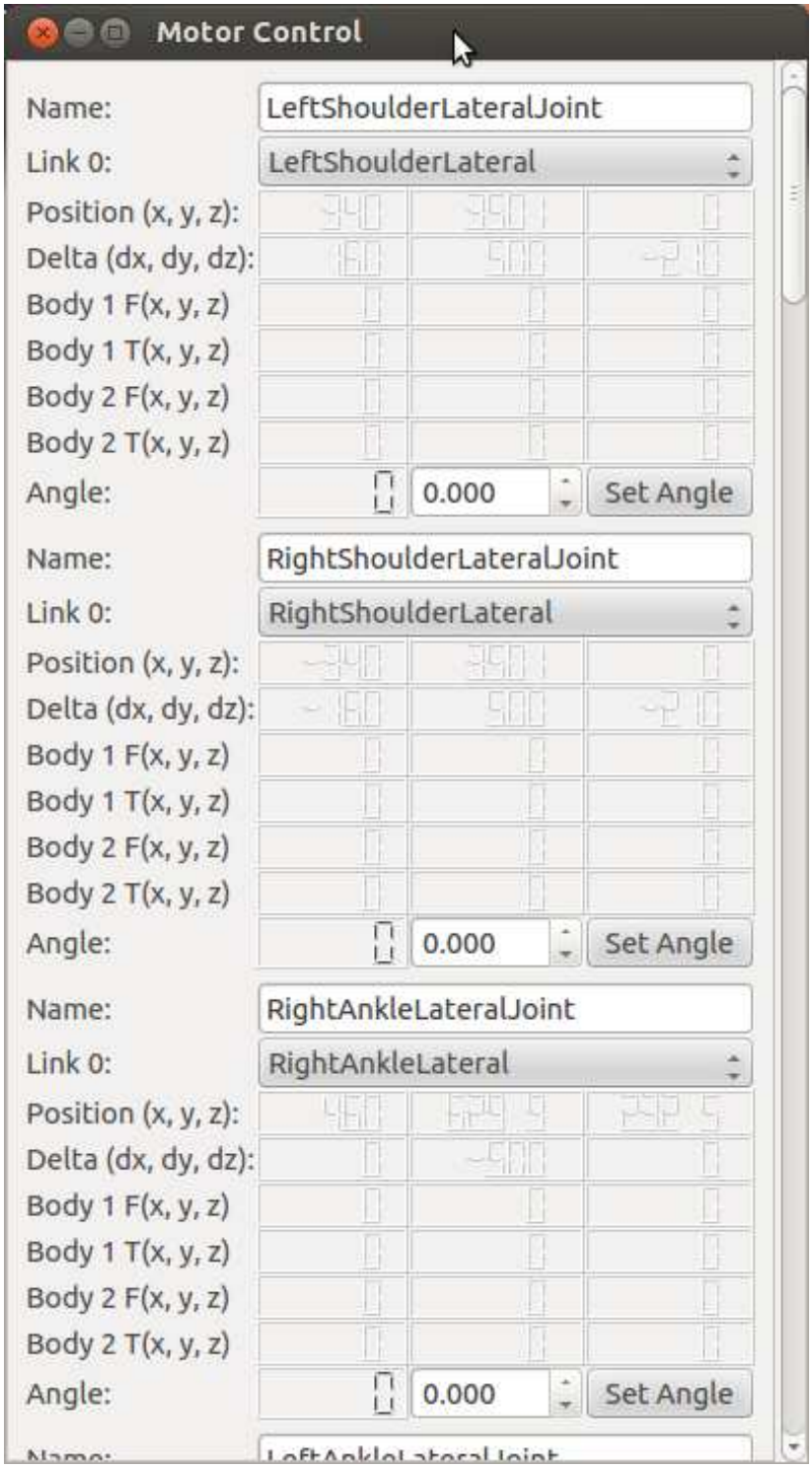


Figure 5.16: Motor Control

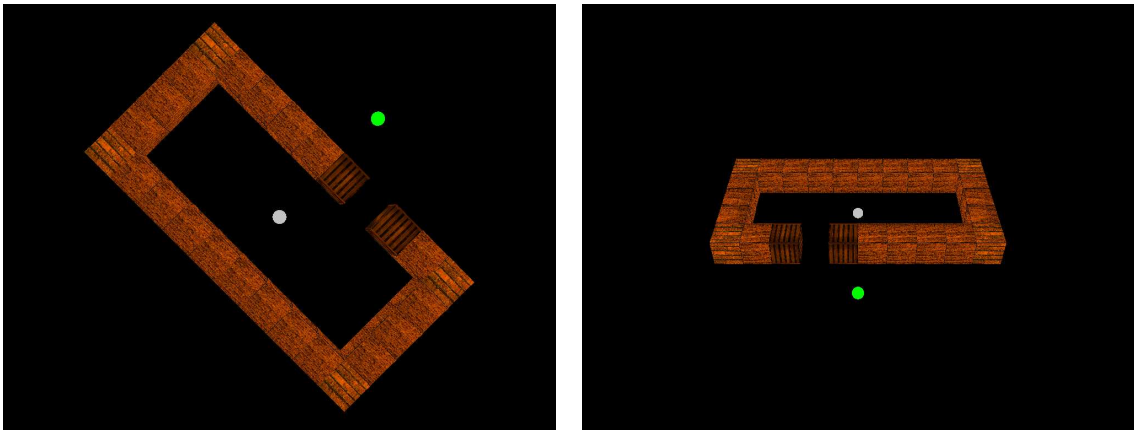
5.2.4.4 Motor Controller

The Motor Controller contains the logic necessary to move n servos simultaneously when servo commands are received. Servo commands provide a high level interface for servo position specification. The motor controller includes Pulse Width Modulation (PWM), trajectory control, interpolation, PID control, and serial communication. The output from each motor adapter is sent to the Motor Controller.

5.2.5 World Models

A world model consists of simple objects such as cubes, cylinders, planes, and spheres combined together to create a complex environment. The forces that act upon these objects can be simulated using the physics engine; forces such as friction and gravity. The Motion Simulator UI provides an easy method to create, modify, and arrange objects in the world. For this thesis six worlds were created to evaluate the Sphere Model and Humanoid Model. For the Humanoid Model worlds the static obstacles and goal are replicated in the real world. The six worlds are described and depicted below in their initial state.

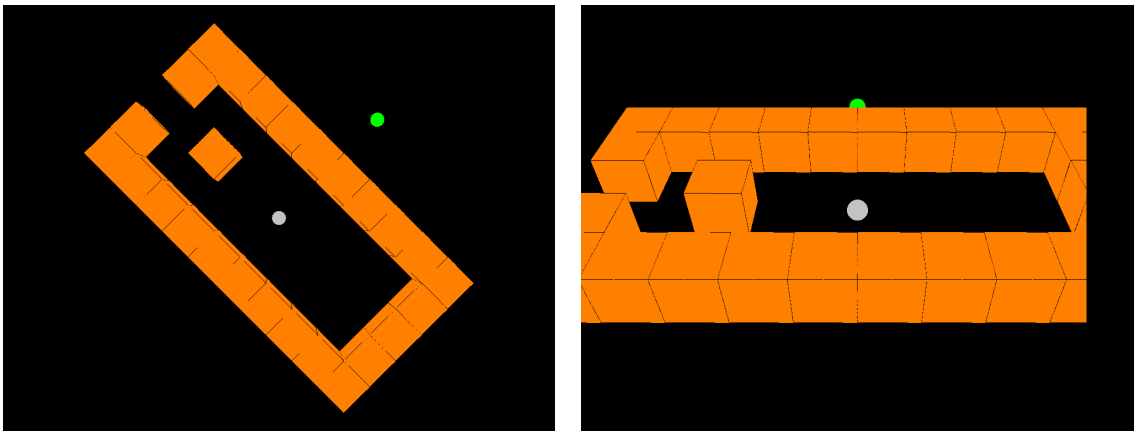
World 0, as shown in Figure 5.17, has the initial configuration of the Sphere Robot in the center of rectangular walls with a single exit. The goal configuration is just outside the exit. There is no straight path directly from the initial configuration to goal configuration.



(a) World 0 View 1 (b) World 0 View 2

Figure 5.17: World 0

World 1, as shown in Figure 5.18, has the initial configuration of the Sphere Robot in the center of rectangular walls with a single exit, but an obstacle is blocking the straight line path from the initial configuration to the exit. The goal configuration is along one of the walls. There is no straight path directly from the initial configuration to goal configuration.



(a) World 1 View 1

(b) World 1 View 2

Figure 5.18: World 1

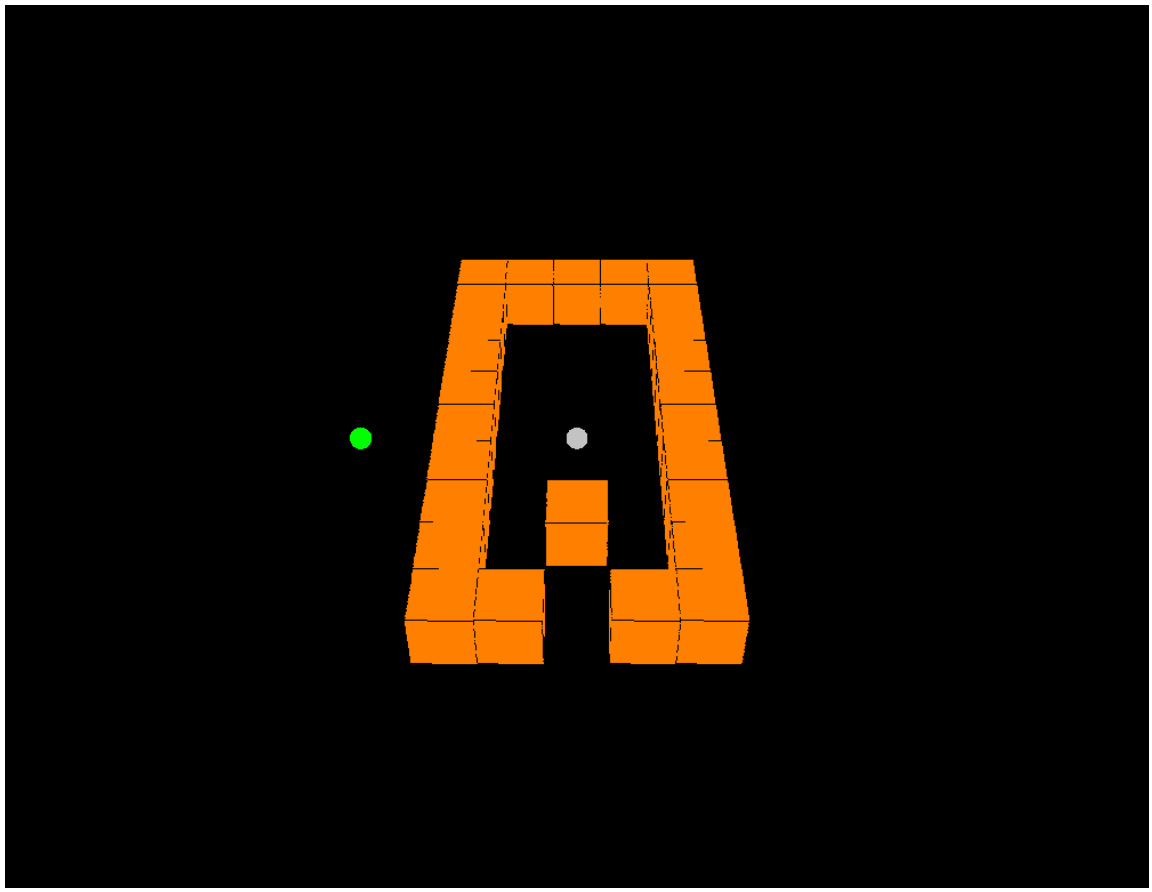
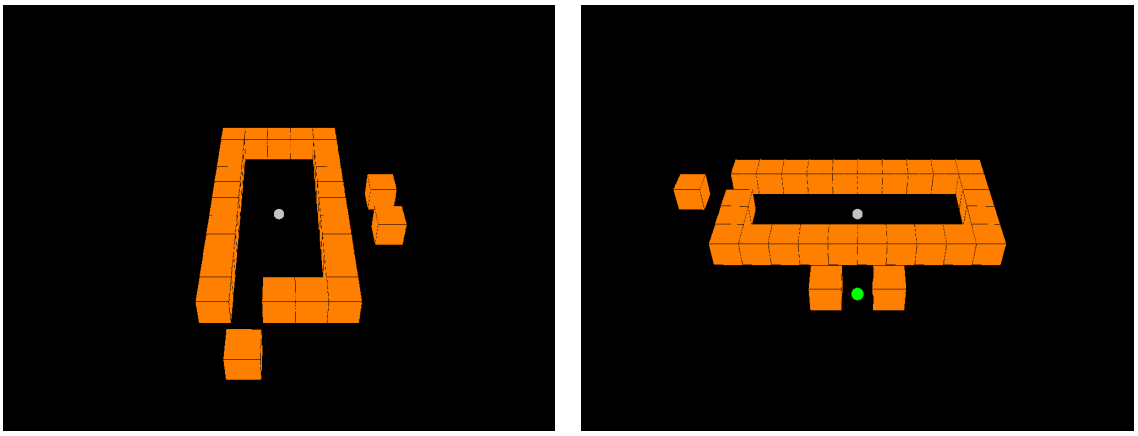


Figure 5.19: World 1-2

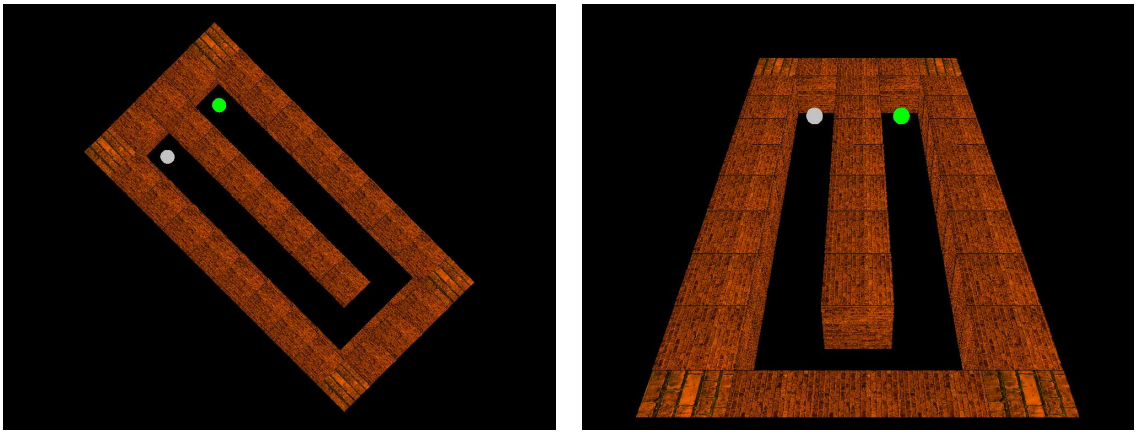
World 2, as shown in Figure 5.20, has the initial configuration of the Sphere Robot in the center of rectangular walls with a single exit, but an obstacle is just outside of the exit which presents a narrow path for the robot. The goal configuration is along one of the walls, but there are two obstacles surrounding the goal. There is no straight path directly from the initial configuration to goal configuration.



(a) World 2 View 1 (b) World 2 View 2

Figure 5.20: World 2

World 3, as shown in Figure 5.21, has the initial configuration of the Sphere Robot in the corner of a narrow passage. The goal configuration is in the corner of an adjacent narrow passage. The robot must navigate a u-shape narrow passage. There is no straight path directly from the initial configuration to goal configuration.



(a) World 3 View 1 (b) World 3 View 2

Figure 5.21: World 3

World 4, as shown in Figure 5.22, has the initial configuration of the Sphere Robot in the center of randomly scattered obstacles. The goal configuration is placed randomly behind an obstacle. There is no straight path directly from the initial configuration to goal configuration.

(a) World 4 View 1 (b) World 4 View 2

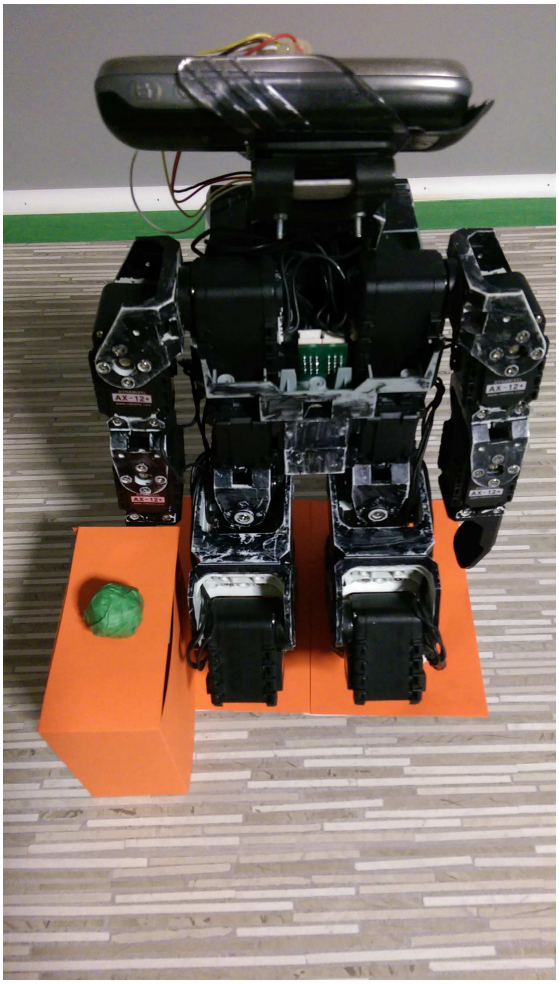
Figure 5.22: World 4

World 5, as shown in Figure 5.23, has the initial configuration of the Humanoid Robot with the tool (right hand) positioned close to a box. The goal sits on top of the box, and must be touched by the tool. There is no straight path directly from the initial configuration to goal configuration. World 5 is implemented both in simulation and the real world. The simulated models attempt to replicate the real world. Ideally, the motion plan that is found in simulation will translate directly to the real world if the simulated models are accurate.

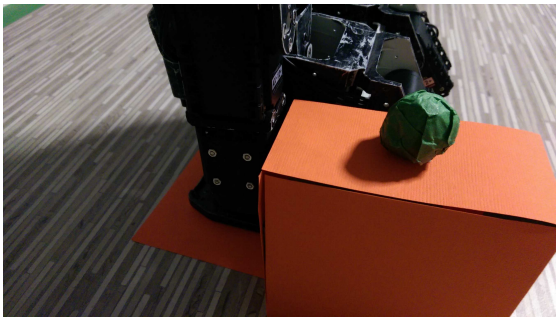
Although the vision system described in Section 5.1.3 could be used to provide the coordinates and sizes of the obstacles and goal in the world for the simulation, the purpose of this thesis is not to exercise the vision system. The coordinates, orientation, and sizes of the obstacles and goal in the simulation are pre-measured from the real world and input into the simulation. This is without loss of generality since vision systems that could provide this feedback exist. The real world is shown in Figure 5.23, and the model world is shown in 5.24.

If the motion planner were to simply solve the IK problem then allow the trajectory planner to interpolate between the initial configuration and configuration generated by the IK solution, the tool trajectory would look something like Figure 5.25. This would be clearly undesirable since the tool comes in contact and would be blocked by an obstacle.

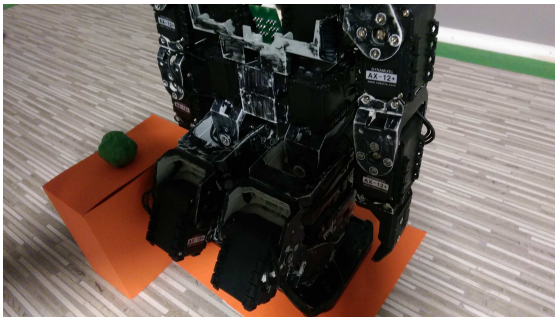
The trajectory of the tool would ideally look something like that shown in Figure 5.26: first avoiding the obstacle, then moving towards the goal configuration.



(a) Real World 5 View 1

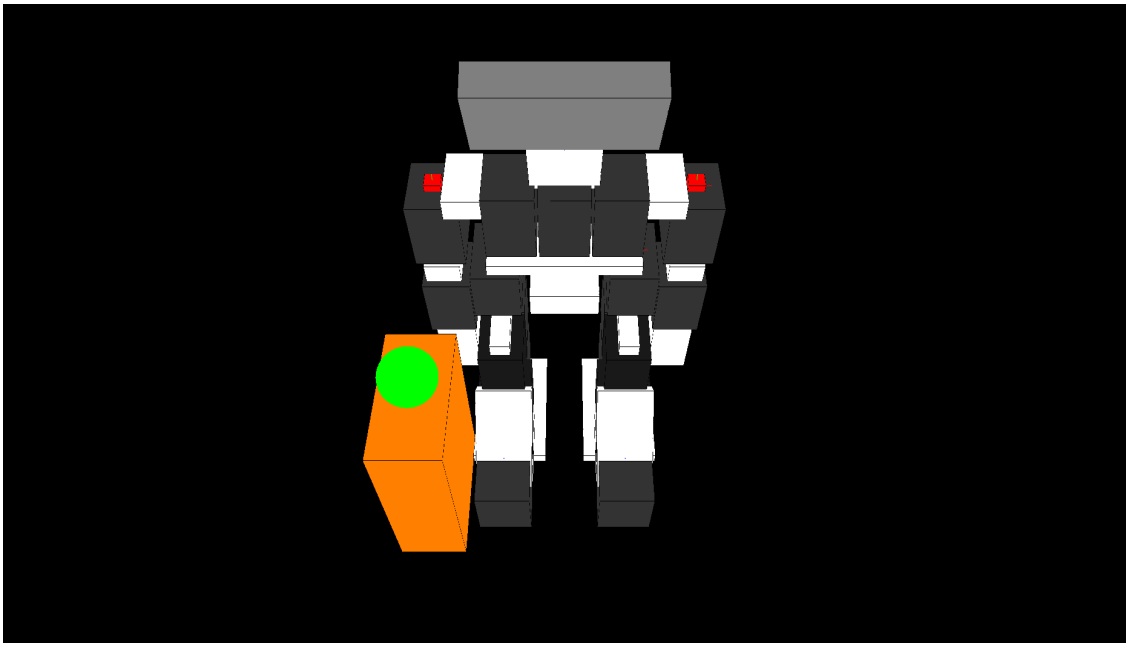


(b) Real World 5 View 2

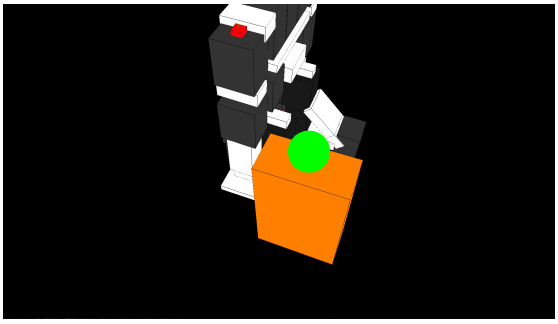


(c) Real World 5 View 3

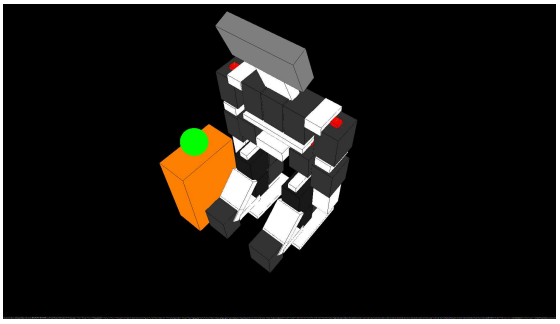
Figure 5.23: Real World 5



(a) Model World 5 View 1



(b) Model World 5 View 2



(c) Model World 5 View 3

Figure 5.24: Model World 5

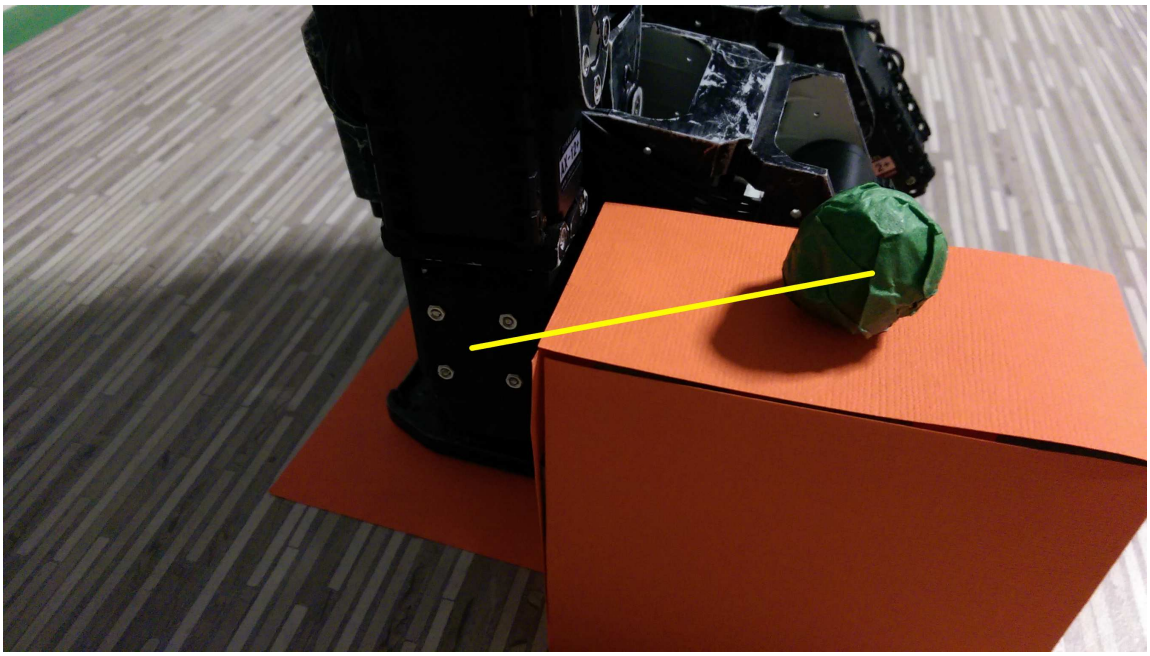


Figure 5.25: World 5 Tool Trajectory Collision

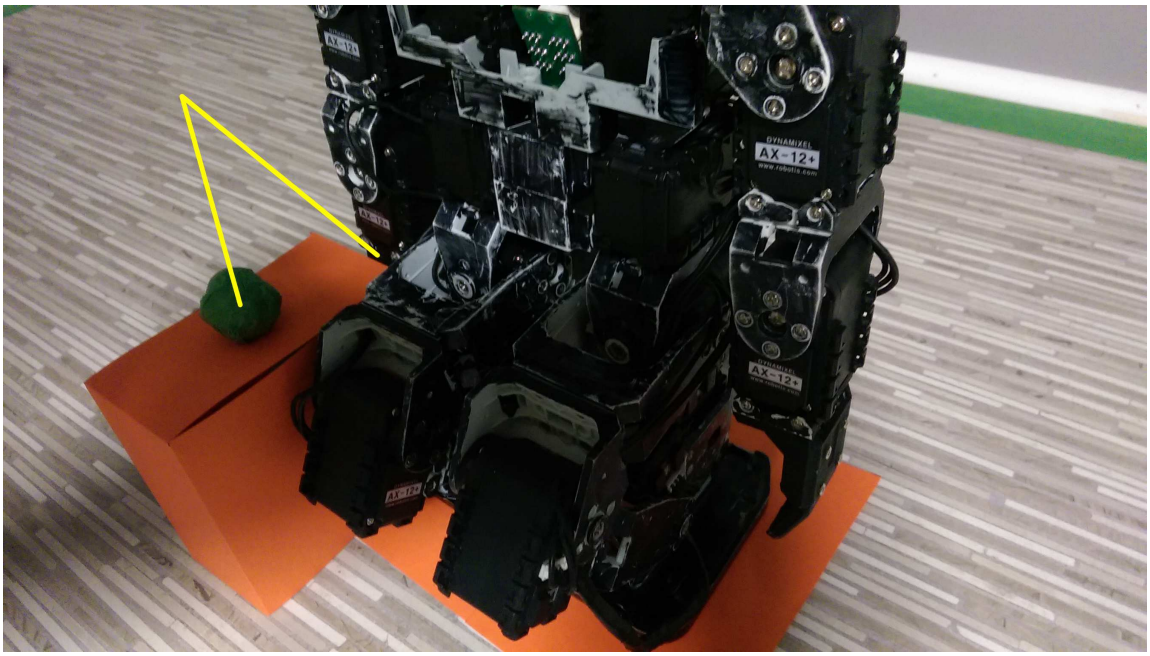


Figure 5.26: World 5 Tool Trajectory

5.3 Motion Planner with RRT and Incremental Simulator

My solution strategy for robot motion planning uses a RRT combined with an incremental simulator. The motion planner is currently a component in the Motion Simulator. By making the motion planner a component in the Motion Simulator, the Motion Simulator can both serve to present an environment for simulated experimentation, and for the motion planner itself to predict the outcome of the actions it is considering. The motion planner component and incremental simulator could easily be removed from the Motion Simulator and implemented on a computer onboard a robot in the future. My motion planner can be classified as a single-query probabilistic sample-based tree algorithm.

The motion planner is classified as a single-query algorithm since it does not require building a map before a solution can be found, unlike the PRM (as discussed in Section 3.2). The solution is found by incrementally building a data structure, which ultimately encompasses the solution. When the solution is found, the motion planner halts and the solution is readily available with minimal effort required to extract the solution from the data structure. The motion planner is categorized as a probabilistic sample-based tree algorithm since the underlying data structure is a tree and it is built by using random robot configurations.

The motion planner operates in the context of the CSPACE as described in Sec-

tion 2.1 and is combined with an incremental simulator. The incremental simulator is a physics engine described in Section 5.2.3.1 which is used for computing state $x(t)$ at time t or the state of the robot at any given time, see Section 2.15 for more details. By using a physics engine it not only provides incremental simulation, but it also provides facilities to test the feasibility of robot configurations. For example kinematic constraints such as joint limitations are considered, kinodynamic constraints are also considered by performing physics calculations to confirm that a configuration is dynamically stable. The physics engine could also be used to find solutions to the IK problem using hill climbing techniques - however, I used a Jacobian transpose method as per Section 2.8.3.3. Finally, the physics engine is used to perform collision detection to confirm that a configuration belongs to CFREE. A similar approach in combining the motion planner with an incremental simulator was used by Sucan et al. [2008], but unlike their work, the sample-based tree planner is a RRT, and complex robots in many different environments are demonstrated.

The single-query sample-based tree algorithm is a variant of the RRT construction algorithm as described in Chapter 4. In particular, the motion planner uses two algorithms from Chapter 4, the EXTEND and CONNECT (Algorithms 8 and 10). The motion planner builds an RRT while exploring the CSPACE. Nodes in the tree represent robot configurations. Robot configurations are randomly generated and a move attempted towards them from the nearest configuration. The configurations that are subsequently added to the RRT are only used if they meet the following

intelligent criteria: they belong to CFREE, the robot configuration is dynamically stable, and the robot configuration is feasible given realistic joint limitations (no joints that can exert infinite force). Just as in any standard RRT a few key components are required: a sample bias method, NN method, a distance metric, and a collision detection method. The standard implementation of a RRT has an inherent Voronoi bias when using uniform sampling because the probability that a configuration in the tree is selected is proportional to the volume of its Voronoi region [Lindemann and LaValle, 2004]. Configurations with larger Voronoi regions are more likely to be chosen. The convergence of the standard RRT implementation can be improved with a goal bias [Lavalle and Kuffner Jr., 2000] which simply selects the goal configuration instead of a random configuration using a predefined probability. The predefined probability must be chosen with care because a probability that is too large can potentially lead to local minima traps. The sample bias method used is a goal bias method. If the goal bias probability is set to zero then the RRT has the standard Voronoi bias, as discussed in Section 2.13. The NN method uses a third party library as described in Section 5.2.3.3 that can configured for k-NN and different distance metrics. The distance metrics used are based on Euclidean distance as described in Section 4.3. The collision detection method is built into the physics engine, and uses a mesh method similar to the bounding polygon collision detection method described in Section 2.16.

What is unique about my motion planner is that these algorithms are combined

with an incremental simulator. In both EXTEND and CONNECT, the algorithms require a move between two configurations that generates a motion that minimizes the distance between the two configurations. This move is where the incremental simulator is used. What also sets my motion planner apart is that the sample bias method, distance metric, k-Nearest parameter, collision detection method, and incremental time step are all configurable, which allows for these parameters to be changed on the fly for different robots and worlds. To the best of my knowledge after surveying many RRT papers, the ability to visualize the internal RRT structure is unique. The ability to visualize the internal RRT structure provides insight into the performance of the motion planner. The RRT structure can be visualized as a tree or a Voronoi diagram.

For simulated robot models, a method to control the joints is necessary. The strategy used to control the joints is by use of a PID controller as described in Section 2.11. The target angle for the joint is set and the PID controller manages the low level details of moving the joint to the target angle. This was also the control strategy of choice since the real robot's servo motors employ a similar control strategy.

For the real robot, the motion planner uses a simulated robot model during planning. In order to translate the motion planner output to the real robot, a method to convert the simulated robot model joint angles to the real robot is required. This Motor Adapter discussed in Section 5.2.4.3 was used to perform this translation.

5.3.1 Random Number Generation

The motion planner requires a method of generating random configurations for the RRT. A Random Number Generator (RNG) is used to generate a random number within a specified range for each DOF. It was desirable to make the results of the motion planner repeatable so that the same results could be regenerated if the same settings for parameters were used for the motion planner. Pseudo RNGs use a seed value to begin the random sequence. If you create two pseudo RNGs with the same seed value, they will produce the exact same random sequence. Since the sequence can be repeated it is not a true RNG it is a pseudo RNG. If the seed value is saved for the RRT it is possible to produce the exact same RRT.The algorithm for the pseudo RNG is given in Algorithm 12, note that rand() returns pseudo random number between 0 and RAND_MAX. For experimentation many permutations of the configurable parameters are executed, and if a particular case must be inspected later, it can actually be run again given the seed value. Although the motion planner is probabilistic, we can deterministically reproduce the results. If a seed value is not provided then the pseudo RNG is seeded using the CPU clock ticks, which generates a different pseudo random sequence for every instantiation of the pseudo RNG.

Typically RNGs produce random numbers within a specified range called the sampling range. For an RRT the selection of the sampling range could be a range that encompasses every possible configuration. This would provide the most coverage of the state space. However information about the state space could be used to

reduce the size of the sampling range which can allow the RRT to converge to a solution faster. Avoiding sampling the entire state space is also advantageous because potentially uninteresting regions of world can be ignored. The following are some options that could be used to reduce the size of the sampling range.

The sampling range can be incrementally increased inside of a bounding object. For example the sampling range could be bounded by a sphere. This brings into question how large the initial sampling range should be. A reasonable approach would be to use the distance from the initial configuration to goal configuration as the radius of the sampling range when a bounding sphere is used. Figure 5.27 depicts this type of incremental sampling range in 2D. Figure 5.28 depicts the sampling range as it is incrementally increased. Another approach for selecting the sampling range could be to use two radius, one as stated before and the other a radius extended from the goal configuration. Figure 5.29 depicts this type of incremental sampling range in 2D. This type of sampling range might be useful for instance if there are many obstacles surrounding the goal configuration. The algorithm for generating pseudo RNG in a sphere is given in Algorithm 13. An example of the samples is given in Figure 5.30. The next thing that comes into question is when should the sampling range be incremented? There are many possible options, for example if the goal configuration is not reached within a specified amount of time, if the number of configurations meets a threshold, or if the dispersion does not meet a threshold.

A pseudo RNG is used for one other task in the motion planner RRT. The sample

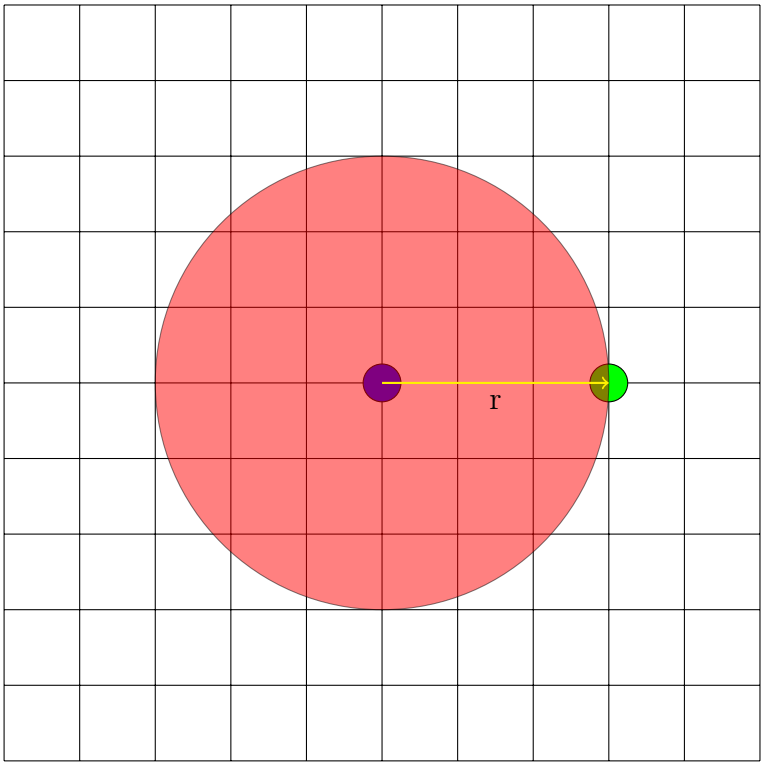


Figure 5.27: Pseudo RNG Incremental Sample Range, Single Radius

bias method is only used based on a defined bias probability. For example if the bias probability is 10% then the samples should be bias 1 out of 10 times. The pseudo RNG can be used to achieve the required bias probability by generating a random sample in the range [1, 100], if the number is less than or equal to 10 in our example of a 10% bias probability then the sample should be biased.

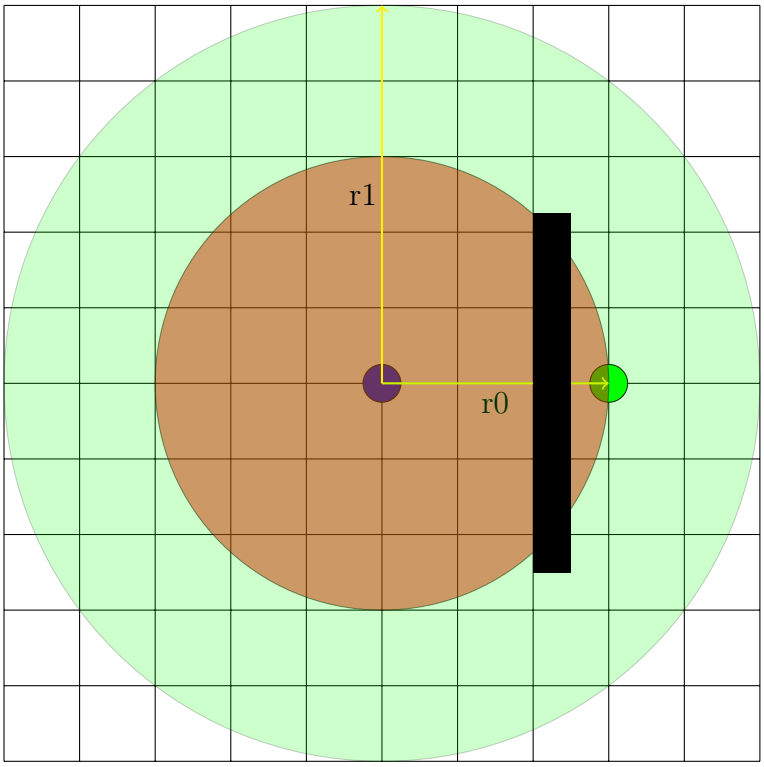


Figure 5.28: Pseudo RNG Incremental Sample Range, Single Radius Increment

Algorithm 12 Pseudo Random Number Generator

1: **procedure** RANDOM(min, max)

2: $r \leftarrow rand()\ /\ RAND\_MAX$

3: **return** $min + r * (max - min)$

4: **end procedure**

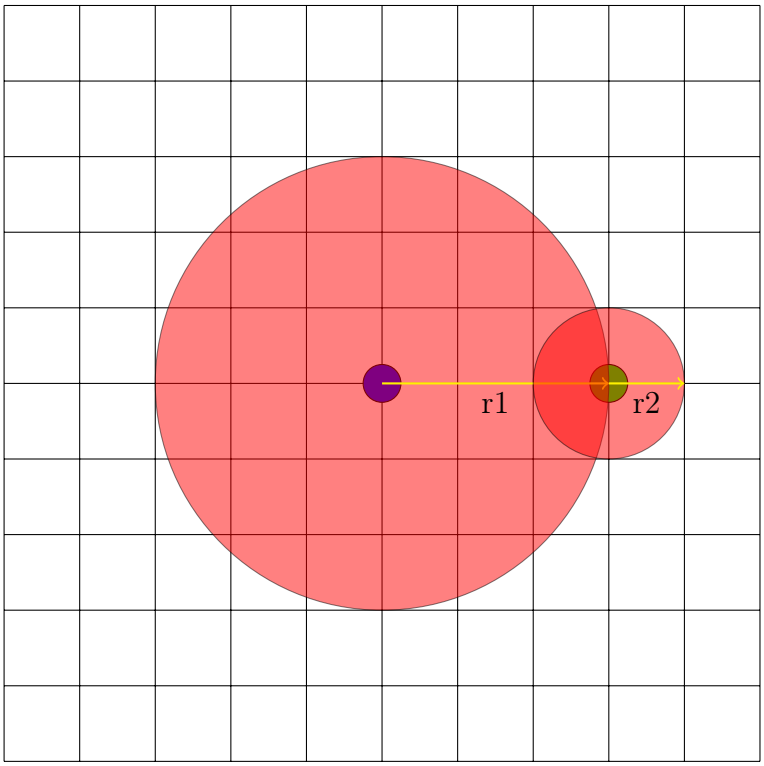


Figure 5.29: Pseudo RNG Incremental Sample Range, Dual Radius

Algorithm 13 Pseudo Random Number Generator in Sphere

1: **procedure** RANDOMSPHERE($radius$)
2: $l \leftarrow Random(0, 2\pi)$
3: $h \leftarrow Random(-\pi/2, \pi/2)$
4: $x \leftarrow Random(0, radius) * cos(l) * cos(h)$
5: $y \leftarrow Random(0, radius) * sin(h)$
6: $z \leftarrow Random(0, radius) * sin(l) * cos(h)$
7: **return** (x, y, z)
8: **end procedure**

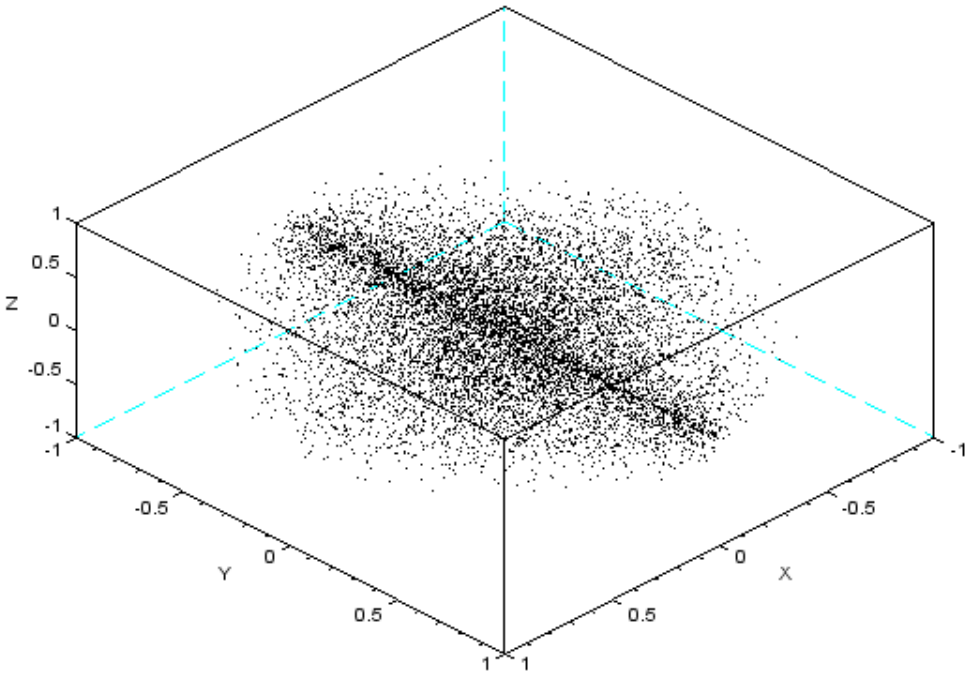


Figure 5.30: Pseudo RNG in Sphere

5.3.2 Random Configuration Generation

When generating random configurations for the robot, an naive approach could, for each joint, select a angle randomly over the entire joint range. However, this would likely generate many physically impossible configurations. A more sensible approach would be to consider joint constraints such as angle limits during random configuration generation. Pseudocode for the random configuration generation that is implemented is shown in Algorithm 14. Algorithm 14 uses a different approach for each robot type. The approach used for the Sphere robot is shown in Algorithm 15 and the Humanoid robot in Algorithm 16.

Algorithm 14 Random Configuration

```
1: procedure RandomConfiguration
2:     if SampleBias() then                ▷ Determine if sample should be biased.
3:         configuration ← Bias()
4:     else
5:         configuration ← RandomRobotConfiguration()
6:     end if
7:     return configuration
8: end procedure
```

Algorithm 15 Random Sphere Configuration

1: **procedure** RANDOMROBOTCONFIGURATION
2: $configuration[0] \leftarrow Random(MIN_X, MAX_X)$
3: $configuration[1] \leftarrow Random(MIN_Y, MAX_Y)$
4: $configuration[2] \leftarrow Random(MIN_Z, MAX_Z)$
5: $configuration[3] \leftarrow Random(MIN_R X, MAX_R X)$
6: $configuration[4] \leftarrow Random(MIN_R Y, MAX_R Y)$
7: $configuration[5] \leftarrow Random(MIN_R Z, MAX_R Z)$
8: **return** $configuration$
9: **end procedure**

Algorithm 16 Random Humanoid Configuration

1: **procedure** RANDOMROBOTCONFIGURATION
2: **for** $i \leftarrow 0, n$ **do** Where n is the number of joints.
3: $configuration[i] \leftarrow Random(j_i Min, j_i Max)$ ▷ Where j_iMin and j_iMax are the joint limits for joint i.
4: **end for**
5: **return** $configuration$
6: **end procedure**

5.4 Summary

In this chapter, the implementation of my approach to robot motion planning was described in detail. Simulation and real world environments were implemented

to evaluate my motion planner. In the next chapter, the evaluation of my motion planner is presented.

Chapter 6

Evaluation

6.1 Overview

In order to evaluate the motion planner presented in my thesis, experiments were performed. The purpose, setup, criteria, results, analysis, and observations of the experiments are discussed in this chapter.

6.2 Experiment Purpose

The purpose of my experiments was to evaluate the the performance of the motion planner presented in my thesis in a systematic way. Specific parameters of the motion planner were incrementally changed, while others were kept constant. As the parameters were incrementally changed, the behaviour of the motion planner

was observed and relevant data recorded. The data gathering was done in hopes of discovering by analysis additional optimizations that could be made to my motion planner, as well as judging its performance.

It is desirable to optimize the motion planner for real-time applications. In particular this means that the runtime of the motion planner must be reduced. The RRT construction algorithm can be optimized for real-time applications by only using methods that can be executed quickly. The methods that have the fastest execution time were determined empirically by investigating the runtime impact of different NN methods, sample biasing methods, incremental distance algorithms, and distance metrics. Using the data gathered, analysis was performed and potential modifications to the motion planner were identified. The modifications to the motion planner that were feasible were made and further experiments were conducted.

6.3 Experiment Setup

The first step of the experiment setup was to choose robots and worlds that were sufficiently complex to evaluate the performance of the motion planner and to demonstrate that the motion planner is generic enough that it could be applied to a wide array of different problems. For the robots, increasing the number of DOF provides sufficient complexity. For the worlds, their measure of difficulty depends in part on the robot, as a large robot may possibly have trouble with narrow passages while a small robot may have no trouble whatsoever. The robots and worlds I used

for my experimentation are discussed in detail in Sections (5.2.4.1, 5.1.1, 5.2.4.2, and 5.2.5). Simulated and real robots were chosen, and for the real robot an equivalent simulated robot model is built which the motion planner uses. Various worlds with narrow passages, random obstacles, and no direct path to the goal configuration were chosen.

For each world described in Section 5.2.5, the motion planner was executed for the robot with the following settings: the first combination of settings (which will be referred to as Setting 1 throughout the remainder of this thesis) are the EXTEND algorithm described in Algorithm 8, Euclidean distance metric described in Equation 4.1, k=1 NN computed exactly, and a variable goal bias incremented in 10% increments. The second combination of settings (which will be referred to as Setting 2 throughout the remainder of this thesis) are the CONNECT algorithm described in Algorithm 10, Euclidean distance metric described in Equation 4.1, k=1 NN computed exactly, and a variable goal bias incremented in 10% increments.

Each combination of settings was performed three times for the simulated robots using a different seed value for the RNG. The seed values were saved so that any runs that appeared to be anomalous could be ran again and inspected if necessary. For the real robot each combination of settings was performed once due to the amount of time it takes to execute the plan on the real robot.

The data was gathered as per the criteria set out in the next section during the experiments, which were conducted using the Motion Simulator described in

Section 5.2.2.

6.4 Experiment Criteria

Once the robots and worlds were chosen for evaluation, the criteria for evaluating the performance of the motion planner was defined. Since a major concern was making the solution applicable to real-time applications, one obvious metric is the Total Execution Time of the motion planner. The total execution time is the time from the moment the start and goal configuration are chosen and input into the motion planner to the time that the motion planner returns a path between the start and goal configuration. Another concern, due to limited resources on mobile robots such as micro-controllers, is the amount of memory used. In order to provide an idea of how big the motion planner data structure is, assuming that there is a correlation between the data structure size and memory usage, the Number of Configurations was used as a metric. A greater number of configurations can be good and bad. For example if there are a large number of configurations and they are dispersed, then it may be a good indication that the motion planner has done a good job at exploring the world. On the other hand, a large number of configurations that are not dispersed may be an indication that the motion planner is having a difficult time finding a solution if one exists. In order to understand how much exploration of the world is done, Dispersion is another metric used. The measure of dispersion used is simply the standard deviation between all robot configurations that are considered by the

motion planner throughout the planning process. A similar metric for dispersion was used by Lindemann and LaValle [2004] as a measurement of how well the CSPACE is covered by the RRT. Aside from number of configurations it is also interesting to consider the number of Modified Random Configurations as this is a good indication of a few things. Firstly the number of modified random configurations can give you a good indication of how cluttered the world is with obstacles if most of the generated random configurations are in COBS as opposed to CFREE. Secondly the number of modified random configurations can provide insight into how good the motion planner is at selecting configurations, if it is really good then no random configurations would need to be modified, they could simply be used. Since the motion planner is composed of many complex modules, it is important to understand not only the total execution time but also how some of the complex modules affect the total execution time since these modules may not be the focus of this research. For example the focus of this research is not solving the IK problem, or trajectory planning, therefore the Total Move Test Time was tracked. The total move test time encompasses the time that it takes to determine if the robot can move between two configurations. If we look at the delta between the total execution time and total move test time, we remove time that this research is not focusing on optimizing. A simple Jacobian method as described in Section 2.8.3.1 is used for solving the IK problem and trajectory planning is done by simple interpolation. Algorithms for these problems can always be changed, or updated with the current state of the art

but will not be optimized for this thesis. In addition to total move test time, any time incurred for searching data structures such as NN as described in Section 2.12 or graph and tree searching as described in Section 2.14 was tracked, since they are also not the focus of this research. Finally, a Qualitative Inspection Ranking considers the quality of the motion plan from a qualitative perspective, by manually inspecting the path. A number of questions are considered: Is the path reasonable, or is there clearly a better solution? Does the path translate well from simulation compared to the real world? Can the real robot execute the plan? Does the robot achieve the goal configuration from the initial configuration collision free?.

A summary of the criteria used for evaluating the motion planner is given in Table 6.1 and descriptions of the qualitative inspection ranking are given in Table 6.2.

Criteria	Description
Total Execution Time	The motion planner's response time to a plan request.
Number of Configurations	The number of configurations in the motion planner's data structure.
Modified Random Configurations	The number of generated random configurations that had to be modified before use in the motion planner's data structure.
Dispersion	A measure of how distributed the configurations are throughout the world
Total Move Test Time	The total amount of time to check if a move between to configurations is possible.
Total NN Query Time	The total amount of time to query the NN data structure
Qualitative Inspection	How good the generated plan is by manual inspection of the output.

Table 6.1: Summary of Evaluation Criteria

Rank	Description
Disagree	There is a different solution that is clearly better than the planner selected.
Somewhat Disagree	There is a different solution that may be better than the planner selected.
Somewhat Agree	There may be other solutions, but not much better than the planner selected.
Agree	The best solution was selected by the planner.

Table 6.2: Qualitative Inspection Ranking

6.5 Results, Analysis and Observations

In this section, the results of the experiments are presented with analysis and observations. Each of the result tables will be discussed in order. Common findings across all result tables are discussed first.

The data and results gathered from the experiments are summarized for the Sphere Robot in World 0 in Table 6.3. Each combination of parameters was run three times with a different seed value.

Table 6.3: Result Matrix World 0

Variant Description	NN k	NN Epsilon	Bias Probability %	Number of Configurations	Modified Random Configurations	Total NN Query Time (ms)	Total Move Test Time (ms)	Total Execution Time (ms)	Dispersion (STD)
EXTEND - Goal Bias	1	0	1	1337	1334	0	361263	371998	1328.46
				959	956	0	271984	279770	1094.49
				3057	3054	0	765805	793443	2269.65
EXTEND - Goal Bias	1	0	10	221	218	0	60186	61715	595.215
				171	168	0	44857	46108	459.172
				212	209	0	51062	52857	501.947
EXTEND - Goal Bias	1	0	20	123	120	0	30537	31469	390.563
				100	97	0	23725	24519	347.464
				141	138	0	32543	33557	309.794
EXTEND - Goal Bias	1	0	30	68	65	0	17821	18205	263.511
				83	80	0	19526	19982	241.752
				37	34	0	9309	9434	103.75
EXTEND - Goal Bias	1	0	40	52	49	0	13616	13914	178.372
				45	42	0	12213	12443	218.927
				37	34	0	9412	9497	111.366
EXTEND - Goal Bias	1	0	50	55	52	0	14715	14959	198.339
				70	67	0	17528	17800	225.603
				2402	2399	0	486911	504670	1150.72
EXTEND - Goal Bias	1	0	60	43	40	0	12514	12654	157.015
				61	58	0	15418	15659	179.228
				2982	2979	0	602860	624690	1375.78
EXTEND - Goal Bias	1	0	70	43	40	0	11013	11122	148.897
				64	61	0	15019	15231	150.638
				2425	2422	0	490852	503946	1178.47
EXTEND - Goal Bias	1	0	80	15247	15244	7	3229033	3775267	2844.56
				3538	3535	0	719869	741207	1302.37
				6729	6726	1	1350019	1445961	1764.51
CONNECT - Goal Bias	1	0	1	158	45	0	53576	54816	497.019
				76	56	0	22226	22818	345.483
				258	243	0	54684	56381	297.5
CONNECT - Goal Bias	1	0	10	136	43	0	46167	47312	518.001
				17	13	0	4405	4510	108.893

Table 6.3: Result Matrix World 0

Variant Description	NN k	NN Epsilon	Bias Probability %	Number of Configurations	Modified Random Configurations	Total NN Query Time (ms)	Total Move Test Time (ms)	Total Execution Time (ms)	Dispersion (STD)
				257	228	0	56990	59297	528.681
CONNECT - Goal Bias	1	0	20	23	16	0	6906	7041	223.787
				103	87	0	26130	26827	451.411
				279	228	0	67810	70518	682.761
CONNECT - Goal Bias	1	0	30	23	16	0	7011	7203	223.787
				216	165	0	62373	63845	686.359
				297	252	0	69497	71394	753.115
CONNECT - Goal Bias	1	0	40	11	8	0	3003	3067	84.2766
				45	30	0	14019	14236	322.251
				205	192	0	43654	44840	481.7
CONNECT - Goal Bias	1	0	50	25	17	0	7308	7416	202.161
				45	38	0	11013	11278	216.768
				2183	2106	0	453464	468251	1235.97
CONNECT - Goal Bias	1	0	60	21	15	0	6213	6284	106.07
				49	42	0	12416	12640	203.069
				1893	1845	0	389112	399435	1090.01
CONNECT - Goal Bias	1	0	70	21	17	0	5406	5443	96.8259
				49	44	0	11113	11291	151.783
				1264	1234	0	256796	261465	906.407
CONNECT - Goal Bias	1	0	80	483	475	0	103145	104200	493.368
				4313	4274	0	884007	916303	1415.79
				2028	1998	0	411561	420005	931.356

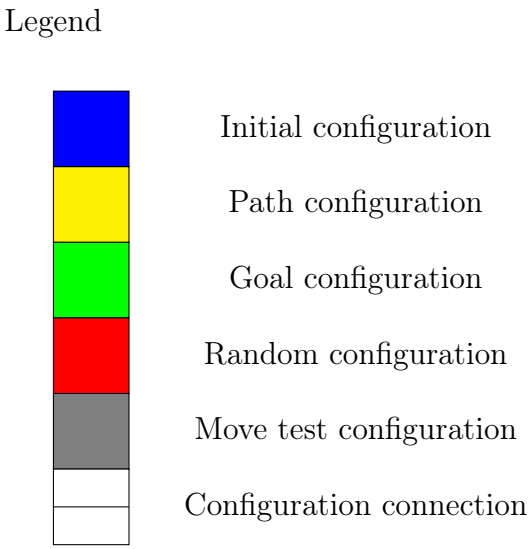
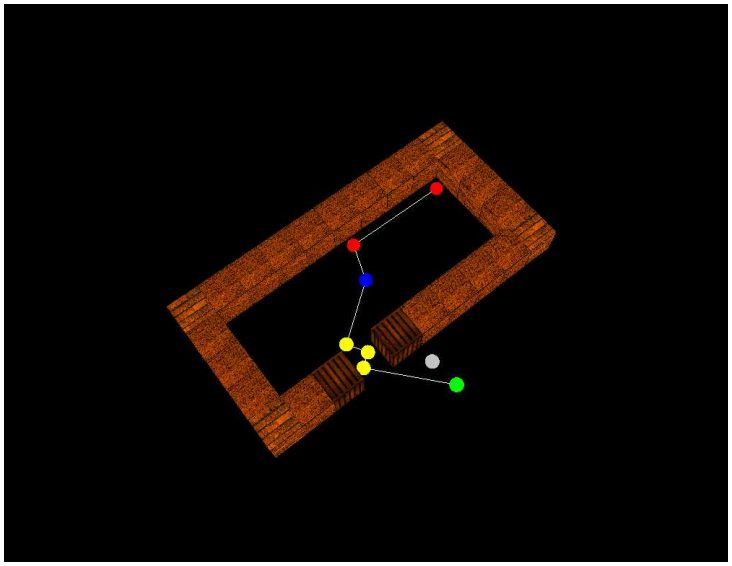


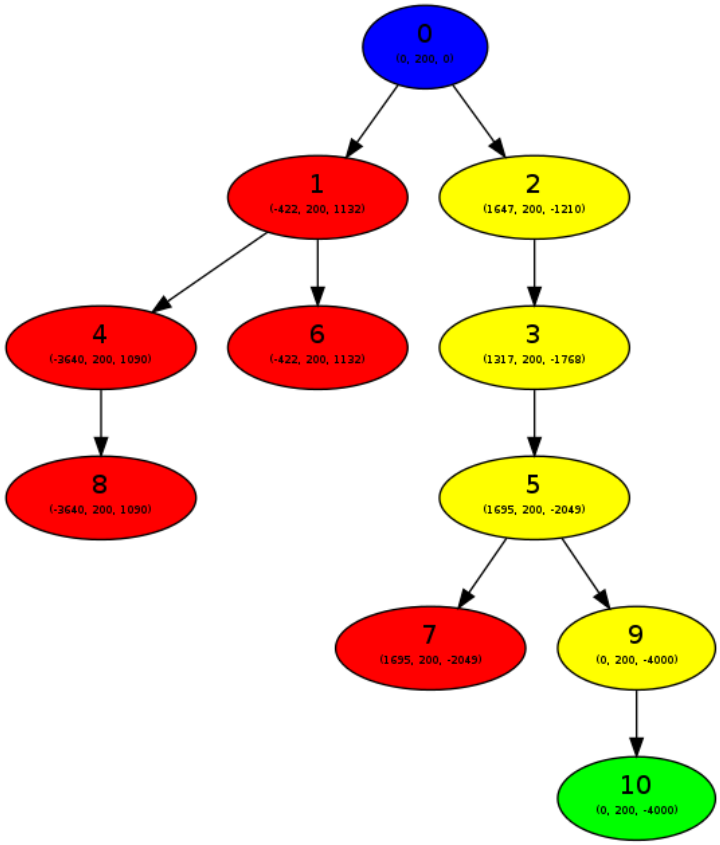
Figure 6.1: Colour Coding

All images in this section are colour coded to show the initial, path, goal, random, and move test configurations. Colour coding legend is as shown in Figure 6.1.

- Initial Configuration: The initial configuration is the starting state of the robot.
- Path Configuration: A path configuration is a configuration on the path from the initial configuration to goal configuration.
- Goal Configuration: The goal configuration is the desired end state of the robot.
- Random Configuration: A random configuration is a configuration that was randomly generated and added to the RRT.
- Move Test Configuration: A move test configuration is used to test if a move is valid between two configurations.
- Configuration Connection: A configuration connection shows that a move is valid between two configurations.

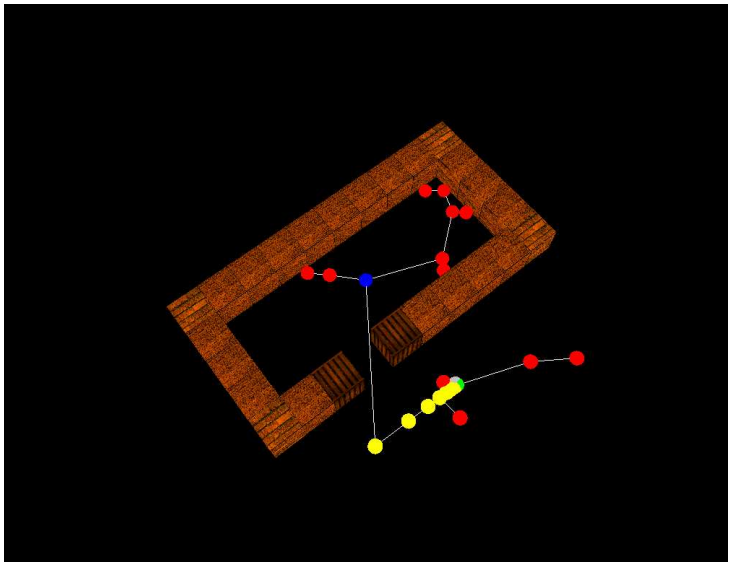


(a) Path

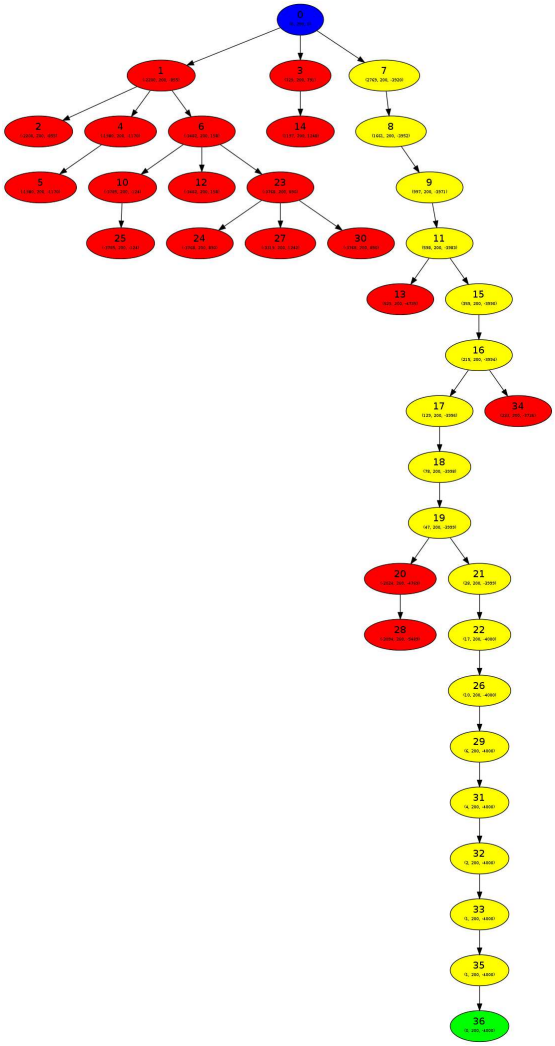


(b) Tree

Figure 6.2: Best CONNECT Result for World 0 - Goal Bias 40



(a) Path



(b) Tree

Figure 6.3: Best EXTEND Result for World 0 - Goal Bias 40

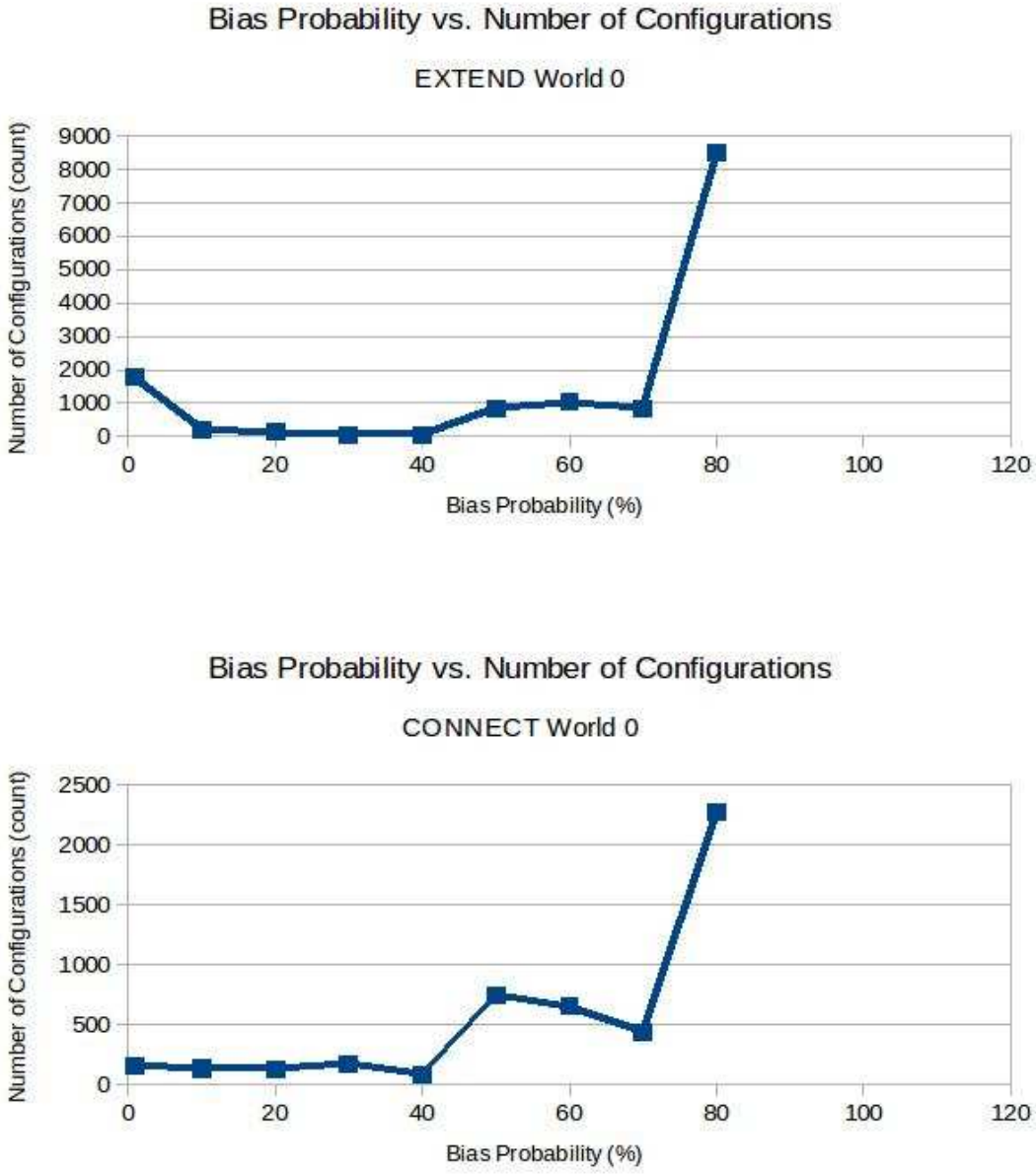


Figure 6.4: Bias Probability vs. Number of Configurations - World 0

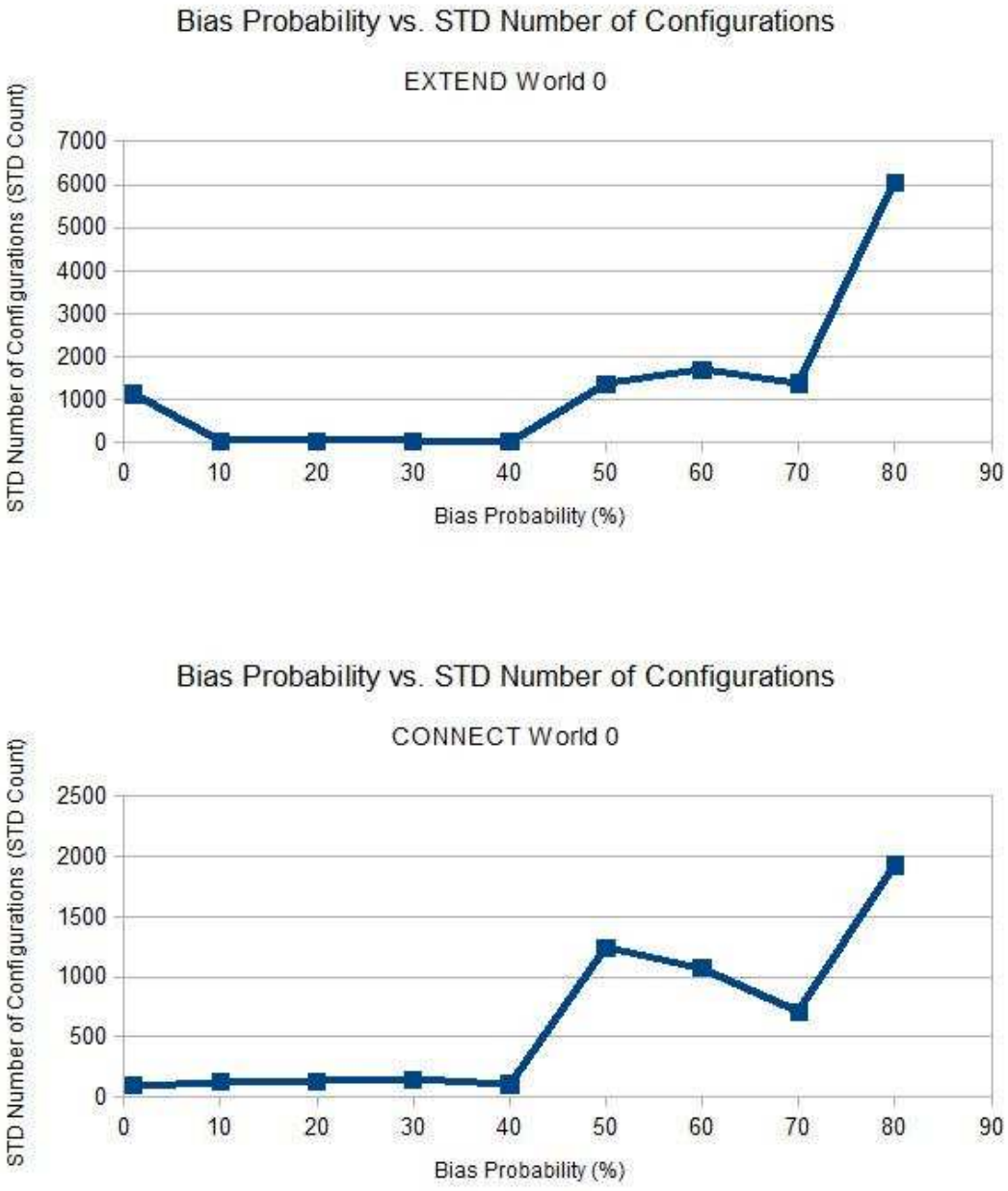


Figure 6.5: Bias Probability vs. STD Number of Configurations - World 0

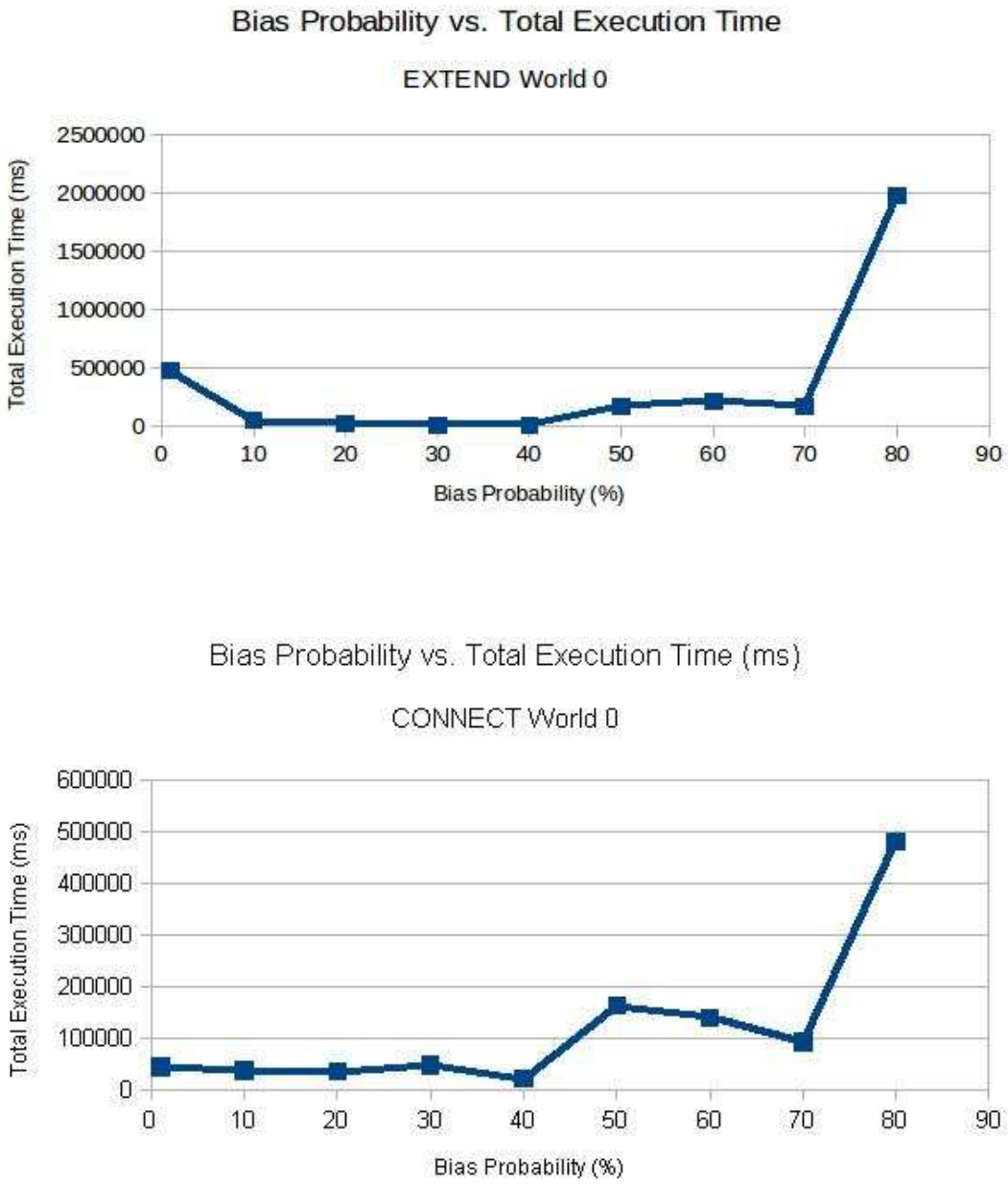


Figure 6.6: Bias Probability vs. Total Execution Time - World 0

For the results of the Sphere Robot in World 0 shown in Table 6.3 there are a few apparent things to note. The first thing that you will notice from the results is that the total move test time accounts for the vast majority of the motion planner total execution time. In fact on average the total move test time is approximately 97% of the motion planner total execution time. Due to the implementation of the move test (IK and trajectory planning solutions), determining if the robot can move between two configurations is slow since the incremental simulator is stepped in real-time. This could easily be solved by stepping the incremental simulator faster than real-time, or if the move test could be done with the models alone by using parametric equations. These improvements will not be considered for this thesis since the IK and trajectory planning solutions are not the focus of this research. This is a common finding across all result tables. However, if we look at the delta between the total execution time and total move test time, then the fastest solution average returned by the motion planner for a specific combination of settings was given in 0.2 seconds.

For the experiments the visualization of the motion plan provided by the Motion Simulator was used to view the plan as it was generated in real-time. The visualization is implemented in OpenGL as described in Section 5.2.3.6 which can use a lot of resources when asked to draw a lot of objects. It is expected that if the visualization was turned off that an improvement in total execution time can be realized.

Secondly, for the results of the Sphere Robot in World 0 shown in Table 6.3 based on the NN Query Time statistic, the NN method is efficient and does not

significantly impact the runtime of the motion planner. The majority of NN queries returned in sub-millisecond time. The NN method is actually so efficient that there is no need to optimize its performance since any gains would be negligible. This is a common finding across all result tables. From these findings it was decided not to tweak NN epsilon which allows for the NN query algorithm to stop when close enough instead of calculating the exact NN. It was also decided to not change the Euclidean distance metric which is used exclusively by the NN method. If the robots used for the experiments had substantially more DOF, then the NN method might have a noticeable impact on the runtime of the motion planner. It is expected that the number of DOF necessary for there to be a noticeable impact on the runtime of the motion planner is an unrealistic amount for a real robot. Determining the threshold for the NN method and the number of DOF is not the focus of this research since it will not be applicable to real robots.

Third, for the results of the Sphere Robot in World 0 shown in Table 6.3 Setting 1 (EXTEND), the majority of the random configurations are modified because of the way the EXTEND algorithm works which was no surprise. The motion towards the random configuration from the NN is only performed for n time steps unlike CONNECT which will complete the motion unless an obstacle is hit or the robot cannot physically perform the motion. The number of modified configurations for EXTEND could potentially be reduced by increasing n. This is also a common finding across all result tables. As n is increased, the EXTEND algorithm will start

to behave more like the CONNECT algorithm.

Fourth, for the results of the Sphere Robot in World 0 shown in Table 6.3 if the goal bias is too large, the RRT can become trapped since exploration of CFREE is low. Also if the goal bias is too small then the RRT takes much longer to converge to the goal configuration. These findings are not a surprise and were expected since Lavalle and Kuffner Jr. [2000] warned of this problem with poor selection of the goal bias probability. This was a common finding across all result tables. The start (small goal bias) and tail end (large goal bias) of the charts in Figures 6.4 and 6.6 illustrate these findings.

For the Sphere Robot in World 0, the best result for CONNECT is shown in Figure 6.2 and the best result for EXTEND is shown in Figure 6.3. The figures show the path and and the underlying tree data structure for the motion planner. The CONNECT algorithm out performed the EXTEND algorithm for this relatively simply case. Since the goal configuration is not very far, adding random configurations to the RRT that are far away from the goal configuration actually causes a loss of performance for the CONNECT algorithm. The EXTEND algorithm does not add random configurations that are very far since motion from NN is only attempted for n time steps before modifying the random configuration so this is actually advantageous in this case.

The tree images clearly illustrate one of the major difference between the EXTEND and CONNECT algorithms. As you can see EXTEND generates longer

branches than CONNECT. For configurations that are far apart, EXTEND must generate many intermediate configurations between the two configurations where as if it is possible to move between the two configurations CONNECT does not require any intermediate configurations. Having the intermediate configurations could be useful if a trajectory planner could use the additional information to refine the trajectory, otherwise due to the wasted time generating intermediate configurations makes CONNECT a better choice than EXTEND.

When the dispersion is low and the number of configurations is also low, then typically the total execution time is small. In this case a solution was found quickly. A low dispersion is not necessarily bad since the goal of the motion planner is to find a solution rather than to do a good job at exploration. If dispersion is low and the number of configurations is high then this would definitely be bad because this would mean that the motion planner is trapped and is not doing a sufficient job of exploring to get out of the local minima. The latter was not observed in any of the experiments conducted.

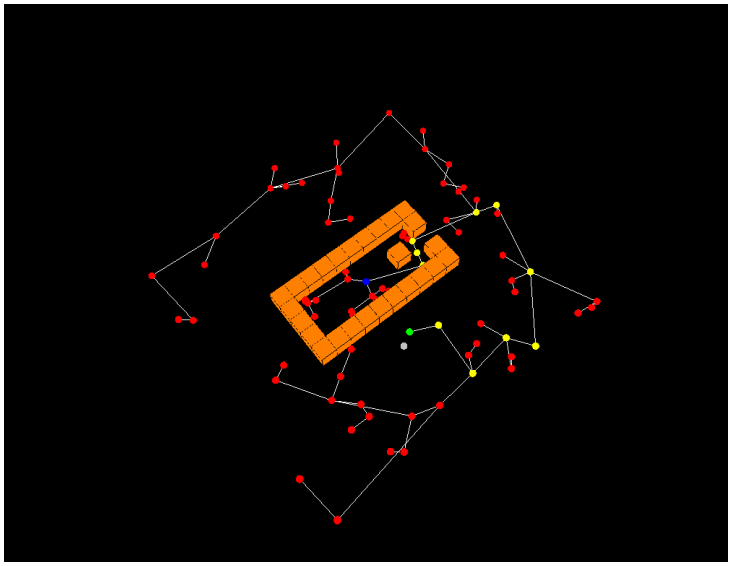
The data and results gathered from the experiments are summarized for the Sphere Robot in World 1 in Table 6.4. Each combination of parameters was run three times with a different seed value.

Table 6.4: Result Matrix World 1

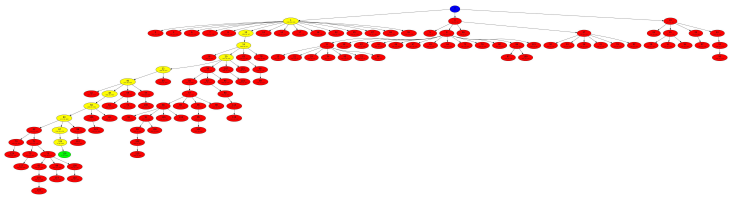
Variant Description	NN k	NN Epsilon	Bias Probability %	Number of Configurations	Modified Random Configurations	Total NN Query Time (ms)	Total Move Test Time (ms)	Total Execution Time (ms)	Dispersion (STD)
EXTEND - Goal Bias	1	0	1	1441	1438	0	386742	401482	1460.63
				3591	3588	0	781045	817874	2348.89
				2236	2233	0	583494	604415	1585.21
EXTEND - Goal Bias	1	0	10	482	479	0	111344	115097	944.936
				543	540	0	125166	129337	1003.02
				291	288	0	73501	75682	710.525
EXTEND - Goal Bias	1	0	20	293	290	0	66992	68926	663.742
				417	414	0	95712	98873	916.749
				266	263	0	62983	64743	688.748
EXTEND - Goal Bias	1	0	30	285	282	0	64988	66679	674.397
				424	421	0	90915	93156	888.885
				4460	4457	0	874789	920853	2124.57
EXTEND - Goal Bias	1	0	40	496	493	0	108331	110901	1004.36
				246	243	0	54068	55433	752.8
				271	268	0	59074	60399	782.925
EXTEND - Goal Bias	1	0	50	235	232	0	49959	50902	496.43
				376	373	0	76788	78520	701.59
				402	399	0	87612	89329	959.242
EXTEND - Goal Bias	1	0	60	793	790	0	168430	171131	1290.05
				416	413	0	84807	86340	709.099
				658	655	0	138991	141262	1305.12
EXTEND - Goal Bias	1	0	70	896	893	0	191032	194132	1312.63
				848	845	0	173516	176175	1117.09
				747	744	0	150418	152667	1295.88
EXTEND - Goal Bias	1	0	80	1692	1626	0	332314	338379	1840.68
				1488	1485	0	306873	312278	1585.38
				1172	1169	0	232723	236185	1078.8
CONNECT - Goal Bias	1	0	1	1317	1016	0	312360	321962	1424.22
				211	128	0	60390	62181	639.057
				127	61	0	40054	41368	508.147
CONNECT - Goal Bias	1	0	10	136	116	0	29843	30762	464.983
				156	116	0	39155	40443	563.782

Table 6.4: Result Matrix World 1

Variant Description	NN k	NN Epsilon	Bias Probability %	Number of Configurations	Modified Random Configurations	Total NN Query Time (ms)	Total Move Test Time (ms)	Total Execution Time (ms)	Dispersion (STD)
				3946	3727	0	803707	842251	1462.42
CONNECT - Goal Bias	1	0	20	232	155	0	62090	64045	644.674
				224	188	1	51465	53087	664.944
				196	96	0	59985	61948	675.925
CONNECT - Goal Bias	1	0	30	230	170	0	57886	59384	681.036
				244	210	0	55065	56730	673.958
				311	198	0	87259	89698	939.74
CONNECT - Goal Bias	1	0	40	246	198	0	58397	59892	622.834
				153	119	0	36857	37754	670.466
				205	176	0	46271	47286	661.358
CONNECT - Goal Bias	1	0	50	1495	1448	0	302583	310522	995.281
				343	308	0	74901	76872	782.053
				231	180	0	55684	56735	843.262
CONNECT - Goal Bias	1	0	60	301	275	0	61999	62526	602.343
				391	354	0	85031	86710	859.166
				407	384	0	84017	85503	728.977
CONNECT - Goal Bias	1	0	70	2288	2154	0	480709	492541	1804.97
				816	746	0	178350	181470	1192.45
				446	415	0	95139	96442	851.054
CONNECT - Goal Bias	1	0	80	591	561	0	125961	127468	899.206
				936	890	0	193664	196416	1124.66
				872	852	0	177707	180131	731.025

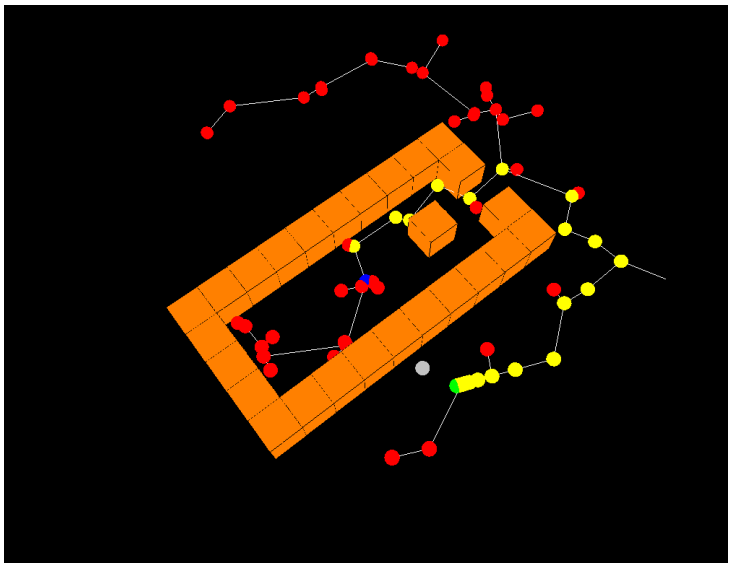


(a) Path



(b) Tree

Figure 6.7: Best CONNECT Result for World 1 - Goal Bias 1



(a) Path

(b) Tree

Figure 6.8: Best EXTEND Result for World 1 - Goal Bias 50

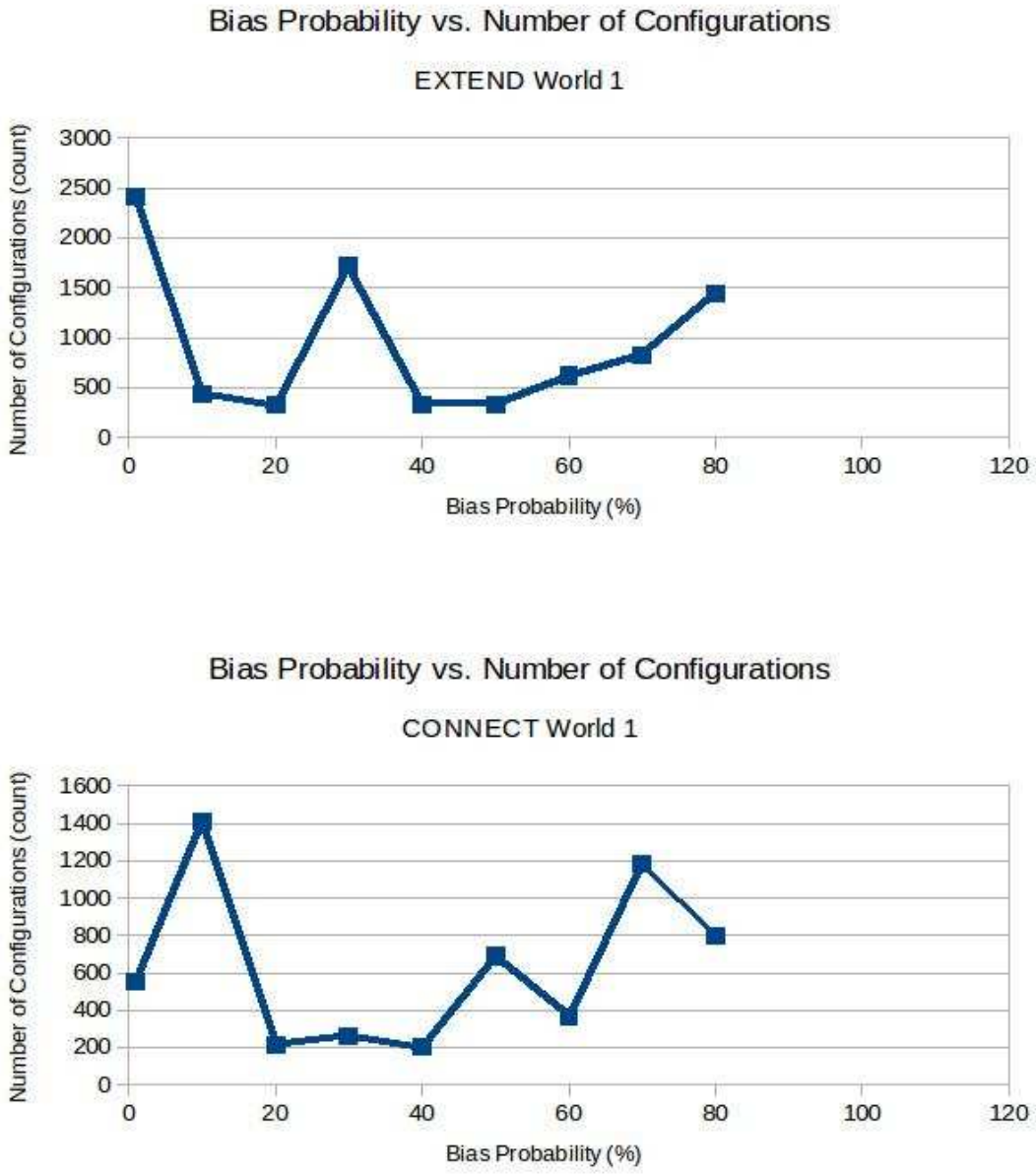


Figure 6.9: Bias Probability vs. Number of Configurations - World 1

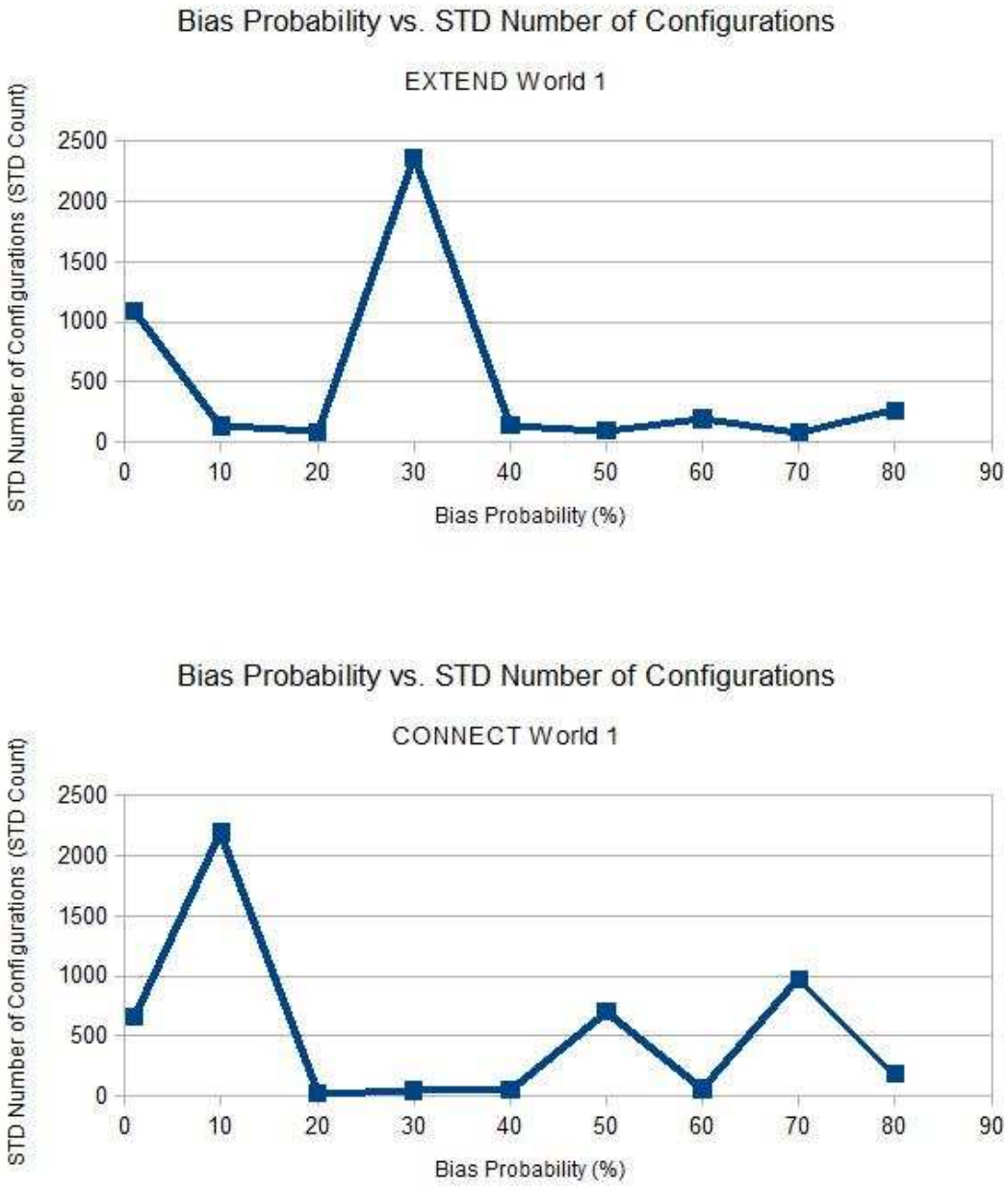


Figure 6.10: Bias Probability vs. STD Number of Configurations - World 1

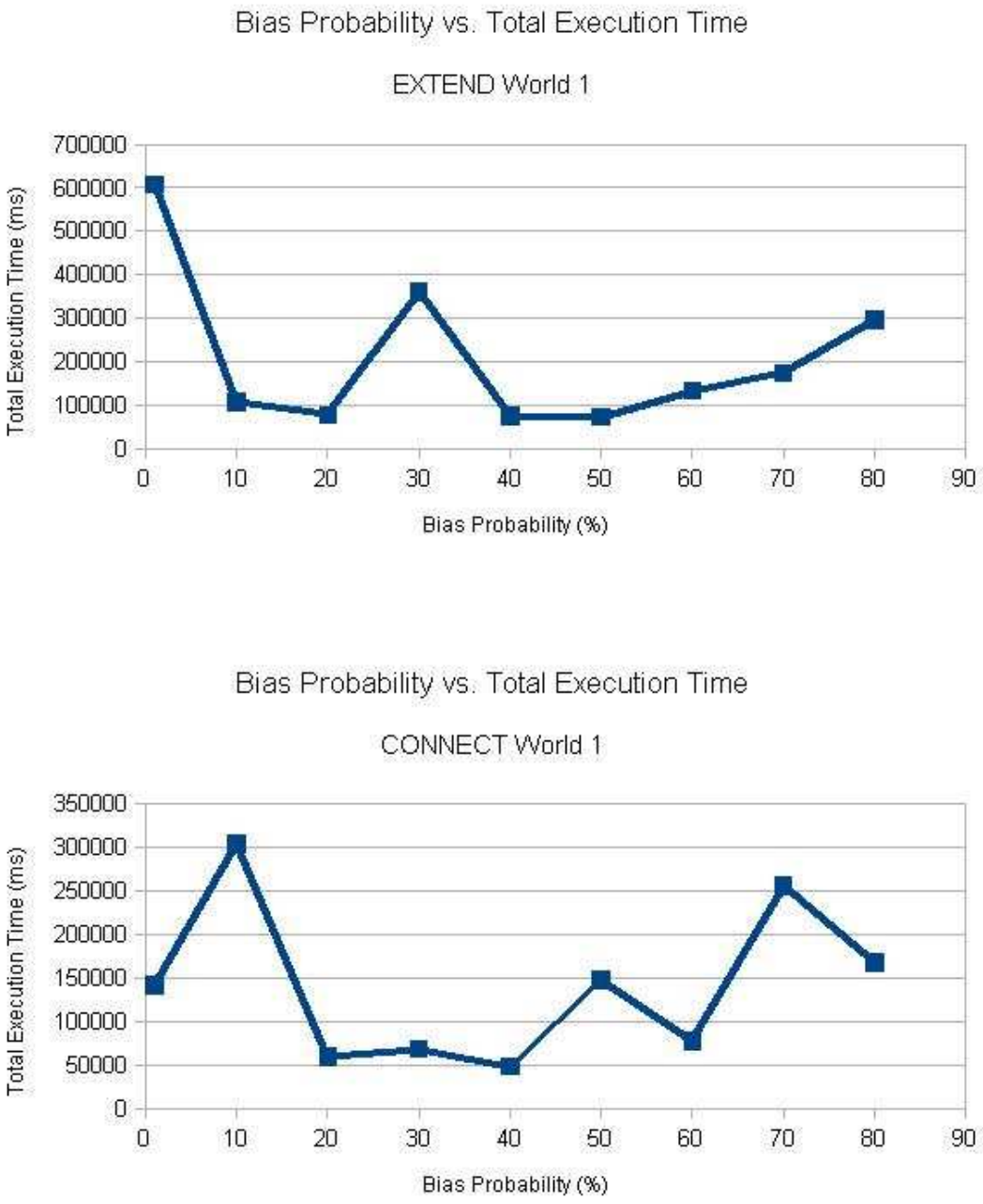


Figure 6.11: Bias Probability vs. Total Execution Time - World 1

For the results of the Sphere Robot in World 1 shown in Table 6.4 if we look at the delta between the total execution time and total move test time, then the fastest solution average returned by the motion planner for a specific combination of settings was given in 1.14 seconds. The best result for CONNECT is shown in Figure 6.7 and the best result for EXTEND is shown in Figure 6.8. Based on visual inspection of the motion plan, I somewhat agree that the motion planner selected the best solution. The motion plan determined is very reasonable although there might be a shorter path distance wise.

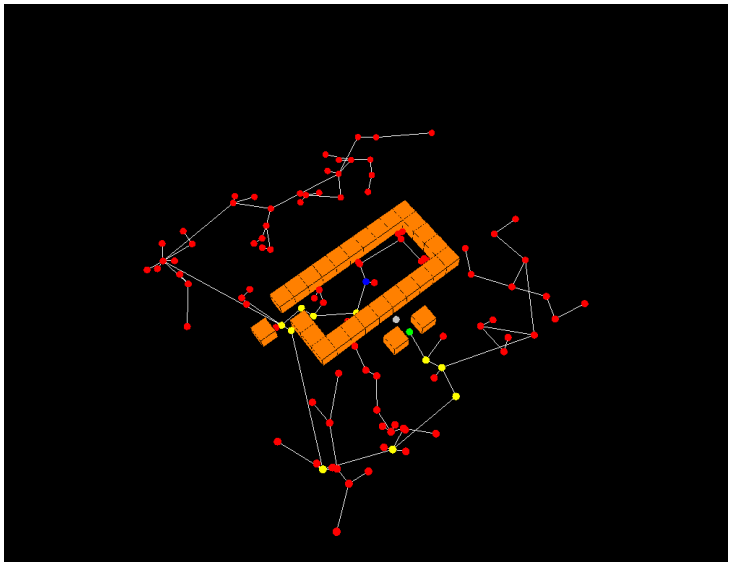
The data and results gathered from the experiments are summarized for the Sphere Robot in World 2 in Table 6.5. Each combination of parameters was run three times with a different seed value. For a few setting combinations the motion planner did not complete in a reasonable time, therefore the run was halted. Runs that were halted are empty rows in the tables.

Table 6.5: Result Matrix World 2

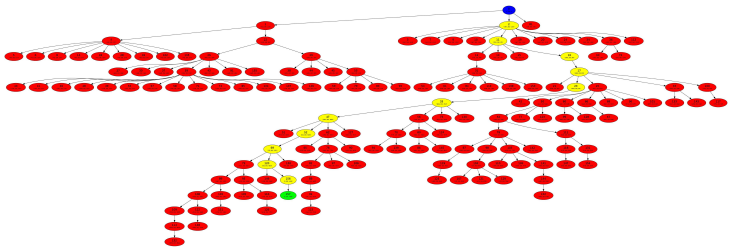
Variant Description	NN k	NN Epsilon	Bias Probability %	Number of Configurations	Modified Random Configurations	Total NN Query Time (ms)	Total Move Test Time (ms)	Total Execution Time (ms)	Dispersion (STD)
EXTEND - Goal Bias	1	0	1	1927	1924	0	564599	578310	1640.53
				1277	1274	0	353197	365301	1233.96
				2690	2687	0	770412	795013	1830.2
EXTEND - Goal Bias	1	0	10	983	980	0	274830	281661	1249.64
				3527	3524	0	960830	1000833	2047.39
				558	555	0	149972	154704	968.395
EXTEND - Goal Bias	1	0	20	603	600	0	147978	152163	1049.96
				3730	3727	0	991025	1031717	2215.76
				4616	4612	0	1316280	1374805	2326.62
EXTEND - Goal Bias	1	0	30	8513	8510	0	2282746	2421398	3101.39
				513	510	0	110949	113982	853.092
				1734	1731	0	405629	419852	1751.97
EXTEND - Goal Bias	1	0	40	316	313	0	69784	71732	612.379
				851	848	0	193042	198049	1425.77
				422	419	0	103146	105652	992.563
EXTEND - Goal Bias	1	0	50	1343	1340	0	310408	317963	1495.02
				1185	1182	0	260120	267266	1580.4
				1233	1230	0	304394	310903	1344.27
EXTEND - Goal Bias	1	0	60	1740	1737	0	392122	400867	1689.19
				753	750	0	163910	167081	1283.86
				7031	7028	0	1686319	1788473	3217.82
EXTEND - Goal Bias	1	0	70	1692	1689	0	370256	377735	1555.84
				942	939	0	207646	211133	1509.29
				3017	3014	0	592677	611175	2364.81
EXTEND - Goal Bias	1	0	80	1625	1622	0	335216	341344	1918.75
				1482	1479	0	316012	321607	1718.19
CONNECT - Goal Bias	1	0	1	158	70	0	67804	69321	539.236
				531	232	0	228947	234862	935.784
				1701	226	0	941298	962741	1388.98
CONNECT - Goal Bias	1	0	10	804	303	0	357603	365754	1149.65
				441	271	0	148428	152371	895.08

Table 6.5: Result Matrix World 2

Variant Description	NN k	NN Epsilon	Bias Probability %	Number of Configurations	Modified Random Configurations	Total NN Query Time (ms)	Total Move Test Time (ms)	Total Execution Time (ms)	Dispersion (STD)
				188	80	0	81335	83108	620.46
CONNECT - Goal Bias	1	0	20	1811	687	0	812347	829796	1633.9
				756	650	0	196864	203338	1081.02
				196	91	0	83111	84431	669.425
CONNECT - Goal Bias	1	0	30	259	169	0	88839	90852	761.986
				556	474	0	139920	141953	964.702
				311	188	0	134432	135109	918.609
CONNECT - Goal Bias	1	0	40	960	538	0	367553	374211	1112.78
				528	387	0	154766	156437	1155.69
				337	206	0	102153	103669	860.911
CONNECT - Goal Bias	1	0	50	205	188	0	49063	50112	350.79
				908	749	0	235673	238656	1444.31
				1435	917	0	428406	435320	1387.98
CONNECT - Goal Bias	1	0	60	1173	978	0	320872	326572	1652.15
				224	212	0	49686	50237	531.481
				634	572	0	157615	160494	1092.13
CONNECT - Goal Bias	1	0	70	1675	1408	0	450756	458494	1792.25
				2179	1641	0	584481	595264	1688.73
				1773	1405	0	513732	522156	1519.47
CONNECT - Goal Bias	1	0	80	1610	1419	0	412989	418934	1405.78
				6783	5645	0	1647341	1754905	2295.83
				2632	2324	0	663111	676931	2033.76

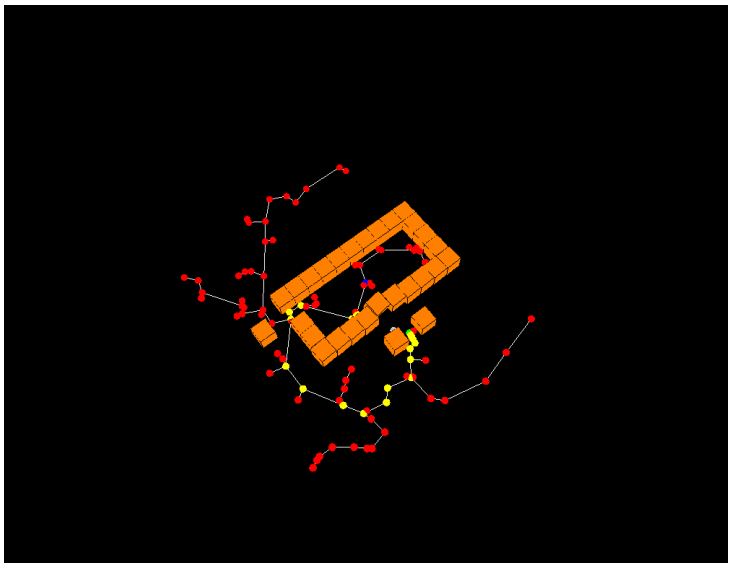


(a) Path



(b) Tree

Figure 6.12: Best CONNECT Result for World 2 - Goal Bias 1



(a) Path

(b) Tree

Figure 6.13: Best EXTEND Result for World 2 - Goal Bias 40

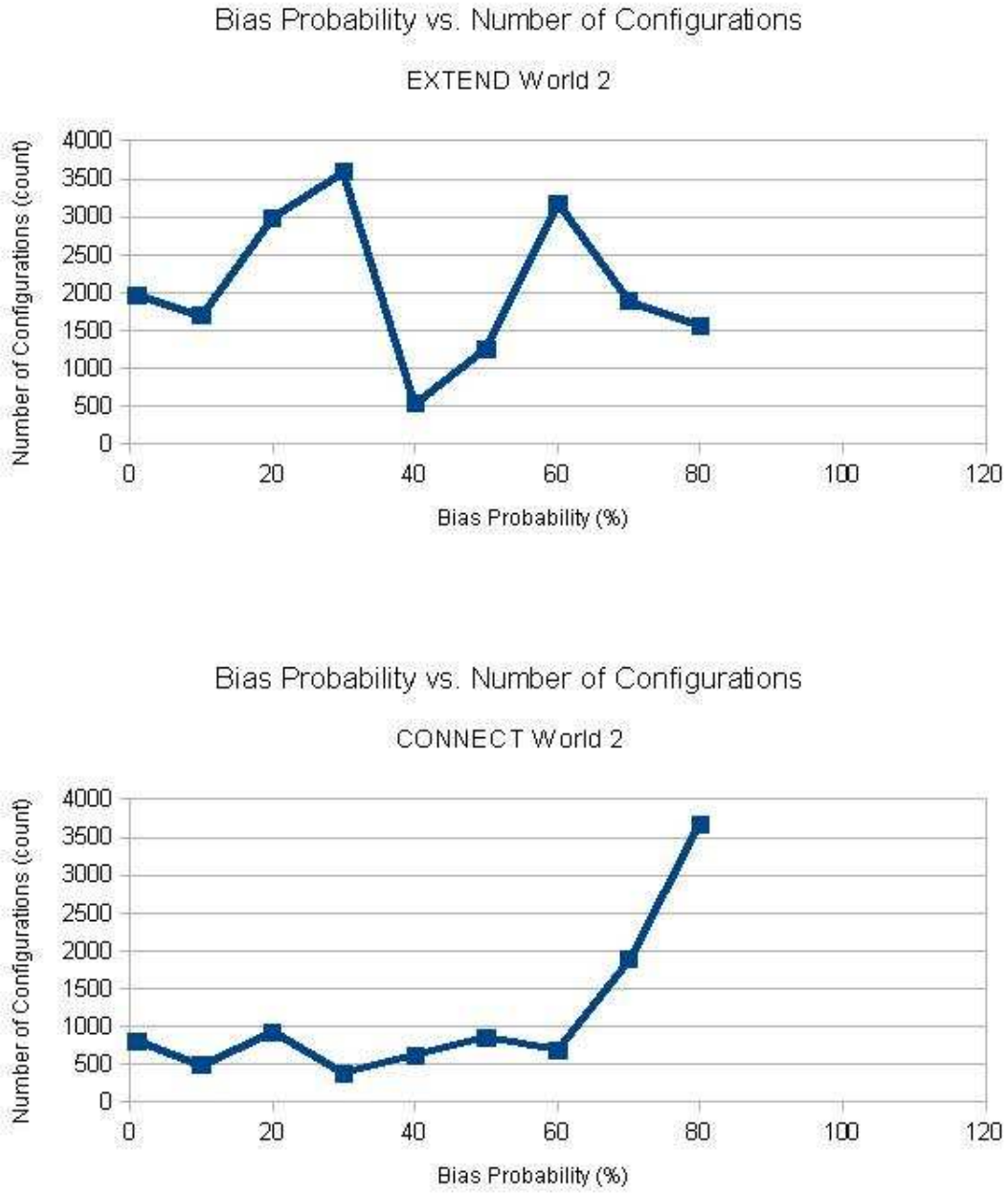


Figure 6.14: Bias Probability vs. Number of Configurations - World 2

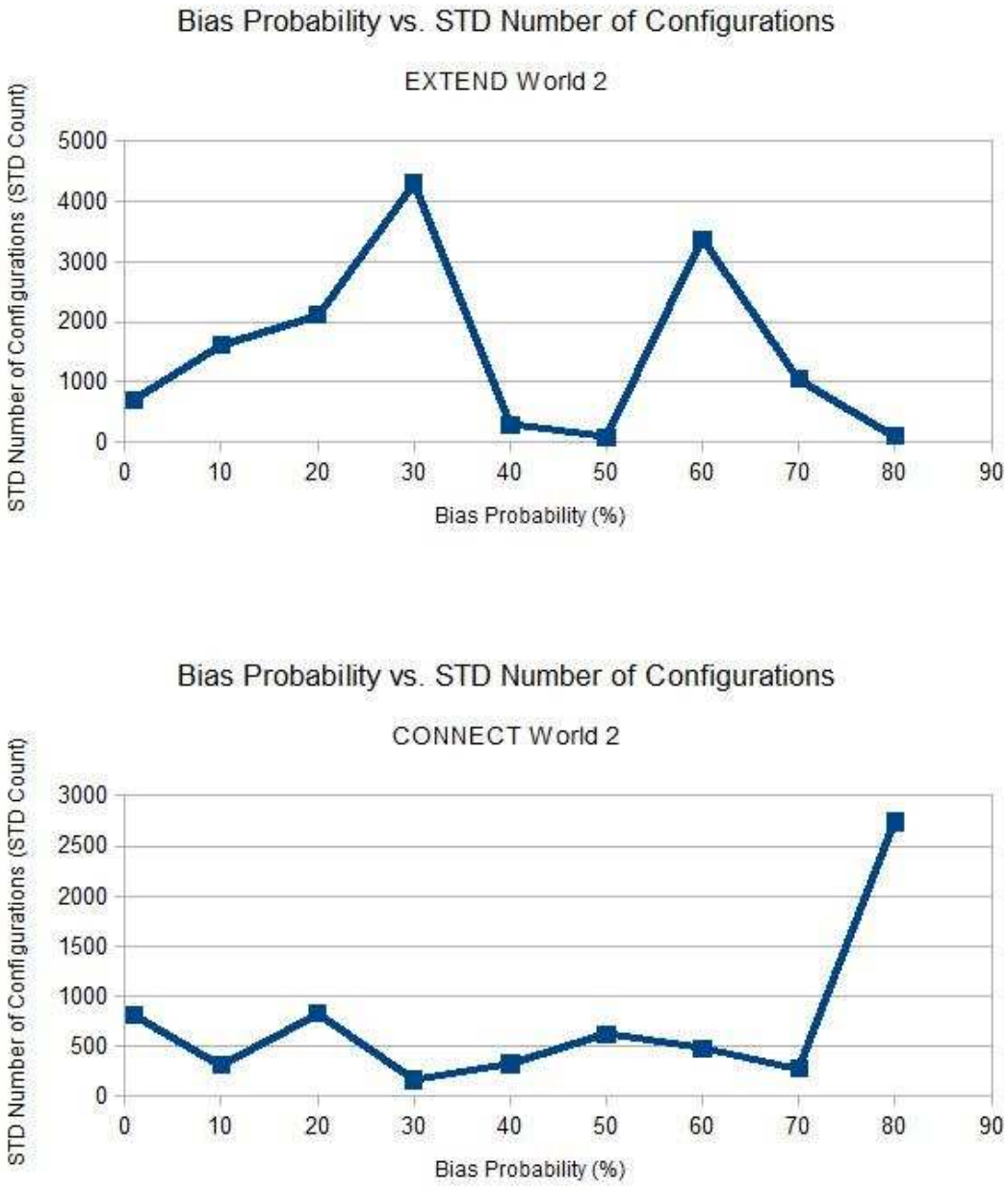


Figure 6.15: Bias Probability vs. STD Number of Configurations - World 2

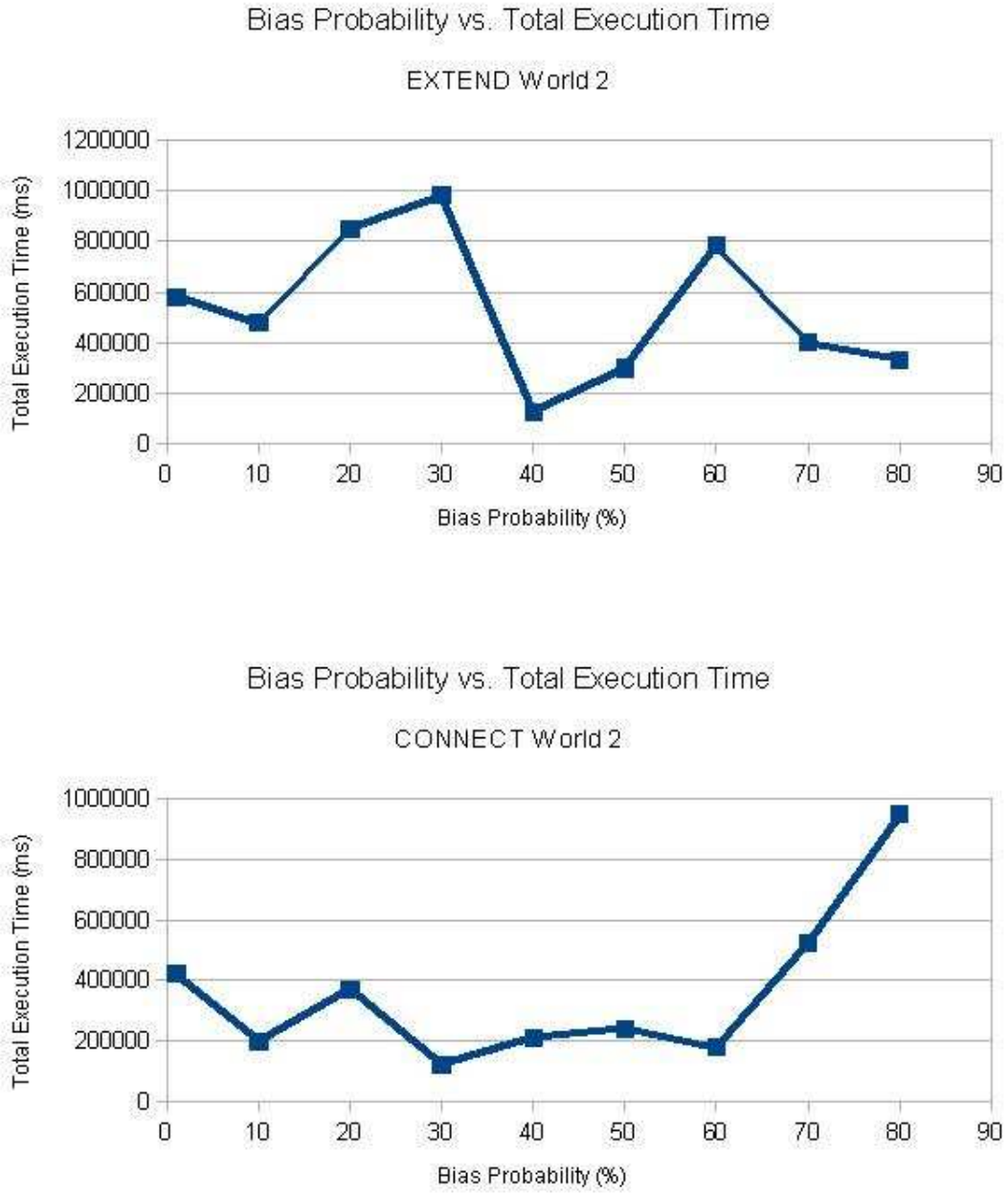


Figure 6.16: Bias Probability vs. Total Execution Time - World 2

For the results of the Sphere Robot in World 2 shown in Table 6.5 if we look at the delta between the total execution time and total move test time, then the fastest solution average returned by the motion planner for a specific combination of settings was given in 1.57 seconds. The best result for CONNECT is shown in Figure 6.12 and the best result for EXTEND is shown in Figure 6.13. Based on visual inspection of the motion plan, I somewhat agree that the motion planner selected the best solution. The motion plan determined is very reasonable although there is definitely a shorter path distance wise.

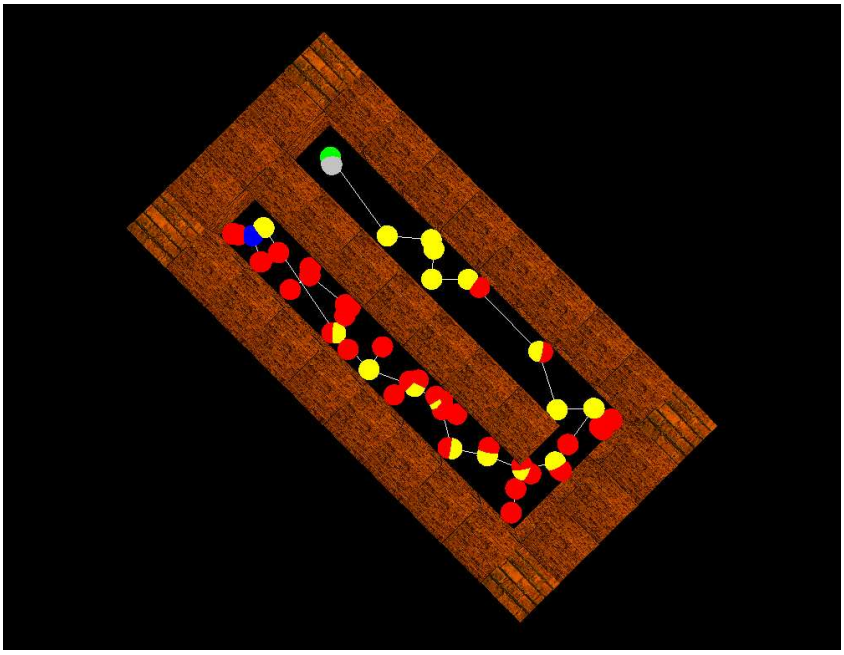
The data and results gathered from the experiments are summarized for the Sphere Robot in World 3 in Table 6.6. Each combination of parameters was run three times with a different seed value. For a few setting combinations the motion planner did not complete in a reasonable time, therefore the run was halted. Runs that were halted are empty rows in the tables.

Table 6.6: Result Matrix World 3

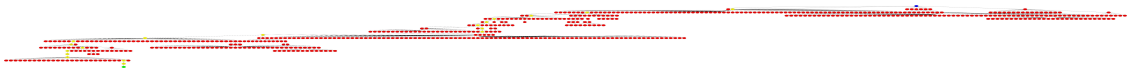
Variant Description	NN k	NN Epsilon	Bias Probability %	Number of Configurations	Modified Random Configurations	Total NN Query Time (ms)	Total Move Test Time (ms)	Total Execution Time (ms)	Dispersion (STD)
EXTEND - Goal Bias	1	0	1	4791	4788	0	1393327	1454455	3210.59
				4453	4450	0	918780	965864	3189.55
				4888	4885	0	1426564	1490332	3239.18
EXTEND - Goal Bias	1	0	10	2240	2237	1	647304	664623	2384.11
				1847	1844	0	547246	559420	2108.02
				7081	7078	0	1505483	1617548	4243.29
EXTEND - Goal Bias	1	0	20	4306	4303	0	1090347	1127735	3514.03
				3362	3359	0	972224	1005263	3167.74
				2036	2033	0	421154	437392	2375.21
EXTEND - Goal Bias	1	0	30	3866	3863	0	788237	821961	3440.94
				5714	5711	0	1662769	1743867	4302.2
				4525	4522	0	1356105	1408841	3736.23
EXTEND - Goal Bias	1	0	40	2814	2811	0	578709	596544	3075.52
				4836	4833	0	1429723	1486562	4097.51
				4477	4474	3	1314780	1359004	3870.25
EXTEND - Goal Bias	1	0	50	6638	6635	0	1362164	1462168	4864.71
				7243	7240	1	2123256	2253969	5189.71
				4652	4649	1	986375	1034355	4061.29
EXTEND - Goal Bias	1	0	60	8259	8256	0	1689123	1833903	5590.09
				10791	10788	13	3222552	3500271	6490.12
				12119	12116	11	2552199	2897153	6743.04
EXTEND - Goal Bias	1	0	70	9285	9282	947	1974268	2155907	6104.2
				7240	7237	32	2100898	2217897	5112.17
				12141	12138	4358	3495575	3845080	6872.55
EXTEND - Goal Bias	1	0	80	5637	5634	0	1174970	1239016	4498.92
				13928	13925	84	4108394	4559232	7123.6
CONNECT - Goal Bias	1	0	1	1441	1415	0	421289	432202	1812.48
				605	589	0	150185	155075	1206.86
				2683	2620	0	756680	781588	2353.8
CONNECT - Goal Bias	1	0	10	1674	1630	0	513090	524956	2029.33
				1508	1479	352	321812	349538	1952.34

Table 6.6: Result Matrix World 3

Variant Description	NN k	NN Epsilon	Bias Probability %	Number of Configurations	Modified Random Configurations	Total NN Query Time (ms)	Total Move Test Time (ms)	Total Execution Time (ms)	Dispersion (STD)
				770	753	0	237270	242153	1507.15
CONNECT - Goal Bias	1	0	20	2527	2483	0	535368	564234	2758.45
				1796	1773	0	532416	544710	2304.08
				4439	4367	0	1369492	1419076	3534.77
CONNECT - Goal Bias	1	0	30	1941	1908	0	415350	433658	2440.61
				3766	3721	0	1142527	1179000	3505.61
				2117	2078	0	632405	645988	2602.36
CONNECT - Goal Bias	1	0	40	2095	2067	0	440815	459103	2688.72
				3385	3352	0	1021308	1049778	3439.83
				2285	2247	146	681814	714964	2803.69
CONNECT - Goal Bias	1	0	50	4583	4543	0	954791	1004239	4064.97
				4345	4301	0	1296818	1340262	4054.75
				3265	3221	0	972981	998966	3420.71
CONNECT - Goal Bias	1	0	60	4347	4314	0	893586	935857	4074.48
				8520	8453	1	2471712	2631438	5803.1
				11847	11759	2464	3469693	3789498	6721.8
CONNECT - Goal Bias	1	0	70	8772	8718	3	1833366	1994688	5874
				3808	3779	0	906855	945554	3765.42
				6298	6260	5	1357805	1436843	4992.4
CONNECT - Goal Bias	1	0	80	5610	5583	8	1154007	1217548	4464.94
				6758	6728	252	1407238	1505300	4781.29
				7064	7033	0	1517848	1616480	5174.36

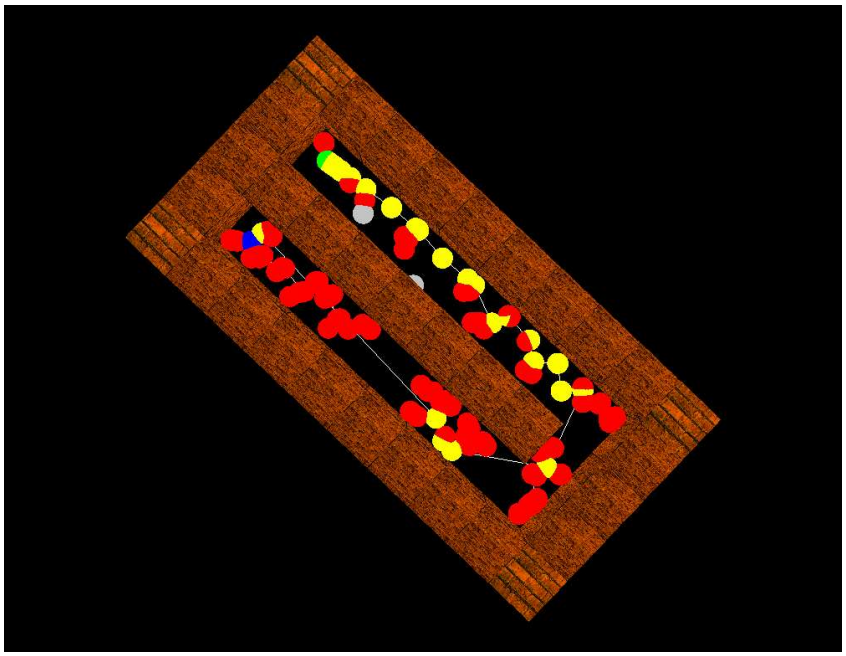


(a) Path



(b) Tree

Figure 6.17: Best CONNECT Result for World 3 - Goal Bias 1



(a) Path

(b) Tree

Figure 6.18: Best EXTEND Result for World 3 - Goal Bias 10

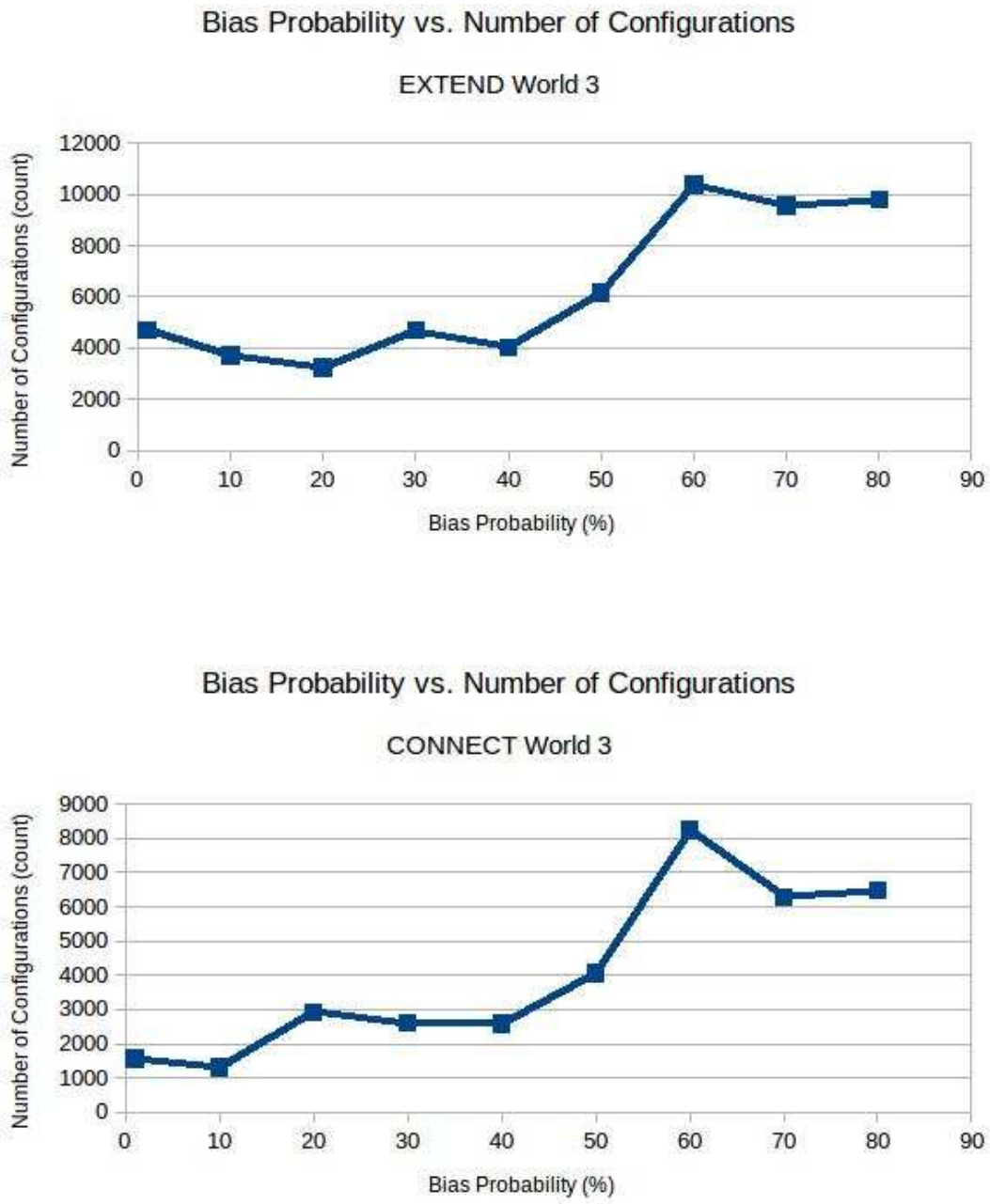


Figure 6.19: Bias Probability vs. Number of Configurations - World 3

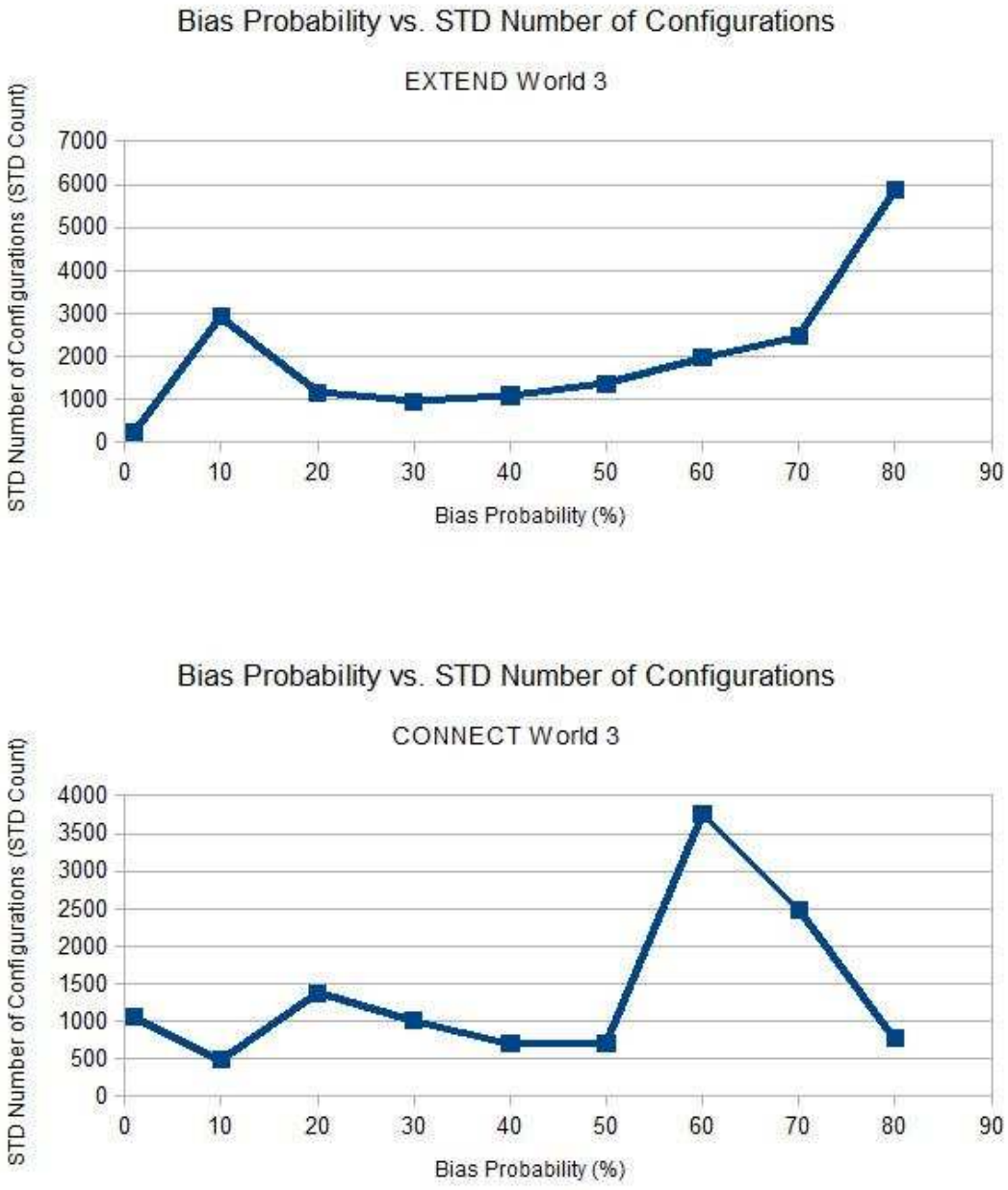


Figure 6.20: Bias Probability vs. STD Number of Configurations - World 3

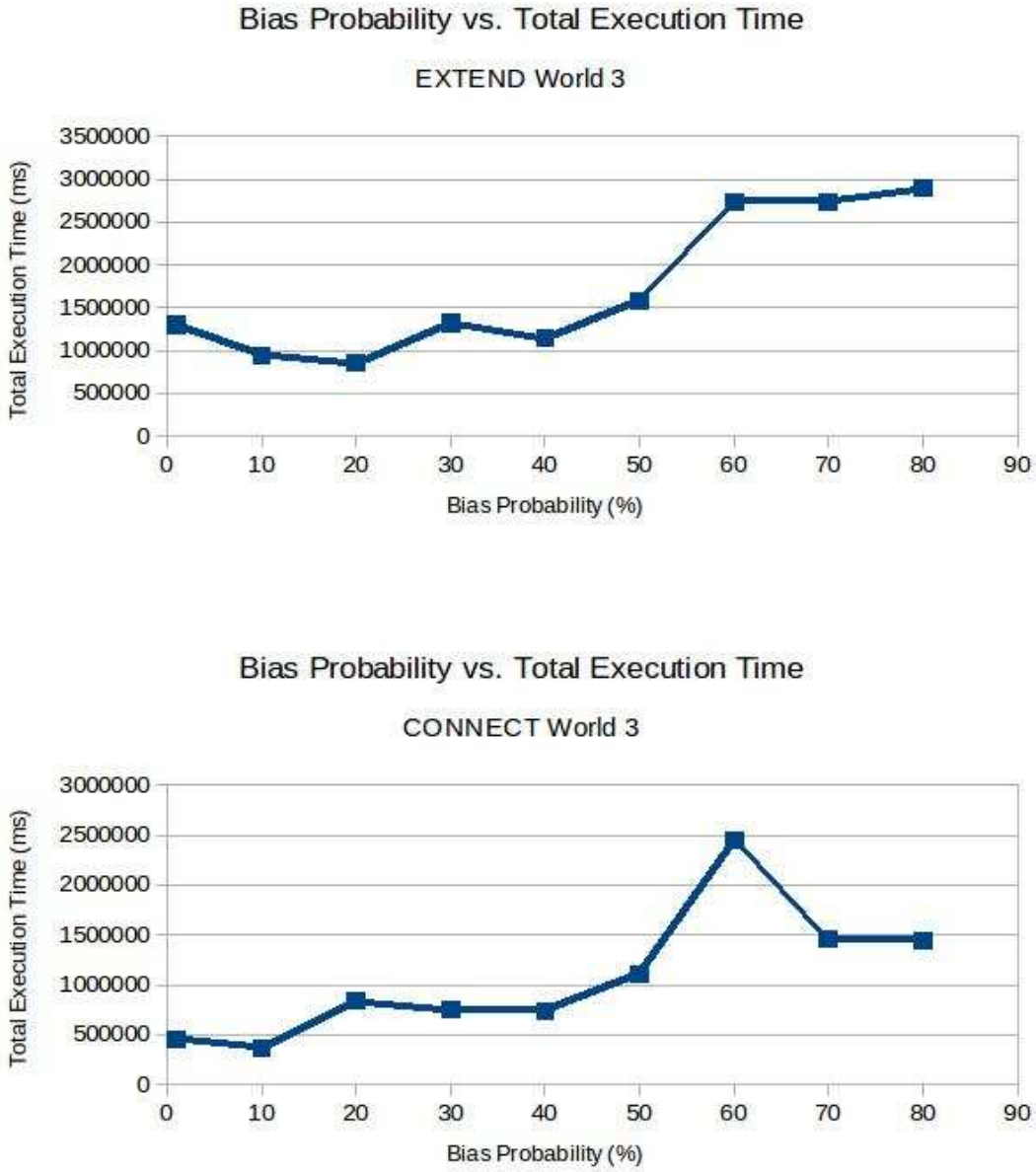


Figure 6.21: Bias Probability vs. Total Execution Time - World 3

For the results of the Sphere Robot in World 3 shown in Table 6.6 if we look at the delta between the total execution time and total move test time, then the fastest solution average returned by the motion planner for a specific combination of settings was given in 13.57 seconds. The best result for CONNECT is shown in Figure 6.17 and the best result for EXTEND is shown in Figure 6.18. Based on visual inspection of the motion plan, I somewhat disagree that the motion planner selected the best solution. The motion plan uses many intermediate configurations that appear to be superfluous. The path however, does not seem unreasonable.

Figures 6.5, 6.10, 6.15, 6.20, 6.25 show the variability of the number of configurations with the same settings but a different RNG seed. It is desirable for the motion planner to return results in approximately the same amount of time when the same settings are used, while a large variability with the same settings is undesirable. This may not be completely unavoidable since a motion planner is probabilistic and will converge to a solution eventually if one exists, but a potential workaround if the motion planner is taking longer than expected to find a solution is to start the motion planner again with a different RNG seed value because there is a chance that it will converge quicker.

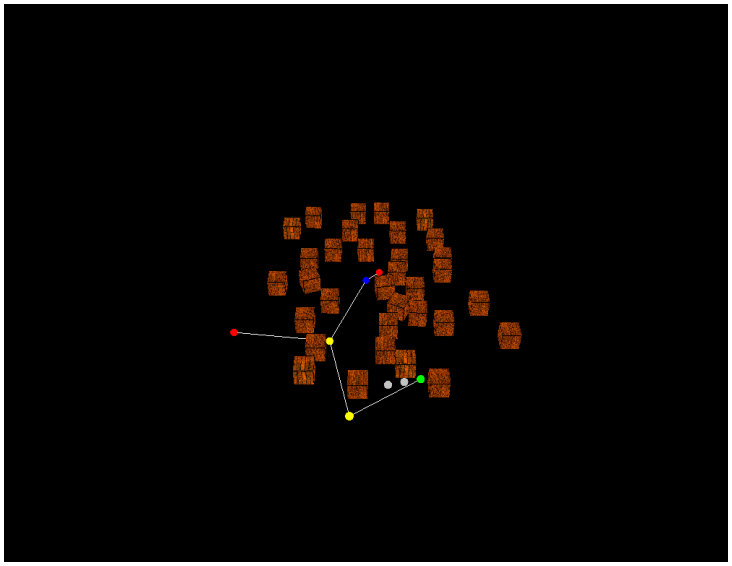
The data and results gathered from the experiments are summarized for the Sphere Robot in World 4 in Table 6.7. Each combination of parameters was run three times with a different seed value.

Table 6.7: Result Matrix World 4

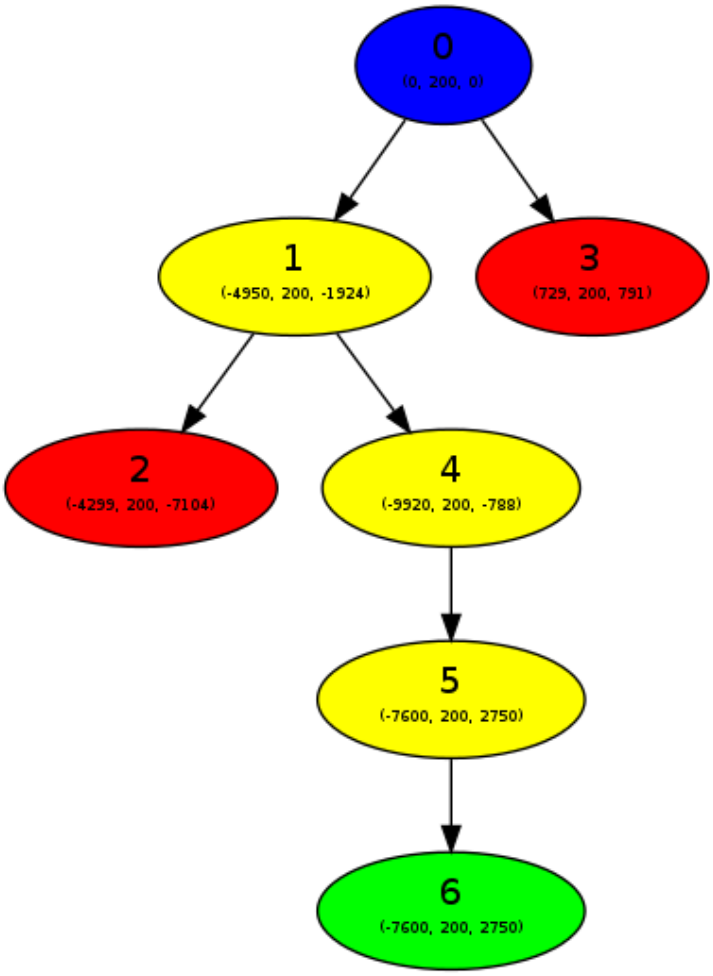
Variant Description	NN k	NN Epsilon	Bias Probability %	Number of Configurations	Modified Random Configurations	Total NN Query Time (ms)	Total Move Test Time (ms)	Total Execution Time (ms)	Dispersion (STD)
EXTEND - Goal Bias	1	0	1	3963	3960	0	824697	859393	2088.88
				1447	1444	0	409973	423237	1246.48
				2302	2299	0	646889	669420	1548.02
EXTEND - Goal Bias	1	0	10	913	910	0	186730	192349	1029.24
				1226	1223	0	329085	339941	1182.24
				558	555	0	149654	154282	735.654
EXTEND - Goal Bias	1	0	20	382	379	0	77303	79240	638.795
				2280	2277	0	607567	626587	1450.88
				127	124	0	31937	32974	325.558
EXTEND - Goal Bias	1	0	30	4886	4883	1	988362	1036415	1971.58
				580	577	0	145070	148908	870.231
				110	107	0	25834	26483	307.235
EXTEND - Goal Bias	1	0	40	2860	2856	0	711878	734884	1464.04
				173	170	0	40744	41856	504.849
				112	109	0	26636	27292	306.598
EXTEND - Goal Bias	1	0	50	81	78	0	19423	19724	333.142
				1198	1195	0	288518	294947	926.153
				110	107	0	25031	25483	296.933
EXTEND - Goal Bias	1	0	60	77	74	0	17720	17962	331.317
				145	142	0	32536	33139	282.664
				558	555	0	128948	131210	659.662
EXTEND - Goal Bias	1	0	70	653	650	0	141954	143724	550.299
				254	251	0	56966	57779	451.731
				249	246	0	54055	54770	492.452
EXTEND - Goal Bias	1	0	80	1477	1474	0	317346	322587	608.874
				4288	4285	0	931800	966986	1066.41
				722	719	0	152673	154584	888.461
CONNECT - Goal Bias	1	0	1	373	113	0	126758	129380	758.364
				605	198	0	201235	205702	863.154
				27	11	0	8017	8166	181.067
CONNECT - Goal Bias	1	0	10	172	91	0	48992	49781	557.471
				633	217	0	209138	213275	843.756

Table 6.7: Result Matrix World 4

Variant Description	NN k	NN Epsilon	Bias Probability %	Number of Configurations	Modified Random Configurations	Total NN Query Time (ms)	Total Move Test Time (ms)	Total Execution Time (ms)	Dispersion (STD)
				9	3	0	2404	2428	153.781
CONNECT - Goal Bias	1	0	20	160	123	0	37056	37613	456.416
				306	181	0	84150	85627	674.715
				7	2	0	1805	1822	124.328
CONNECT - Goal Bias	1	0	30	282	153	0	80151	81721	604.667
				825	432	0	239486	243534	1023.47
				1056	514	0	315427	321494	882.456
CONNECT - Goal Bias	1	0	40	465	299	0	122317	124038	708.824
				290	195	0	74513	75758	630.477
				437	253	0	120910	122858	604.921
CONNECT - Goal Bias	1	0	50	100	83	0	21929	22233	416.414
				195	148	0	46164	46851	545.295
				774	509	0	203112	205861	707.433
CONNECT - Goal Bias	1	0	60	158	145	0	31236	31672	413.368
				3294	2283	0	841085	864036	1244.36
				568	415	0	139401	141055	817.65
CONNECT - Goal Bias	1	0	70	176	166	0	34440	34769	434.598
				3716	2869	0	882556	910428	1281.06
				926	720	0	217943	220658	862.998
CONNECT - Goal Bias	1	0	80	478	444	0	97219	98073	374.052
				230	216	0	45254	45632	248.334
				658	610	0	133464	134785	880.067

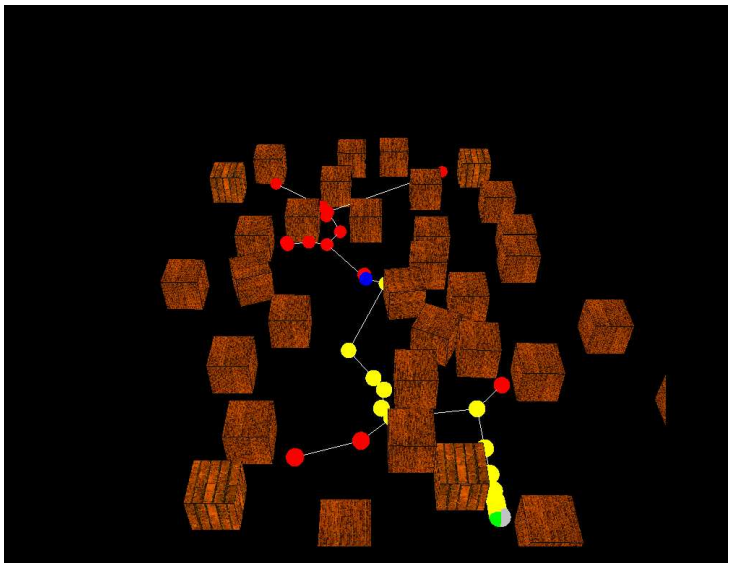


(a) Path

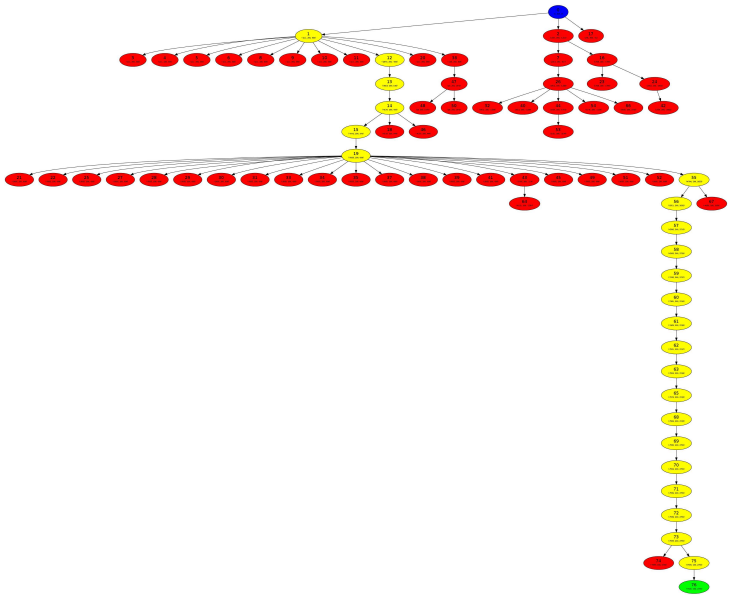


(b) Tree

Figure 6.22: Best CONNECT Result for World 4 - Goal Bias 60



(a) Path



(b) Tree

Figure 6.23: Best EXTEND Result for World 4 - Goal Bias 20

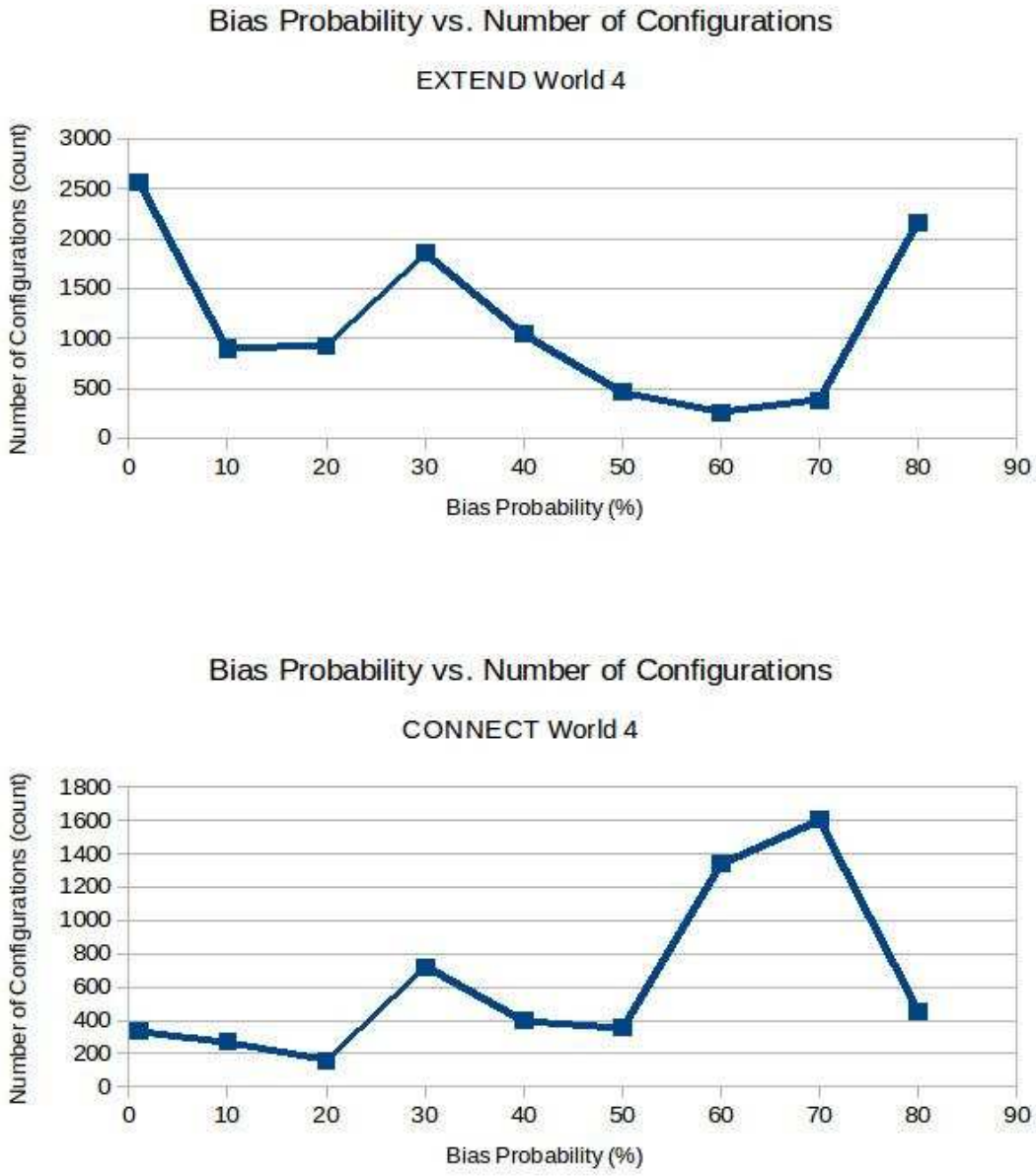


Figure 6.24: Bias Probability vs. Number of Configurations - World 4

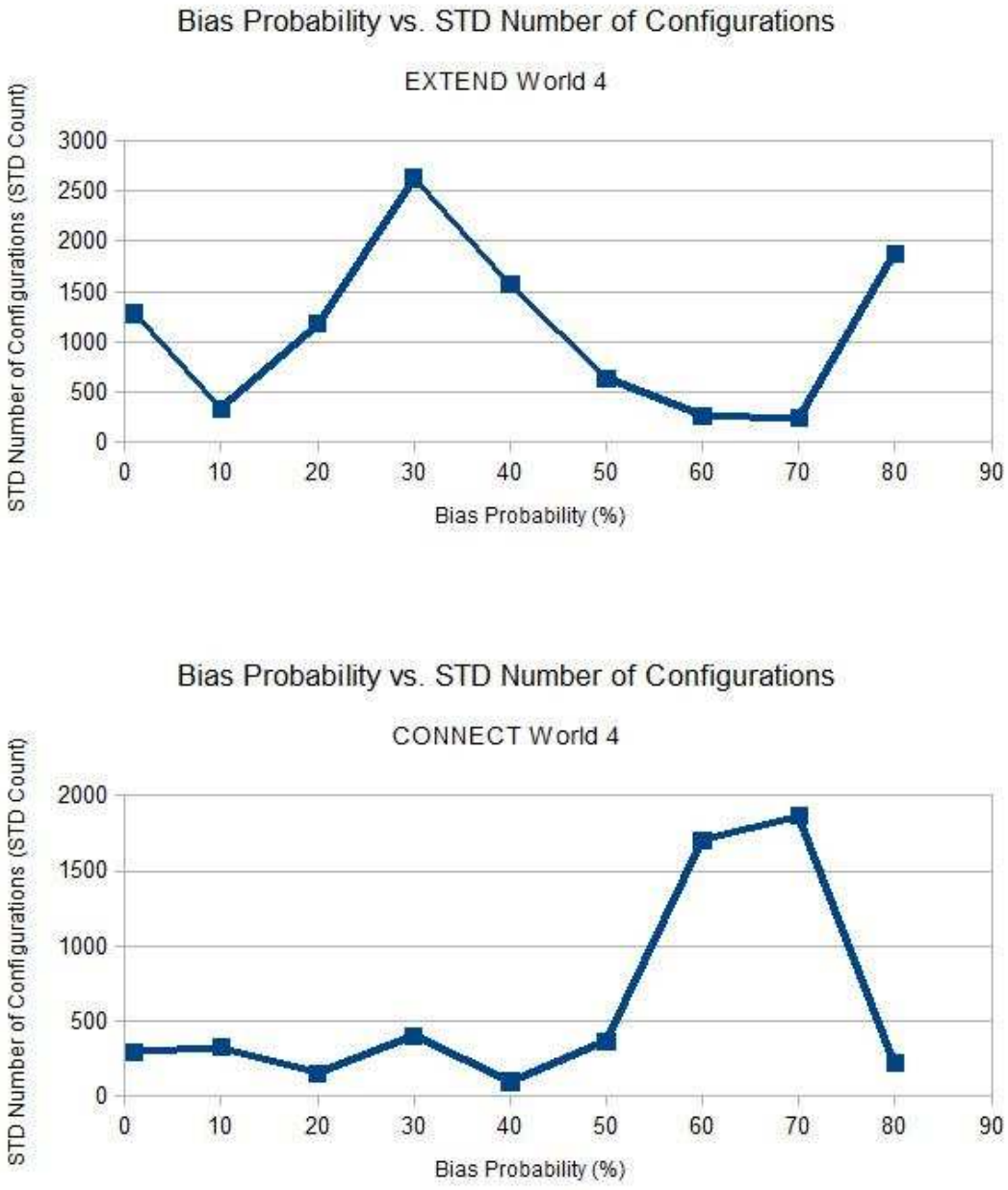


Figure 6.25: Bias Probability vs. STD Number of Configurations - World 4

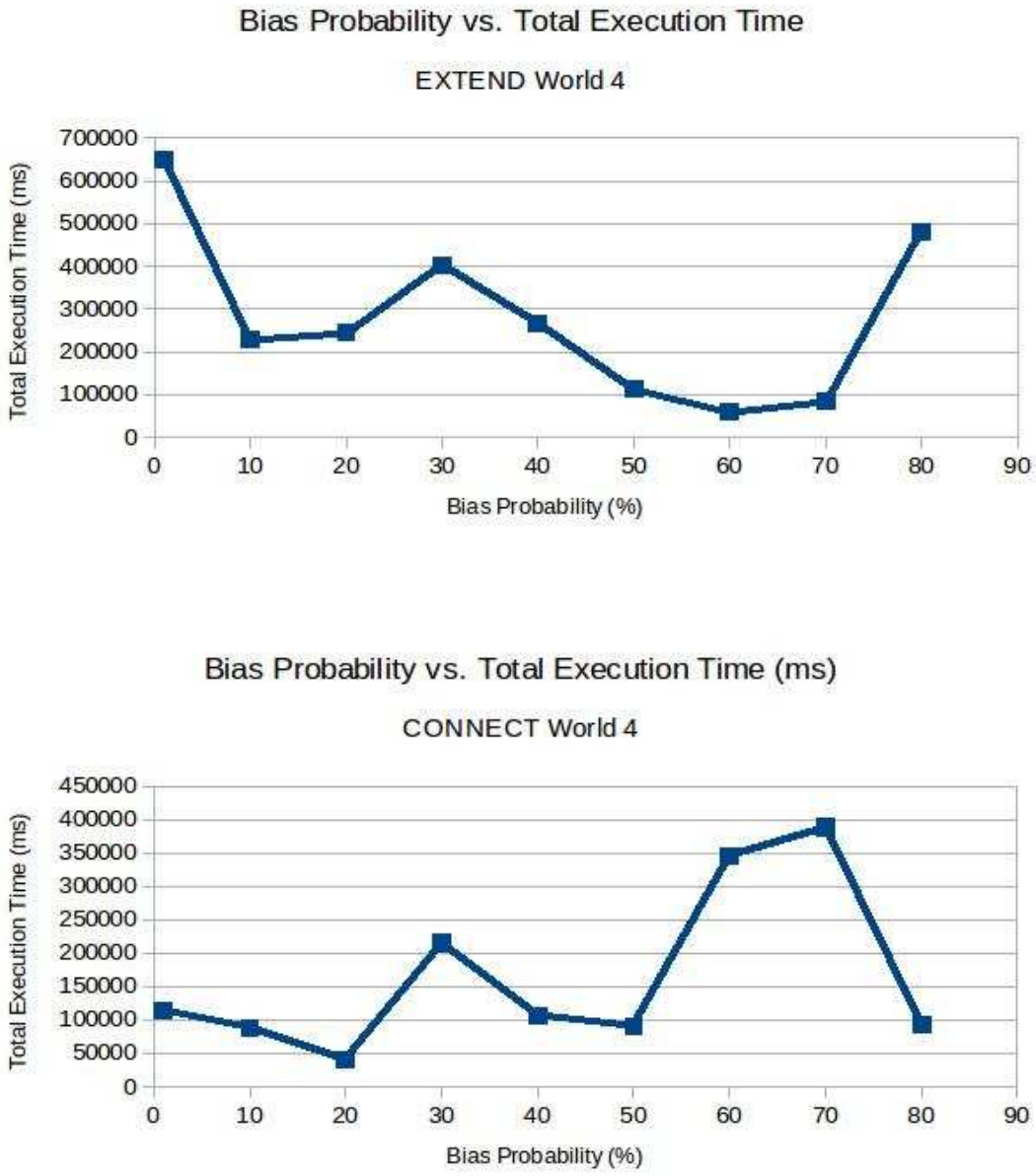


Figure 6.26: Bias Probability vs. Total Execution Time - World 4

For the results of the Sphere Robot in World 4 shown in Table 6.7 if we look at the delta between the total execution time and total move test time, then the fastest solution average returned by the motion planner for a specific combination of settings was given in 0.68 seconds. The best result for CONNECT is shown in Figure 6.22 and the best result for EXTEND is shown in Figure 6.23. Based on visual inspection of the motion plan, I somewhat agree that the motion planner selected the best solution. The motion plan determined is very reasonable although there might be a shorter path distance wise.

The data and results gathered from the experiments are summarized for the Humanoid Robot in World 5 in Table 6.8. Each combination of parameters was run once due to the amount of time it takes to execute the plan on the real robot.

Table 6.8: Result Matrix World 5

Variant Description	NN k	NN Epsilon	Bias Probability %	Number of Configurations	Modified Random Configurations	Total NN Query Time (ms)	Total Move Test Time (ms)	Total Execution Time (ms)	Dispersion (STD)
EXTEND - Goal Bias	1	0	1	1974	1970	0	529837	550030	2001.0631
EXTEND - Goal Bias	1	0	10	744	740	0	171477	177192	1374.1374
EXTEND - Goal Bias	1	0	20	364	360	0	86287	88698	943.58476
EXTEND - Goal Bias	1	0	30	581	577	0	124554	127624	1217.77245
EXTEND - Goal Bias	1	0	40	680	675	0	148413	151934	1375.9732
EXTEND - Goal Bias	1	0	50	551	547	0	87612	122381	1314.16154
EXTEND - Goal Bias	1	0	60	901	897	0	190418	193529	1788.0144
EXTEND - Goal Bias	1	0	70	1228	1223	0	261714	265961	1798.3031
EXTEND - Goal Bias	1	0	80	2039	2034	0	420416	427821	2171.9706
CONNECT - Goal Bias	1	0	1	174	84	0	54874	56674	696.16139
CONNECT - Goal Bias	1	0	10	214	159	0	53642	55407	772.38134
CONNECT - Goal Bias	1	0	20	318	212	0	85063	87742	883.20338
CONNECT - Goal Bias	1	0	30	315	233	0	79304	81356	933.01932
CONNECT - Goal Bias	1	0	40	210	163	0	50494	51723	918.53842
CONNECT - Goal Bias	1	0	50	316	247	0	76287	77727	1155.26894
CONNECT - Goal Bias	1	0	60	536	485	0	116492	118793	1177.05742
CONNECT - Goal Bias	1	0	70	611	569	0	130340	132126	1165.94398
CONNECT - Goal Bias	1	0	80	1282	1219	0	265320	269090	1540.7842

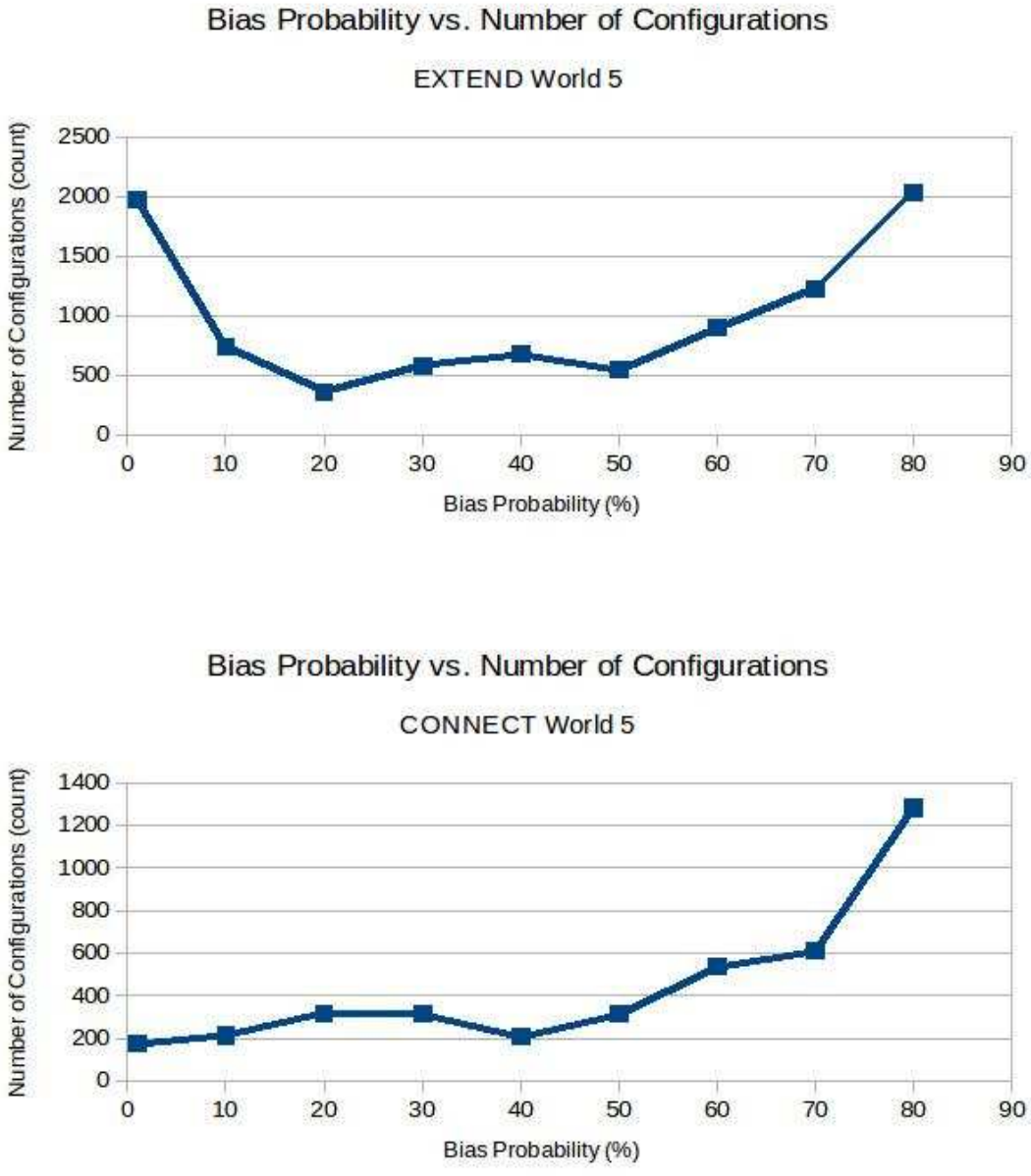


Figure 6.27: Bias Probability vs. Number of Configurations - World 5

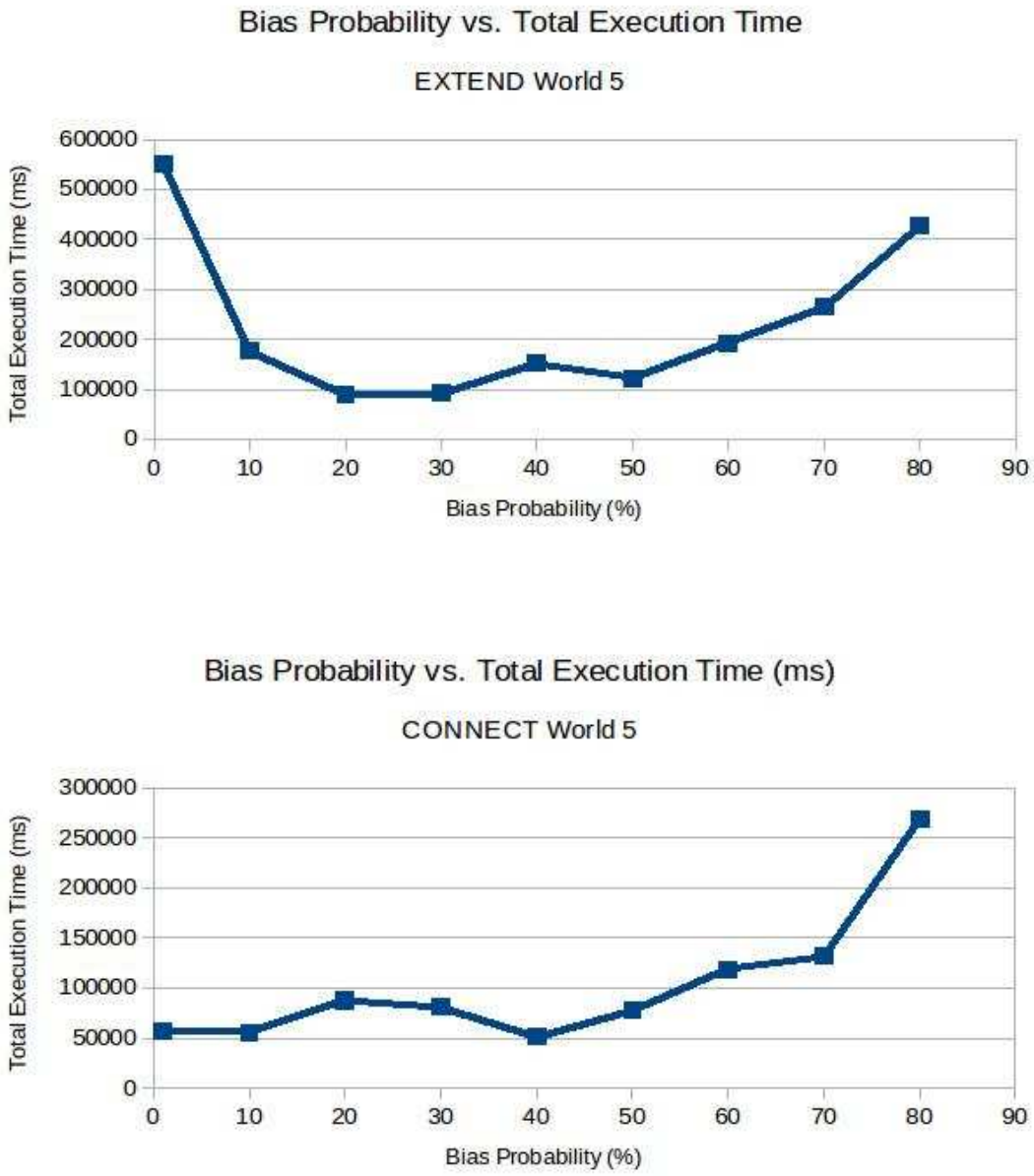
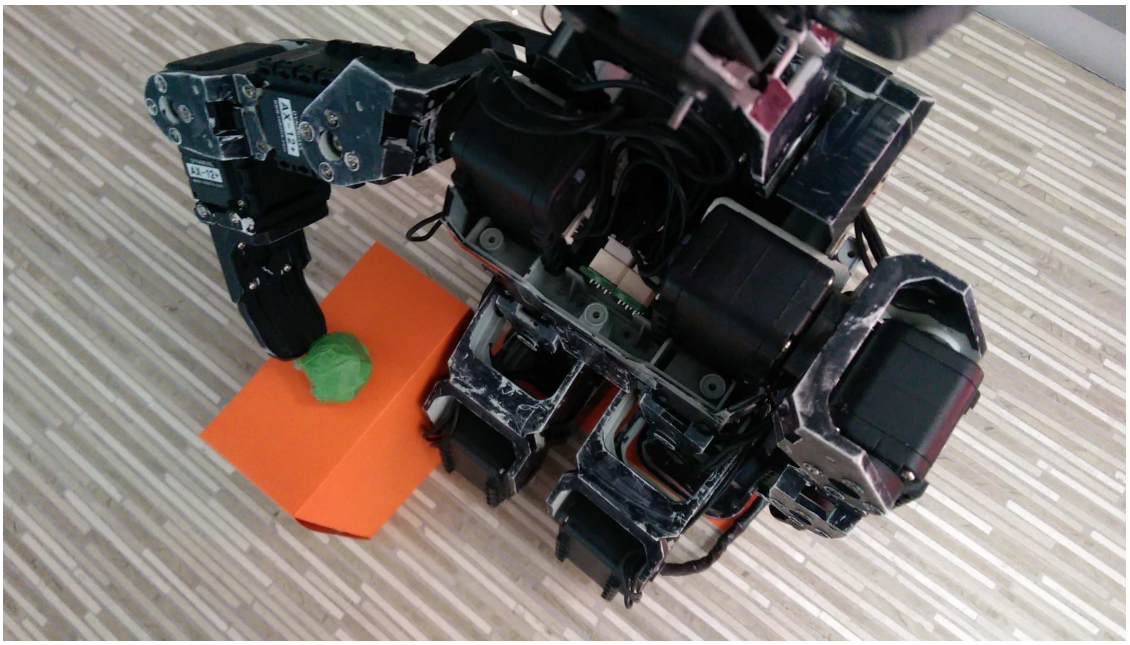


Figure 6.28: Bias Probability vs. Total Execution Time - World 5

(a) Model World 5



(b) Real World 5

Figure 6.29: Best Result for World 5

For the results of the Humanoid Robot in World 5 shown in Table 6.8 if we look at the delta between the total execution time and total move test time, then the fastest solution returned by the motion planner for a specific combination of settings was given in 1.23 seconds. The best result in simulation is shown in Figure 6.29a and the associated execution on the real robot is shown in Figure 6.29b. Based on visual inspection of the motion plan, I somewhat agree that the motion planner selected the best solution. The motion plan determined is very reasonable although there might be a shorter path distance wise.

In any cases, there is no doubt that the Sphere Robot would be able to successfully execute the motion plan found by the motion planner. For the Humanoid Robot there is also no doubt that it would be able to execute the motion plan found by the motion planner.

Based on the results, the motion planner had the greatest difficulty with the Sphere Robot in World 3. World 3 is the world with narrow passages. One problem with the motion planner in this case is that the initial configuration and goal configuration are in terms of Euclidean distance extremely close to one another. Although they are close, there is an impassible obstacle between them. Whenever an attempt is made to connect a configuration to the goal configuration, selecting the nearest configuration is actually not the right thing to do since there is obviously no path. The motion planner does not find a solution until a random configuration is generated that is closer to the goal configuration then the initial configuration which takes quite a long time. A potential solution to this problem is discussed in Section 6.6.

6.6 Modifications and Optimizations

There are a number of possible optimizations and improvements that could be performed in my implementation as the basis for experimentation beyond that presented here. One possibility is to always pick the goal on the first iteration of the RRT construction algorithm. This handles the simple cases where there is a direct path to the goal configuration from the initial configuration. My experiments did not contain any trivial cases, therefore this optimization was not implemented. If trivial cases are expected, based on the ease of implementation and performance gain for the trivial case, there is no reason not to implement this optimization.

Another simple improvement that could be implemented is, regardless of bias probability, choose the goal configuration if the last configuration added to the RRT is within distance ϵ from the goal configuration. This could potentially be a way to prevent unnecessary additional exploration when a configuration in the RRT is already close to the goal configuration with a direct path. Based on the ease of implementation and performance gain, this optimization should be implemented in the future.

One problem with my motion planner identified in experiments with the Sphere Robot in World 3 is when the initial configuration and goal configuration are close to one another but there is no direct path between the two configurations. A potential solution to this problem could be to try to connect every configuration added to the RRT to the goal configuration. Unfortunately, since the current move test

implementation uses the incremental simulator stepped in real-time, move tests are slow. This optimization would result in a significant increase in total execution time. If the move test implementation is improved in the future, then this optimization could be implemented.

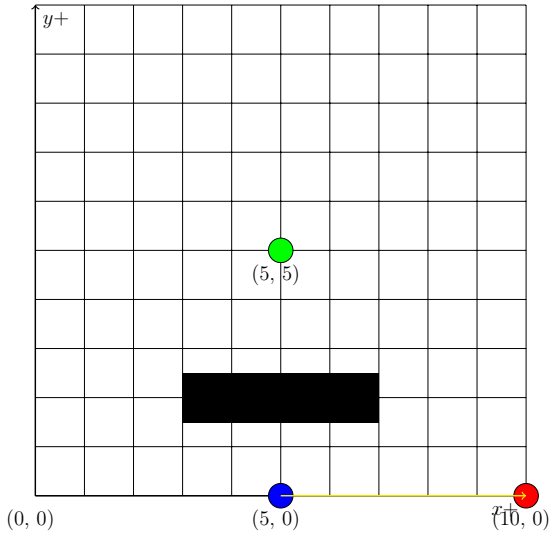
A different solution to this problem is to use the k-NNs instead of just the NN. This was also suggested by Urmson and Simmons [2003] for a different reason as described in Section 4.5.14. To understand why using k-NNs would be advantageous, consider the following simple 2D example shown in Figure 6.30. The initial state of the RRT is shown in Figure 6.30a, the path between the initial configuration at (5, 0) and goal configuration at (5, 5) is blocked by an obstacle. A random configuration at (10, 0) is connected to the initial configuration and there exists a direct path from the random configuration to the goal configuration. If the next random configuration chosen is the goal configuration, the nearest neighbor is the initial configuration with a distance of 5 units. The RRT construction algorithm given in Algorithm 5 will attempt to move from the nearest configuration to the goal configuration, thus generating the configuration shown at (5, 1.25) in Figure 6.30b since the obstacle is blocking the path. Using only the NN now requires additional random configurations to be generated even though there is clearly a path between a configuration in the RRT to the goal configuration. If k-NNs are used, the connection between the configuration with a direct path to the goal configuration is made as shown in Figure 6.30c.

The RRT construction algorithm given in Algorithm 5 can be modified by having the NearestNeighbor(rand_c, RRT) method return a list of the k-NNs, and the Move method tests the list of k-NNs until the random configuration is reached or all k neighbors are tested. The neighbor whose motion minimizes the distance to the random configuration is returned. Pseudocode for the algorithm is given in Algorithm 17.

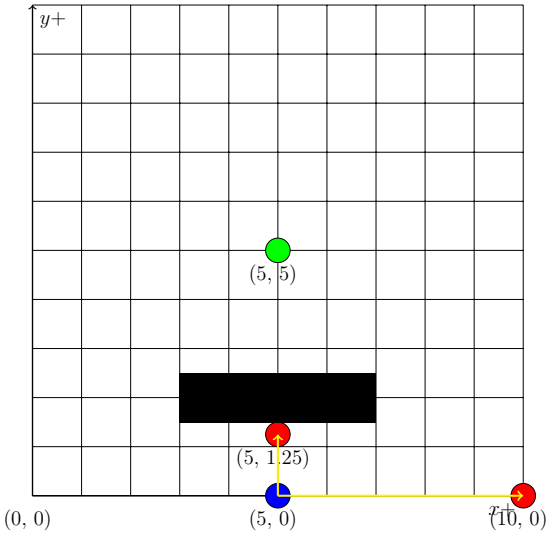
Although there can be an increase in performance when using k-NNs there can also a decrease in performance. If move tests are an expensive operation and all of the k-NNs are blocked as shown in Figure 6.31c, the additional move test time is three times (when k=3) longer even though the result is the same as shown in Figure 6.31b and 6.31d.

Algorithm 17 k-Nearest Move

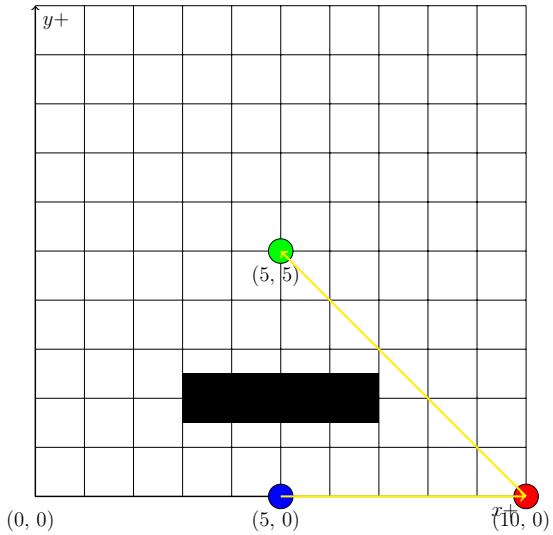
```
1: procedure MOVE(rand_c, kNearest, k)
2:     nearest_distance ← MAX_VALUE
3:     motion ← NULL
4:     for i ← 0, k do                          ▷ Test each nearest configuration
5:         near_c ← kNearest[i]
6:         distance ← RobotMoveTest(rand_c, near_c)          ▷ Check if random
   configuration can be reached from nearest configuration
7:         if distance < nearest_distance then
8:             nearest_distance ← distance
9:             motion ← RobotMove(rand_c, near_c)
10:            if distance == 0 then
11:                return motion
12:            end if
13:        end if
14:    end for
15:    return motion
16: end procedure
```



(a) Initial State

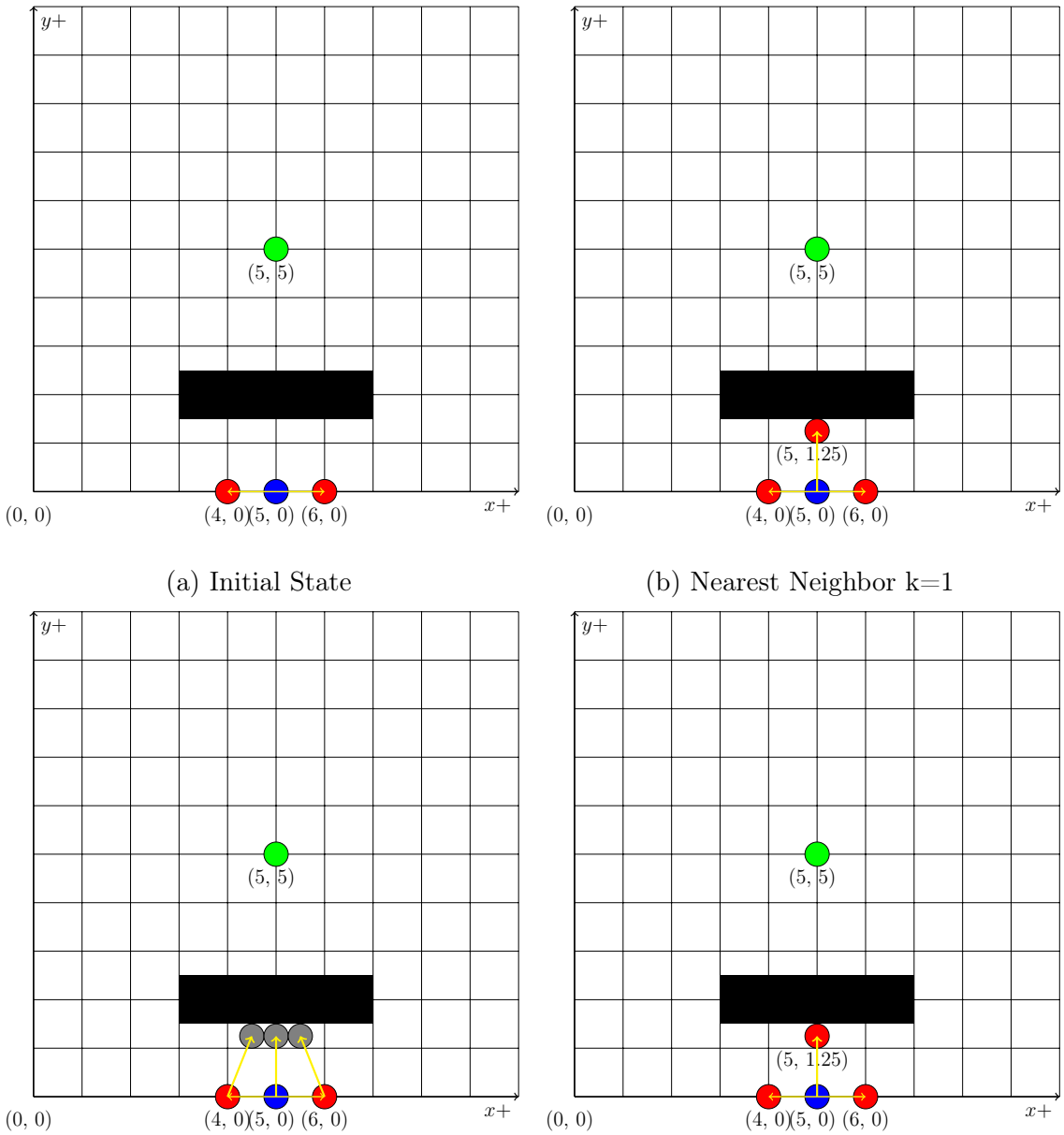


(b) Nearest Neighbor k=1



(c) Nearest Neighbor k=2

Figure 6.30: 2D RRT k-Nearest Example



(a) Initial State

(b) Nearest Neighbor k=1

(c) Nearest Neighbor k=3, Move Test

(d) Nearest Neighbor k=3

Figure 6.31: 2D RRT k-Nearest Example

In my motion planner no stopping conditions were implemented. Since my solution uses a probabilistic approach, there is the possibility that the motion planner can execute indefinitely when no solution exists or if it becomes trapped in local minima. For example, a case that clearly has no solution is where the initial configuration is enclosed in box with the goal configuration outside of the box. Currently my motion planner will never halt in such a scenario. This could easily be fixed by implementing one of the following stopping conditions: maximum number of configurations reached, maximum total execution time reached, no configuration within distance n from the goal configuration after specific time t, or if the random configuration generator starts to generate many duplicate configurations.

6.7 Summary

In this chapter, experiments with my motion planner were conducted with different robots and worlds. The performance of my motion planner was evaluated and suggestions were made for improvements that could be made to the motion planner. The results were overall promising, but there is much room for improvement.

Chapter 7

Conclusion

The focus of my research was on solving the motion planning problem for robots. The motion planning problem for robots is an extremely difficult problem. It is a form of the general mover's problem which is PSPACE-hard [Reif, 1979]. In order to provide a workable solution to the motion planning problem for robots, a wide array of knowledge in robotics was required. I became proficient in a wide array of topics in robotics as demonstrated in Chapter 2. In addition to this knowledge, various problems had to be addressed in parallel with the motion planning problem for robots, such as: FK, IK, differential constraints, dynamics, trajectory planning, motor control, NN, and collision detection to name a few. Each of these problems individually are challenging.

I made many contributions to the motion planning problem for robots. In my thesis, state of the art robot motion planning techniques were surveyed. A solu-

tion to the general movers problem in the context of motion planning for robots was presented. My robot motion planner solved the general movers problem using a single-query sample-based tree planner combined with an incremental simulator. I provided an extensive analysis of the different elements that constitute a single-query sample-based tree algorithm; namely the Rapidly Exploring Random Tree (RRT). My motion planner was also tested in both simulation and the real world, which demonstrates that the simulated robot and world models can be translated and applied to the real world. Experiments were conducted and the results analyzed. Based on the results, methods for tuning my motion planner to improve the performance were proposed.

I developed a framework for rapidly prototyping robot and world models which was referred to as the Motion Simulator in my thesis. The robot and world models can be used as inputs to the motion planner. The infinite possibilities of different models as inputs allows for evaluation of the motion planner under many different conditions. This framework in itself is a significant contribution because it could be used by others in the future. Not only could the framework be used to create different robot and world models, but the motion planner module could be replaced with a completely different one to evaluate different types of motion planners.

To the best of my knowledge after surveying many RRT papers, the ability to visualize the internal RRT structure is unique. The ability to visualize the internal RRT structure provides insight into the performance of the motion planner. The

RRT structure can be visualized as a tree or a Voronoi diagram. The benefits of being able to see the internal RRT structure were apparent in Chapter 6. The visual representation of the RRT structure clearly illustrated the difference between the EXTEND and CONNECT algorithms. It was easy to see that the EXTEND algorithm generates longer branches than the CONNECT algorithm since motion towards random configurations is only performed for n time steps. For configurations that are far apart, EXTEND must generate many intermediate configurations between the two configurations where as if it is possible to move between the two configurations CONNECT does not require any intermediate configurations.

In conclusion, my thesis presented a novel approach to robot motion planning that can be applied to different types of robots and environments. However, my approach is not a silver bullet to a difficult problem. Although the results were promising, there is still much room for improvement in the future.

7.1 Future Work

In Section 6.6, based on the results from the experiments, many modification and optimizations were suggested. In the future, the affects of the suggested modifications and optimizations should be proven by additional experimentation.

The motion planner is currently implemented on a desktop computer. For the real robot, the motion plan is calculated on the desktop computer then sent to the robot via serial communication in order to determine if the motion plan generated in

simulation is valid. The desktop computer should be eliminated in the future, and the motion planner should be implemented in a computer on-board the real robot. The visualization of the motion plan that the Motion Simulator generates in real-time as the motion plan is created is also not necessary on the real robot. Omitting the visualization portions will allow for a computer with fewer resources to be used and also decrease the total execution time of the motion planner.

As stated in Section 5.2.5, the world models for the real world use pre-measured values for the coordinates, orientation, and sizes of the obstacles and goal in the environment. The coordinates, orientations, and sizes of the obstacles and goal in the environment should come from a vision system such as the one described in Section 5.1.3 and a map generated by a localization and mapping algorithm such as the one described in Section 5.1.4. Errors in the vision system would provide an interesting challenge to a motion planner, and such a planner (or added technology) would have to adapt to such errors.

The total move test time accounts for the majority of the total execution time of the motion planner as found in Section 6.5. This must be significantly improved in the future. Currently, due to the implementation of the move test (IK and trajectory planning solutions), determining if the robot can move between two configurations is slow since the incremental simulator is stepped in real-time. This could easily be solved by stepping the incremental simulator faster than real-time, or if the move test could be done with the models alone by using parametric equations.

The sample bias method of the RRT was designed to be configurable. In this thesis, a goal bias was explored. Using the Motion Simulator framework to evaluate other sample bias methods as described in Section 4.1 would be easy to do with little effort.

The Motion Simulator allows for robot and world models to be created relatively easily. It would be beneficial to model other real robots and worlds, then evaluate the motion planner presented in this thesis with these robots and worlds.

Although the humanoid robot Blitz was completely modelled, whole body motion was not demonstrated in the experiments. In the future whole body motion shall be evaluated with the motion planner presented in this thesis.

Appendix A

Acronyms

ADC Analog-to-Digital Converter 115

AMotor Angular Motor 127

ANN Approximate Nearest Neighbor vi

API Application Programming Interface 130

BSP Binary Space Partitioning iv

CAD Computer-aided design 135

CFREE Configuration Free 14

CL-RRT Closed-Loop RRT v

COBS Configuration Obstacle 14

COG Center of Gravity 115

COP Center of Pressure iv

CPU Central Processing Unit 119

CSPACE Configuration Space iii

CT Collision Tendency 88

D-H Denavit Hartenberg iv

DARPA Defense Advanced Research Projects Agency 92

DOF Degrees of Freedom 2

DTE Data Terminal Equipment 104

DUC DARPA Urban Challenge 92

EET Exploring/Exploiting Tree v

FBD Functional Block Diagram viii

FIRA Federation of International Robot-soccer Association 97

FK Forward Kinematics iv

GPU Graphics Processing Unit

hRRT Heuristically-guided RRT v

IK Inverse Kinematics iv

IrDA Infrared Data Association 104

JT Jacobian Transpose 87

JT-RRT Jacobian Transpose RRT v

k-d k-dimensional 49

LED Light Emitting Diode 105

MP-RRT Multipartite RRT v

NN Nearest Neighbor iv

NOOP No Operation 120

NP Non-deterministic Polynomial-time 4

OB-RRT Obstacle-Based RRT v

ODE Open Dynamics Engine v

OpenGL Open Graphics Library vi

PID Proportional-Integral-Derivative iv

POS Polygon of Support iv

PRM Probabilistic Road Map iv

pRRT Particle RRT v

PSPACE Polynomial Space 3

PUMA Programmable Universal Machine for Assembly 86

PWM Pulse Width Modulation 142

RAM Random Access Memory

RNG Random Number Generator viii

RRT Rapidly Exploring Random Tree i

RTOS Real-Time Operating System 105

SCARA Selective Compliance Articulated Robot Arm 13

SLAM Simultaneous Localization and Mapping 110

TSPACE Task Space iii

TSPACE-RRT Task Space RRT v

UI User Interface 139

XML Extensible Markup Language 123

ZMP Zero Moment Point iv

Appendix B

Glossary

Glossary

Differential Constraints

Constraints that restrict the allowable velocities at each point. [LaValle, 2006]. 2

Dynamics

In physics, dynamics is concerned with the effect of external forces on the state of motion. [Burns, 2012]. 2

Non-Holonomic

Differential constraints that cannot be fully integrated to remove time derivatives of the state variables. [LaValle, 2006]. 36

Appendix C

Scilab Functions

```
function [x, y, z] = randSphere(radius)
r0 = rand()
l = 0 + r0 * ((2*%pi) - 0)
r1 = rand()
h = -(%pi/2) + r1 * ((%pi/2) - -(%pi/2))
x = rand() * radius * cos(l) * cos(h)
y = rand() * radius * sin(h)
z = rand() * radius * sin(l) * cos(h)
endfunction
x = []
y = []
z = []
```

```
for i = 1:10000
[x(i), y(i), z(i)] = randSphere(1)
end
param3d1(x, y, z)
e = gce()
e.mark_mode = 'on';
e.line_mode = 'off';
```

Bibliography

NIST/SEMATECH e-handbook of statistical methods. http://www.itl.nist.gov/div898/handbook/, November 2007.

Learning modules - medical gross anatomy anatomical orientation. http://anatomy.med.umich.edu/modules/anatomical_orientation_module/Module-AnatOrient.pdf, March 2008.

M. A. Ali, H. A. Park, and C. G. Lee. Closed-form inverse kinematic joint solution for humanoid robots. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 704–709. IEEE, 2010.

M. Ang and V. Tourassis. Singularities of euler and roll-pitch-yaw representations. *Aerospace and Electronic Systems, IEEE Transactions on*, AES-23(3):317–324, May 1987. ISSN 0018-9251. doi: 10.1109/TAES.1987.310828.

M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *Signal Processing, IEEE Transactions on*, 50(2):174–188, 2002.

ATMEL. *8-bit AVR Microcontroller with 128K Bytes In-System Programmable Flash (ATmega128, ATmega128L)*, October 2006.

F. Aurenhammer. Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, Sept. 1991. ISSN 0360-0300. doi: 10.1145/116873.116880. URL `http://doi.acm.org/10.1145/116873.116880`.

E. Ayyappa. Normal human locomotion, part 1: Basic concepts and terminology. *Journal of Prosthetics and Orthotics*, 9(1):10–17, 1997.

A. Bagheri, F. Najafi, R. Farrokhi, R. Moghaddam, and M. Felezi. Design, dynamic modification, and adaptive control of a new biped walking robot. *International Journal of Humanoid Robotics*, 3(1):105–126, 2006.

J. Bagot, J. Anderson, and J. Baltes. Vision-based multi-agent slam for humanoid robots. In *Proceedings of the 5th International Conference on Computational Intelligence, Robotics and Autonomous Systems (CIRAS-2008)*, pages 171–176, June 2008.

M. M. L. J. S. A. M. M. Bajd, T. *Robotics*, volume VIII. 2010.

J. Baltes and J. Anderson. Flexible binary space partitioning for robotic rescue. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3144–3149, Las Vegas, October 2003.

J. Baltes, C. Iverach-Brereton, C. T. Cheng, and J. Anderson. Threaded c and freezeros. In *Proceedings of FIRA 2011, CCIS 212*, pages 170–177, Kaohsiung, Taiwan, August 2011.

J. Basch, J. Erickson, L. J. Guibas, J. Hershberger, and L. Zhang. Kinetic collision detection between two simple polygons. *Computational Geometry*, 27(3):211–235, 2004.

D. Berenson and S. Srinivasa. Probabilistically complete planning with end-effector pose constraints. In *IEEE International Conference on Robotics and Automation (ICRA '10)*, May 2010.

J. Bloomenthal and J. Rokne. Homogeneous coordinates. *Vis. Comput.*, 11(1):15–26, Jan. 1994. ISSN 0178-2789. doi: 10.1007/BF01900696. URL `http://dx.doi.org/10.1007/BF01900696`.

Boost. Boost c++ libraries, 2014. URL `http://www.boost.org/doc/`.

G. Brassard and P. Bratley. *Fundamentals of Algorithmics*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996. ISBN 0-13-335068-1.

R. Brooks. A robust layered control system for a mobile robot. *Robotics and Automation, IEEE Journal of*, 2(1):14–23, Mar 1986. ISSN 0882-4967. doi: 10.1109/JRA.1986.1087032.

L. M. Burns. *Modern Physics for Science and Engineering*. Physics Curriculum and Instruction, 2012.

S. R. Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. Technical report, IEEE Journal of Robotics and Automation, 2004.

S.-H. Cha. Comprehensive survey on distance/similarity measures between probability density functions. *International Journal of Mathematical Models and Methods in Applied Sciences*, 1(4):300–307, 2007. URL `http://www.gly.fsu.edu/~parker/geostats/Cha.pdf`.

F. S. Cheng. Advanced techniques of industrial robot programming, advances in robot manipulators. Technical report, 2010.

M. Corral. *Vector Calculus*. Schoolcraft College, 2008.

J. Denavit and R. S. Hartenberg. A kinematic notation for lower-pair mechanisms based on matrices. *Trans. ASME E, Journal of Applied Mechanics*, 22:215–221, June 1955.

T. K. Dey and W. Zhao. Approximating the medial axis from the voronoi diagram with a convergence guarantee. *Algorithmica*, 38(1):179–200, Oct. 2003. ISSN 0178-4617. doi: 10.1007/s00453-003-1049-y. URL `http://dx.doi.org/10.1007/s00453-003-1049-y`.

M. M. Deza and E. Deza. *Encyclopedia of Distances*. Springer Berlin Heidelberg, 2009. doi: 10.1007/978-3-642-00234-2_1.

B. Donald, P. Xavier, J. Canny, and J. Reif. Kinodynamic motion planning, 1993.

D. Eberly. Euler angle formulas. Technical report, 2014.

K. Fujiwara, F. Kanehiro, H. Saito, S. Kajita, K. Harada, and H. Hirukawa. Falling motion control of a humanoid robot trained by virtual supplementary tests. In *Institute of Electrical and Electronics Engineers (IEEE) Conference on Robotics and Automation*, pages 1077–1082, New Orleans, LA, United States, 2004.

E. Gansner, E. Koutsofios, and S. North. *Drawing graphs with dot*, January 2006.

S. Ge and Y. Cui. New potential functions for mobile robot path planning. *Robotics and Automation, IEEE Transactions on*, 16(5):615–620, Oct 2000. ISSN 1042-296X. doi: 10.1109/70.880813.

R. Geraerts and M. H. Overmars. A comparative study of probabilistic roadmap planners. In *IN: WORKSHOP ON THE ALGORITHMIC FOUNDATIONS OF ROBOTICS*, pages 43–57, 2002.

Graphviz. Graphviz - graph visualization software, 2014. URL `http://www.graphviz.org/Documentation.php`.

F. S. Grassia. Practical parameterization of rotations using the exponential map.

J. Graph. Tools, 3(3):29–48, Mar. 1998. ISSN 1086-7651. doi: 10.1080/10867651.1998.10487493. URL http://dx.doi.org/10.1080/10867651.1998.10487493.

M. P. Groover, M. Weiss, and R. N. Nagel. *Industrial Robotics: Technology, Programming and Application*. McGraw-Hill Higher Education, 1st edition, 1986. ISBN 007024989X.

S. Ha, Y. Han, and H. Hahn. Adaptive gait pattern generation of biped robot based on human's gait pattern analysis. In *Proceedings of World Academy of Science, Engineering and Technology*, pages 406–411, 2007.

D. Hsu, J.-C. Latombe, and H. Kurniawati. On the probabilistic foundations of probabilistic roadmap planning. In S. Thrun, R. Brooks, and H. Durrant-Whyte, editors, *Robotics Research*, volume 28 of *Springer Tracts in Advanced Robotics*, pages 83–97. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-48110-2. doi: 10.1007/978-3-540-48113-3_8. URL http://dx.doi.org/10.1007/978-3-540-48113-3_8.

Y. K. Hwang and N. Ahuja. A potential field approach to path planning. *IEEE T. Robotics and Automation*, 8(1):23–32, 1992. URL http://dblp.uni-trier.de/db/journals/trob/trob8.html#HwangA92.

L. Jaillet and T. Simeon. A prm-based motion planner for dynamically changing environments. In *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings.*

2004 IEEE/RSJ International Conference on, volume 2, pages 1606–1611 vol.2, Sept 2004. doi: 10.1109/IROS.2004.1389625.

M. Kalisiak and M. van de Panne. Rrt-blossom: Rrt with a local flood-fill behavior. In *ICRA*, pages 1237–1242. IEEE, 2006. URL `http://dblp.uni-trier.de/db/conf/icra/icra2006.html#KalisiakP06`.

S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. J. Rob. Res.*, 30(7):846–894, June 2011. ISSN 0278-3649. doi: 10.1177/0278364911406761. URL `http://dx.doi.org/10.1177/0278364911406761`.

S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller. Anytime motion planning using the rrt*. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1478–1483. IEEE, 2011.

L. Kavan and J. Žára. Fast collision detection for skeletally deformable models. In *Computer Graphics Forum*, volume 24, pages 363–372. Wiley Online Library, 2005.

L. E. Kavraki, J. claude Latombe, R. Motwani, and P. Raghavan. Randomized query processing in robot path planning (extended abstract). In *Journal of Computer and System Sciences*, pages 353–362, 1995.

L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. on Robotics and Automation*, 12(4):566–580, 1996.

O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *Int. J. Rob. Res.*, 5(1):90–98, Apr. 1986. ISSN 0278-3649. doi: 10.1177/027836498600500106. URL http://dx.doi.org/10.1177/027836498600500106.

T. Koziara and N. Bicanic. Bounding box collision detection. In *ACME conference: University of Sheffield*, 2007.

J. J. Kuffner, Jr., S. Kagami, K. Nishiwaki, M. Inaba, and H. Inoue. Dynamically-stable motion planning for humanoid robots. *Auton. Robots*, 12(1):105–118, Jan. 2002. ISSN 0929-5593. doi: 10.1023/A:1013219111657. URL http://dx.doi.org/10.1023/A:1013219111657.

J. J. Kuffner Jr. and S. M. Lavalle. Rrt-connect: An efficient approach to single-query path planning. In *Proc. IEEE Intl Conf. on Robotics and Automation*, pages 995–1001, 2000.

S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

S. M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, 1998.

S. M. Lavalle and J. J. Kuffner Jr. Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions*, pages 293–308, 2000.

J. Ledin. *Embedded control systems in C/C++ an introduction for software developers using MATLAB*. CMP Books, San Francisco, California.

A. Liegeois. Automatic supervisory control of the configuration and behavior of multibody mechanisms. *IEEE Trans. Systems, Man, and Cybernetics*, 7(12):842–868, 1977.

S. R. Lindemann and S. M. LaValle. Incrementally reducing dispersion by increasing voronoi bias in rrts. In *In Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, volume 4, pages 3251–3257, 2004.

T. Lozano-Perez. Spatial planning: A configuration space approach, 1980.

O. Luders, S. Karaman, E. Frazzoli, and J. How. Bounds on tracking error using closed-loop rapidly-exploring random trees. In *in American Control Conference (ACC*, pages 5406–5412, 2010.

E. Mazer, J. M. Ahuactzin, E.-G. Talbi, and P. Bessiere. The ariadne's clew algorithm, 1996.

K. Mehlhorn and P. Sanders. Algorithms and data structures. 2007.

N. Melchior and R. Simmons. Particle rrt for path planning with uncertainty. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 1617–1624, April 2007. doi: 10.1109/ROBOT.2007.363555.

M. Meredith and S. Maddock. Real-time inverse kinematics: The return of the jacobian. 2004.

D. M. Mount. *ANN Programming Manual*, 2010.

M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *In VISAPP International Conference on Computer Vision Theory and Applications*, pages 331–340, 2009.

R. M. Murray, S. S. Sastry, and L. Zexiang. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1994. ISBN 0849379814.

S. A. Nene and S. K. Nayar. A simple algorithm for nearest neighbor search in high dimensions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19 (9):989–1003, 1997.

OpenGL. Opengl API documentation overview, 2014. URL `http://www.opengl.org/documentation/`.

J. E. Pratt. *Exploiting Inherent Robustness and Natural Dynamics in the Control of Bipedal Walking Robots*. PhD thesis, Massachusetts Institute of Technology (MIT), June 2000a.

J. E. Pratt. *Exploiting Inherent Robustness and Natural Dynamics in the Control*

of Bipedal Walking Robots. PhD thesis, Massachusetts Institute of Technology (MIT), June 2000b.

J. H. Reif. Complexity of the mover's problem and generalizations. In *SFCS '79: Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, pages 421–427, Washington, DC, USA, 1979. IEEE Computer Society. doi: http://dx.doi.org/10.1109/SFCS.1979.10.

M. Rickert, O. Brock, and A. Knoll. Balancing exploration and exploitation in motion planning. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 2812–2817, May 2008. doi: 10.1109/ROBOT.2008.4543636.

J. Ritter. A fast approximation to 3d euclidean distance. In A. S. Glassner, editor, *Graphics Gems*, pages 432–433. Academic Press, 1990.

Robotis. *Bioloid - User's Guide*, 1.00 edition, a.

Robotis. *Bioloid Quick Start - Comprehensive Kit Robot Series*, 1.00 edition, b.

Robotis. *Dynamixel AX-12*, June 2006.

S. Rodriguez, X. Tang, J. ming Lien, and N. M. Amato. An obstacle-based rapidly-exploring random tree. In *in Proc. IEEE International Conference on Robotics and Automation (ICRA*, 2006.

P. J. Schneider and D. Eberly. *Geometric Tools for Computer Graphics*. Elsevier Science Inc., New York, NY, USA, 2002. ISBN 1558605940.

J. Selig. *Introductory Robotics*. Prentice hall, 1992.

E. Shkolnik and R. Tedrake. Path planning in 1000+ dimensions using a task-space voronoi bias. In *In IEEE International Conference on Robotics and Automation*, 2009.

K. Shoemake. Animating rotation with quaternion curves. *SIGGRAPH Comput. Graph.*, 19(3):245–254, July 1985. ISSN 0097-8930. doi: 10.1145/325165.325242. URL `http://doi.acm.org/10.1145/325165.325242`.

R. Smith. *Open Dynamics Engine v0.5 User Guide*, February 2006.

M. W. Spong, S. Hutchinson, and M. Vidyasagar. *Robot Modeling and Control*, chapter 3, pages 73–110. Wiley, 2005.

I. A. Sucan and L. E. Kavraki. A sampling-based tree planner for systems with complex dynamics. *IEEE Transactions on Robotics*, 28(1):116–131, 2012. ISSN 1552-3098. doi: 10.1109/TRO.2011.2160466. URL `http://dx.doi.org/10.1109/TRO.2011.2160466`.

I. A. Sucan, J. F. Kruse, M. Yim, and L. E. Kavraki. Kinodynamic motion planning with hardware demonstrations. In *IROS*, pages 1661–1666, 2008.

Z. Tang, C. Zhou, and Z. Sun. Trajectory planning for smooth transition of a biped robot. In *Institute of Electrical and Electronics Engineers (IEEE) International*

Conference on Robotics and Automation (ICRA) 2003 Conference Proceedings, pages 2455–2460, Piscataway, NJ, United States, 2003.

K. I. Tsianos, I. A. Sucan, and L. E. Kavraki. Sampling-based robot motion planning: Towards realistic applications. *Computer Science Review*, 1:2–11, August 2007. doi: 10.1016/j.cosrev.2007.08.002.

C. Urmson and R. Simmons. Approaches for heuristically biasing rrt growth. In *IEEE/RSJ IROS 2003*, October 2003.

J. M. Vandeweghe, D. Ferguson , and S. Srinivasa. Randomized path planning for redundant manipulators without inverse kinematics. In *IEEE-RAS International Conference on Humanoid Robots*, November 2007.

J. Vaščák. Navigation of mobile robots using potential fields and computational intelligence means. *Acta Polytechnica Hungarica*, 4(1):63–74, 2007. ISSN 1785-8860.

Voro. Voro++, 2014. URL `http://math.lbl.gov/voro++/doc/`.

M. Vukobratovic, B. Borovac, and V. Potkonjak. Towards a unified understanding of basic notions and terms in humanoid robotics. *Robotica 2007*, 25:87–101, July 2006.

R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In A. Gupta, O. Shmueli,

and J. Widom, editors, *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 194–205. Morgan Kaufmann, 1998. ISBN 1-55860-566-5.

R. Wright, B. Lipchak, and N. Haemel. *Opengl® Superbible: Comprehensive Tutorial and Reference, Fourth Edition*. Addison-Wesley Professional, fourth edition, 2007. ISBN 9780321498823.

J. Xu, V. Duindam, R. Alterovitz, and K. Goldberg. Motion planning for steerable needles in 3d environments with obstacles using rapidly-exploring random trees and backchaining. In *CASE*, pages 41–46. IEEE, 2008. ISBN 978-1-4244-2022-3. URL `http://dblp.uni-trier.de/db/conf/case/case2008.html#XuDAG08`.

A. Yershova and S. M. LaValle. Improving motion-planning algorithms by efficient nearest-neighbor searching. *IEEE Transactions on Robotics*, 23(1):151–157, 2007.

K. Zhou, Q. Hou, R. Wang, and B. Guo. Real-time kd-tree construction on graphics hardware. In *ACM SIGGRAPH Asia 2008 Papers*, SIGGRAPH Asia '08, pages 126:1–126:11, New York, NY, USA, 2008. ACM. ISBN 978-1-4503-1831-0. doi: 10.1145/1457515.1409079. URL `http://doi.acm.org/10.1145/1457515.1409079`.

M. Zucker, J. Kuffner, and M. Branicky. Multipartite rrts for rapid replanning in dynamic environments. *Robotics and Automation, 2007 IEEE International Conference on*, pages 1603–1609, April 2007. ISSN 1050-4729. doi: 10.1109/ROBOT.2007.363553.