

# Using Physics Simulators to Aid in Real-Time Robot Planning

*Michael Tong*



Electrical Engineering and Computer Sciences  
University of California at Berkeley

Technical Report No. UCB/EECS-2018-68

<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-68.html>

May 15, 2018

Copyright © 2018, by the author(s).  
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

### Acknowledgement

Would like to firstly thank Professor Ronald Fearing for his advice and guidance throughout this project. Would also like to thank Justin Yim for his help on understanding the precise mechanisms of the ODE physics engine, Anusha Nagabandi for help understanding ROS, Carlos Casarez for help in understanding the VelociRoACH robots, and everyone else in the Biomimetics lab group for their help and support.

---

# Using Physics Simulations to Aid in Real-Time Robot Planning

by Michael Tong

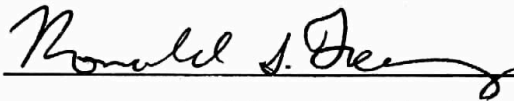
---

## Research Project

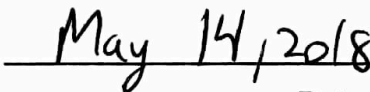
Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

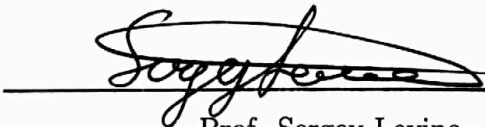


Prof. Ronald S. Fearing  
Research Advisor



Date

\*\*\*\*\*



Prof. Sergey Levine  
Second Reader



Date

## **Abstract**

In this report we focus on building a framework for a feedback loop for robot planning that incorporates physics simulations in modeling complex interactions between the robots and their corresponding environment. By using a physics simulator to handle the dynamics, one can consider the tasks of building an environment and the dynamics of the robot separately, making it much easier to design and implement tests. Additionally, using a physics simulation allows for the testing of hypothetical environments and scenarios that would be prohibitively expensive to physically build and test. This report will show how to conceptually build such a framework, describe in detail the realization of such a framework, and discuss the results of experiments performed using the framework.



# Contents

<b>Contents</b>	<b>i</b>
<b>List of Figures</b>	<b>iii</b>
<b>List of Tables</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Goals and Motivation . . . . .	1
1.2 Problem . . . . .	2
1.3 Related Work . . . . .	2
1.4 Introduction . . . . .	3
<b>2 Physical World Systems</b>	<b>5</b>
2.1 Kinect . . . . .	5
2.2 Robots . . . . .	6
2.3 Optitrack . . . . .	6
<b>3 ROS systems</b>	<b>7</b>
3.1 PCL: point cloud processing . . . . .	8
3.2 PCL: mesh generation . . . . .	10
3.3 Environment Generator . . . . .	11
3.4 Performance Evaluator . . . . .	11
3.5 Driver . . . . .	14
<b>4 VREP systems</b>	<b>18</b>
4.1 Introduction to VREP . . . . .	18
4.2 Simulation Scene . . . . .	19
4.3 Robot Used . . . . .	20
4.4 OMPL . . . . .	21
<b>5 Software Based Experiments</b>	<b>24</b>
5.1 OpenSCAD . . . . .	24

<b>6</b>	<b>Experiments</b>	<b>27</b>
6.1	Picking the lowest energy path . . . . .	28
6.2	Picking rough terrain . . . . .	28
6.3	Running the robot on an environment generated from the physical world . .	30
<b>7</b>	<b>Results</b>	<b>34</b>
7.1	Picking the lowest energy path . . . . .	35
7.2	Picking rough terrain . . . . .	35
7.3	Running the robot on an environment generated from the physical world . .	38
<b>8</b>	<b>Conclusion</b>	<b>41</b>
8.1	Limitations to the system . . . . .	41
8.2	Future Work . . . . .	42
8.3	Overall Conclusion . . . . .	42
<b>A</b>	<b>Modeling the turning radius of the robot</b>	<b>44</b>
<b>B</b>	<b>Sample *.yaml configuration file.</b>	<b>46</b>
<b>C</b>	<b>OpenSCAD shapes</b>	<b>49</b>
C.1	Double Bridge . . . . .	49
C.2	Introduction of random bumps . . . . .	51
	<b>Bibliography</b>	<b>53</b>

# List of Figures

1.1	Block diagram of the system. Blue arrows have been implemented, red arrows are planned but not fully implemented. . . . .	4
3.1	Example of voxelization. (top) Original sampled points. (bottom) Pixels grouped into voxels with their respective centroids in black. . . . .	9
3.2	Setup of experiment. . . . .	10
3.3	Mesh generation example. . . . .	12
3.4	Plot of simulated VelociRoACH turning behavior at various motor ratios. . . . .	17
4.1	Picture of the simulated VelociRoACH. . . . .	20
4.2	Example of projection of the robot onto a 2 dimensional surface. . . . .	22
5.1	Simplified block diagram. . . . .	25
5.2	The double bridge, an example of a simplified mesh generated with OpenSCAD. . . . .	26
6.1	The setup for the experiment described in Section 6.1. . . . .	29
6.2	The setup for the experiment described in Section 6.2. . . . .	30
6.3	Close up rendering of the bumps used for the experiment described in section 6.2. . . . .	31
6.4	Picture of the setup in 6.3. . . . .	32
6.5	Close up of the ramp in 6.3. . . . .	33
6.6	Screenshot from the physics simulation in 6.3. The robot is starting from the right and trying to reach the position marked by the dummy robot (blue) on the left. . . . .	33
7.1	Example path generated from section 6.2. In this case the robot succeeds in traversing the path. . . . .	36
7.2	Example path generated from section 6.2. In this case the robot fails in traversing the path. . . . .	37
7.3	Example path generated from section 6.3. In this case the robot succeeds traversing the path. . . . .	39
7.4	Example path generated from section 6.3. In this case the robot fails traversing the path. . . . .	40

A.1	Example run that was used to generate data for Fig 3.4. . . . .	45
B.1	Annotated screenshot of the scene described in B. . . . .	48
C.1	Conceptual drawing of the ramps of the double bridge. . . . .	50
C.2	Conceptual drawing of the bridges of the double bridge. . . . .	50
C.3	Drawing of the bumps used when constructing the double bridge with artificial height variation. . . . .	52

# List of Tables

3.1	Results of the Kinect experiment. . . . .	9
7.1	Results of the experiment described in section 6.1. . . . .	35
7.2	Results of the experiment described in section 6.2. . . . .	37
7.3	Results of the experiment described in section 6.3. . . . .	38
C.1	Dimensions of the bridge. . . . .	51

## Acknowledgments

Would like to firstly thank Professor Ronald Fearing for his advice and guidance throughout this project. Would also like to thank Justin Yim for his help on understanding the precise mechanisms of the ODE physics engine, Anusha Nagabandi for help understanding ROS, Carlos Casarez for help in understanding the VelociRoACH robots, and everyone else in the Biomimetics lab group for their help and support.

# Chapter 1

## Introduction

### 1.1 Goals and Motivation

The main goal of this report is to predict how a robot will perform in a given environment without having to run a physical experiment. This report's proposed solution to this problem is to use a combination of ROS (a common middleware used in robotics) and VREP (a physics simulator) to simulate what might happen. This allows for one to run more trials than one could using only physical tests because of the lower cost of failure in simulation.

Since VREP can model the interactions between the robots and their environment, we can treat the problems of reconstructing a simulated environment, building dynamics models for robots, and how the robots interact with their environment separately. This allows one to make quantitative predictions about how robots will perform in environments that would otherwise be too complicated to make such predictions. Additionally, this system allows for

planners that do not need to consider the robots' dynamics when planning. Since the physics simulator can simulate any series of actions a robot can perform it can validate and evaluate potential plans given by the planner.

## 1.2 Problem

The problem this report is trying to solve is a path-planning problem, where given some environment find a lowest energy cost path to guide a legged robot from a start position to a goal position. Since trying to model the dynamics of robots is difficult on arbitrary environments, an alternative is to model the environment and robot dynamics independently and to use a physics simulation to model their interactions.

## 1.3 Related Work

The approach this report takes to solving this problem can be seen as a version of software in the loop. Software in the loop is a test methodology where executable code such as algorithms (or even an entire controller strategy), usually written for a particular mechatronic system, is tested within a modelling environment that can help prove or test the software. Currently, there has been work on writing software in the loop for specific systems such as modeling automotive powertrain systems [19] [10] or robot manipulators [1]. There has also been work on building general framework [5].



For the modeling of the robotics models itself, there are several models to learn robot control [12] that can be used in order to plan how legged robots can move along a specified path [3]. The goal of this report is to combine these two ideas to allow for the testing of planning algorithms that work in a simulated environment and on a physical robot.

## 1.4 Introduction

As seen in Fig. 1.1, the system is divided into 3 main parts: the physical world, ROS, and the physics simulator VREP [15]. In the physical world, we rely on sensors such as the Kinect and Optitrack to get an idea of what the environment looks like. In VREP, we build and run physics simulations to get an idea of what might happen to a robot locomoting and to plan optimal paths. Lastly, we use ROS to handle most of the data processing and coordination.

In chapters 2, 3, and 4 the report will go into more detail about each of the blocks in the block diagram as shown in Fig. 1.1. A discussing about a simplified block diagram that only relies on software components is in chapter 5. Chapters 6 and 7 will go into the experiments and results. Finally, the conclusion is in chapter 8.

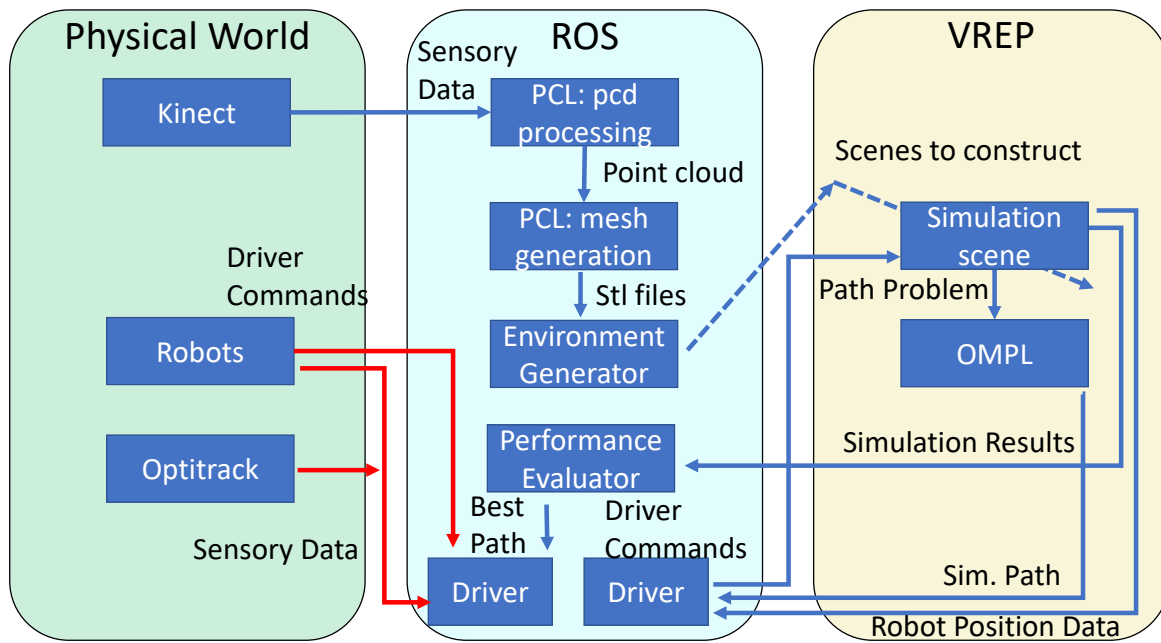


Figure 1.1: Block diagram of the system. Blue arrows have been implemented, red arrows are planned but not fully implemented.

## Chapter 2

# Physical World Systems

The components in this section all exist in the physical world, and their primary goals are to gather data about the real world and to run physical tests in it. The goal of the Kinect is to provide an approximation of the environment in the real world that will be used for physical simulations by the rest of the system. The robots are used to perform the physical tests and provide validation and feedback on the simulated models and results. Finally, the Optitrack system is a motion capture system used to provide information to be used when controlling the robots.

## 2.1 Kinect

In order to get a sample of the environment, we use an RGB-D camera. For this system, we are using the Kinect 2 camera, though other options such as the Intel RealSense cam-

era. When using the Kinect, it can record in 3 definitions: standard ( $424 \times 512$ ), QHD ( $540 \times 960$ ), and HD ( $1080 \times 1920$ ). Using higher resolution results in more computationally expensive processing and a lower framerate from the Kinect. For this system, I have found that processing the environment in QHD seems to be a good compromise between the computation time and fidelity of the image.

## 2.2 Robots

In order to test and develop the system I needed physical robots. The physical robots I decided to use are the VelociRoACH robots [7]. I chose this robot because it is difficult in general to predict the behavior of these legged robots. In contrast to wheeled robots, such as the Zumo robots [20], physics of the legs colliding with the ground are very difficult to model accurately, especially when the terrain is not flat.

## 2.3 Optitrack

The Optitrack is a motion capture system that can track the position and orientation of objects in 3D space in real time. While not fully implemented, the plan was to use the Optitrack system to help in controlling the physical robots by providing their locations and orientations. With the robot's position and orientation, it becomes possible to control the robot using ROS in real-time.

## Chapter 3

# ROS systems

The ROS systems form the backbone of the system, and serve as the center for processing that as well as interfacing with the physical and simulated systems. In order to internally process the environments, the raw point data from the Kinect is converted into a 3D mesh that can be used in physics simulations. These meshes are then used to generate the surrounding environment to use in simulation. To interface with the physical and simulated tests, there is a performance evaluator to evaluate how well simulated tests perform and a driver to control the physical and simulated robots. The next parts of this section will go into more detail about the individual pieces of the ROS system.

## 3.1 PCL: point cloud processing

In this block, the raw point data from the Kinect is downsampled and denoised in order to make it easier for the later steps to process the point cloud data. Currently the only processing done on the point cloud is the downsampling step. Currently the downsampling is done using a technique called voxelization. Voxelization is when one takes the points and group them into voxels of size  $dx \times dy \times dz$ . Then within each voxel one outputs the centroid of the points in the voxel as shown in in Fig. 3.1. In this manner one can take point clouds of arbitrary density and transform it to a point cloud of a specific density determined by the voxel size.

Voxelizing the points has two main advantages over using the raw point cloud data. The first is that the voxelized cloud returns a point cloud of roughly uniform density, which is useful since the Kinect does not uniformly sample the space in front of it. The second is that with less points it is more computationally efficient to compute the resulting meshes over the point cloud. Another benefit of voxelization is that with voxelization the point clouds can be processed in real-time as opposed to much slower methods such as Moving Least Squares.

### Characterizing the effectiveness of voxelization

To quantify how much voxelization reduces the error in measurements. I ran an experiment where I had the Kinect camera 6 meters away from the wall and had it sample a  $2\text{m} \times 2\text{m}$  square directly in front of it. A cartoon of the setup can be shown in Fig. 3.2. After



Figure 3.1: Example of voxelization. (top) Original sampled points. (bottom) Pixels grouped into voxels with their respective centroids in black.

voxelizing the raw camera data, I fit a linear plane to the wall and discarded outlier points that were over 5 times the sampled standard error. This process was repeated until all the points in the sampled point cloud are all part of the wall. The results of this experiment is shown below in Table 3.1. From the results, it seems that using larger voxels does not reduce the standard error by much. However, in practice the surfaces become much smoother than the standard errors would suggest as the points are spread out by their voxel size, so even if the errors were constant they would still be smoothed by a factor proportional to the voxel size.

voxel size (cm)	points	points kept	error (mm)
N/A	28359	28155	4.833
1	28420	28273	4.808
2	11209	11109	4.916
5	1908	1883	4.315
10	464	453	3.644
20	122	120	3.327

Table 3.1: Results of the Kinect experiment.

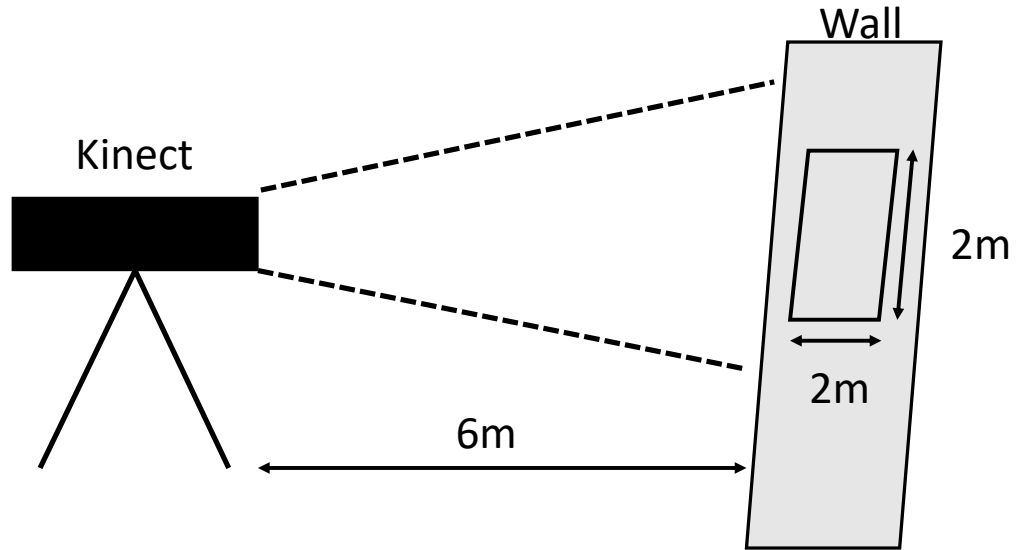


Figure 3.2: Setup of experiment.

## 3.2 PCL: mesh generation

The goal of mesh generation is to transform the sampled point cloud into a surface that is usable by a physics simulator such as VREP. For the mesh reconstruction we use PCL's [16] implementation of mesh reconstruction described in [11]. We use PCL to take in the point clouds and then output the reconstructed meshes as \*.stl files. One important thing to note is when reconstructing the meshes, some physics engines such as ODE [17] incorporate the direction of the facet normals when doing collision detection. So in order to have proper collision, one needs to align all of the surface normals such that they face in the outward direction. Since the Kinect gives the coordinates of its point clouds relative to the camera,



we can write that for every facet in the mesh at position  $x_i$  relative to the camera with the surface normal  $n_i$ , the inner product  $\langle x_i, n_i \rangle \leq 0$ . In other words, the normals all point in the direction towards the camera.

An example of the mesh generation can be seen in Fig. 3.3, where the Kinect is pointed at the environment in Fig. 3.3a and after the voxelization and mesh generation processing steps, the results can be seen in Fig. 3.3b.

### 3.3 Environment Generator

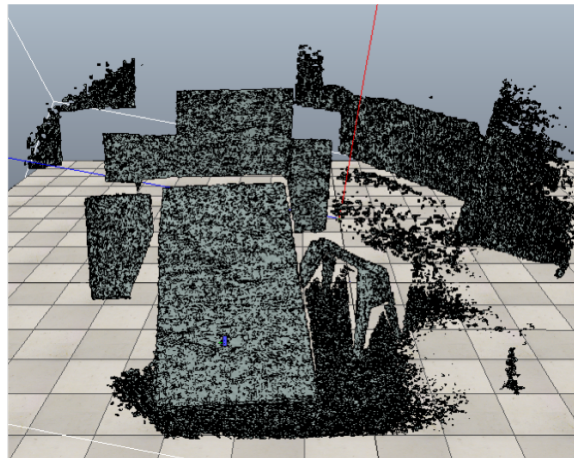
In order to generate the environments, I've been using `bml_vrep` [2]. `bml_vrep` takes in a configuration file (\*.yaml), of which an example can be seen at in the appendix section B. With this configuration file, all of the meshes and models are placed in an VREP simulation environment in their respective positions and orientations to form a scene. Additionally, `bml_vrep` can also be used to control the VREP simulations.

### 3.4 Performance Evaluator

Currently I have two metrics for evaluating the performance of the robot. The first metric is counting the number of times the robot successfully completes a given path trajectory, and the second is the energy expended by the robot's leg joints to traverse a given path trajectory. In order to calculate the energy expended, at every step of the simulation, I use



(a) Sample environment captured through the Kinect.



(b) Visualization of the mesh generated of the sample environment.

Figure 3.3: Mesh generation example.

VREP API calls to get the velocity of each leg  $v_t$  and the force being expended by that leg to maintain its velocity  $f_t$ . Then for each leg, I take the total energy expended by it across the simulation  $E = \sum_t v_t f_t$ . Finally, I take the sum of the energy used by all of the legs and output the resulting number. While this method is imperfect, as this result will return different values if the legs are contacting the ground at the base compared to somewhere closer to the motor, it serves as a good first order approximation when comparing the energy expenditure across multiple potential paths.

The performance evaluator communicates with VREP using ROS services. Each of the services use the `std_srvs/Trigger` message, as it is the only message that contains the string field to give the option of sending custom error messages. Now I will describe each of the ROS service messages that the performance evaluator uses. Note for all of these names `<robot_name>` is a place holder for the robot's name in VREP's simulation.

- `<robot_name>/path_plan`: Tells VREP to start path planning using OMPL.
- `<robot_name>/reset_pose`: Resets the robot position to the original position as specified in the \*.yaml file.
- `<robot_name>/drive`: With a given path, starts the driver code to have the robot drive along a given path.
- `<robot_name>/best_path`: Load the best path found into the VREP planner to try simulating again.

- `<robot_name>/drive_time`: If the path was successfully traversed, get the simulation time it took to traverse the path.
- `<robot_name>/energy_used`: If the path was successfully traversed, get the energy the robot used to traverse the path.
- `<robot_name>/path_found`: If the path was successfully computed (sometimes OMPL will fail and return a null path), return the path.

## 3.5 Driver

The driver code takes in a path and the robot's current position and outputs the commands in order to drive the robot. The path is broadcasted to other systems as a series of  $n$  points  $\{x_1, x_2, \dots, x_n\}$  where  $x_1$  and  $x_n$  are the start and end locations and the desired path of traversal is to pass through each point sequentially. The robot's position  $x_r$  and quaternion  $q_r$  are also broadcast. The next subsections will go into more detail on the path, and the driving controls.

### Path following

The driver keeps track of index  $i \in \{1, 2, \dots, n\}$  that represents the point  $x_i$  the robot is trying to reach. Once the robot comes within distance  $r_{path}$  of  $x_i$ ,  $i$  is incremented except for the last point  $x_n$ . If the robot is trying to reach  $x_n$ , the robot will continue driving until it

comes within distance  $r_{end}$  of  $x_n$ . Once the robot is within  $r_{end}$  distance of  $x_n$ , the robot is assumed to have reached its destination and the driver terminates. The pseudocode for this algorithm is at Algorithm 1.

---

**Algorithm 1** Path following algorithm.

---

```

1:  $path \leftarrow \{x_1, x_2, \dots x_n\}$ 
2:  $i \leftarrow 1$ 
3: while true do
4:   get  $x_r$ , the position of the robot
5:    $x_\delta \leftarrow x_i - x_r$ 
6:   Drive the robot
7:   if  $i \neq n$  and  $\|x_\delta\|_2 < r_{path}$  then
8:      $i++$ 
9:   else if  $i == n$  and  $\|x_\delta\|_2 < r_{end}$  then
10:    return

```

---

## Driving

In order to get the error in heading  $\theta$ , I first make some assumptions about the robot models I work with. For a robot with the quaternion  $q_r = 1$ , the forward direction is  $\hat{x}$  and the left direction is  $\hat{y}$ . Then let the vectors  $x', y'$  be the unit vectors representing  $\hat{x}, \hat{y}$  being rotated by the given quaternion of the robot  $q_r$ . Also let  $x_\delta = x_i - x_r$  be the direction of the next goal point in the path. Then the error  $\theta$  can be expressed as

$$\theta = \tan^{-1}\left(\frac{x_\delta^\top y'}{x_\delta^\top x'}\right)$$

In addition to calculating  $\theta$ , I also need a model for how the roach's turning speed based off of differential drive, I recorded the robots average turning radius across a variety of left and right motor speeds. The results of these plots is shown if Fig. 3.4. For a more detailed

description of how I collected this data, see the appendix at section A. From the plot it looks like the logarithm of motor ratio speeds is linearly related to the curvature of the path taken. Thus empirically the amount the robot turns is proportional to the exponential of the ratio of motor speeds.

With this relationship in mind, I used a P-controller with some gain  $p$  to drive the VelociRoACH where if  $\theta = 0$ , both of the motors run at some base velocity  $\omega_0$ . If  $\theta \neq 0$ , I keep one side of the roach at  $\omega_0$  and exponentially increase the angular velocity of the other leg with respect to the error angle  $\theta$ . The pseudocode for the driving code is at Algorithm 2.

---

**Algorithm 2** Code that drives the robots using differential drive.

---

- 1: Have some destination point  $x_i$
  - 2: Have some gain on your controller  $p$
  - 3: Get  $x_r$ , the position of the robot
  - 4:  $x_\delta \leftarrow x_i - x_r$
  - 5: Get  $q_r$ , the quaternion of the robot
  - 6: Compute  $x', y'$  from the quaternion  $q_r$
  - 7:  $i \leftarrow 1$
  - 8:  $\theta \leftarrow \tan^{-1}(x_\delta^\top y' / x_\delta^\top x')$
  - 9: **if**  $\theta > 0$  **then**
  - 10:     Set the left motor to  $\omega_0$
  - 11:     Set the right motor to  $\omega_0 \exp(|p\theta|)$
  - 12: **else**
  - 13:     Set the right motor to  $\omega_0$
  - 14:     Set the left motor to  $\omega_0 \exp(|p\theta|)$
-

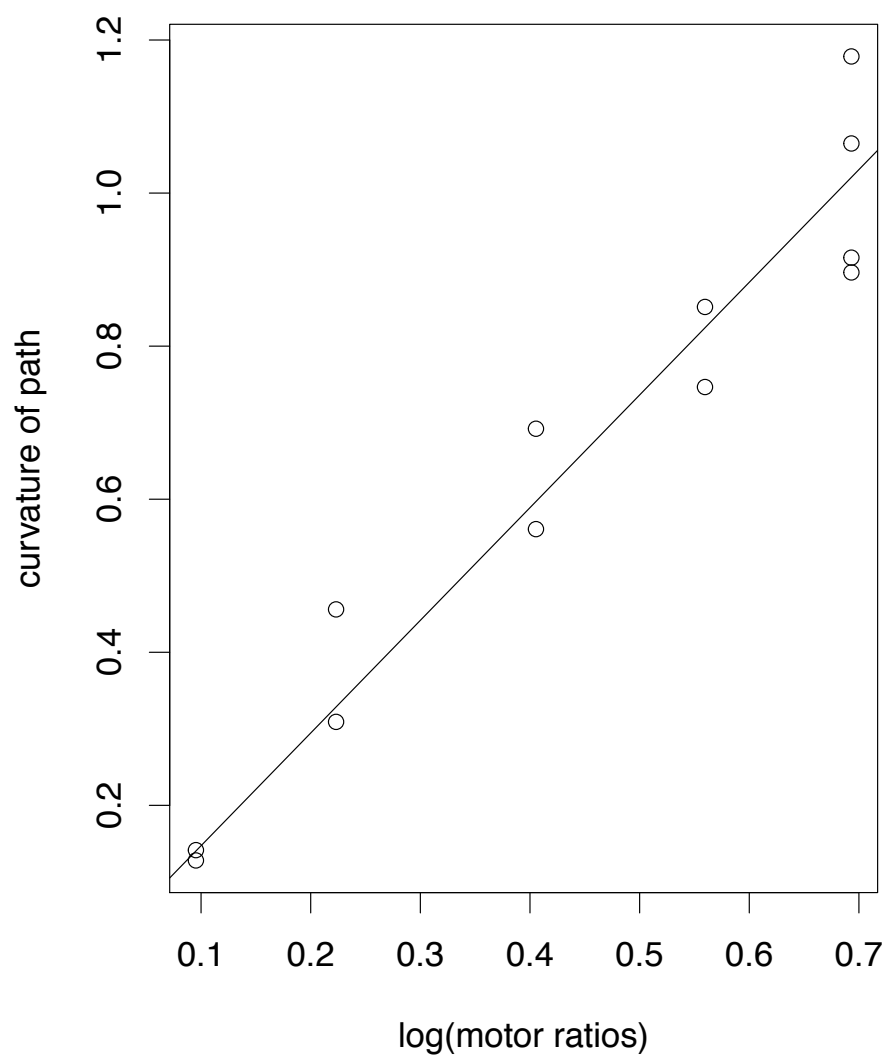


Figure 3.4: Plot of simulated VelociRoACH turning behavior at various motor ratios.

# Chapter 4

## VREP systems

VREP serves as the novel element when doing robotics path planning. Since VREP is a fully realized physics simulator, one can use VREP to account for complex interactions and planning algorithms in a clean and abstract manner. The purpose of these systems in VREP is to generate, test, and evaluate potential paths for the robots to traverse.

### 4.1 Introduction to VREP

Before I describe the corresponding parts of the block diagram in this section, I feel it is necessary to describe some important details about VREP in order to make the following sections more clear. This section serves as an introduction to the parts of VREP that are important to this report.



## Communication through ROS messages

VREP communicates externally using the ROS message and service systems. This is done because for real robots ROS is one of the common software packages when running real physical robots.

## Non-threaded scripting

When writing code in VREP, there is the option for threaded and non-threaded scripts. Threaded scripts run unsynchronized from the beginning of the simulation and non-threaded scripts run at every step of the simulation. Keeping in line with best practices, all of the code I have written in VREP is non-threaded. This reduces the complexity of the simulation scenes being run and makes the code overall more efficient.

## 4.2 Simulation Scene

In order to create a simulation scene that is compatible with ROS, VREP takes in the scenes given by the environment generator and generates a scene from it. One thing to note is that in order to be compatible with `bml_vrep`, there needs to be a ROS publisher broadcasting the simulation time so the ROS controller can properly control the VREP scene.

### 4.3 Robot Used

For the simulation scenes, we are using a simulated approximation of one of the VelociRoACH made by Andrew Pullin [14] which can be see in Fig. 4.1. The simulated roach contains one central body (approximated as a box) with 6 identical legs. To simulate the legs moving, the abduction and protraction angles change as the leg spins in order to match what the real VelociRoACH does.

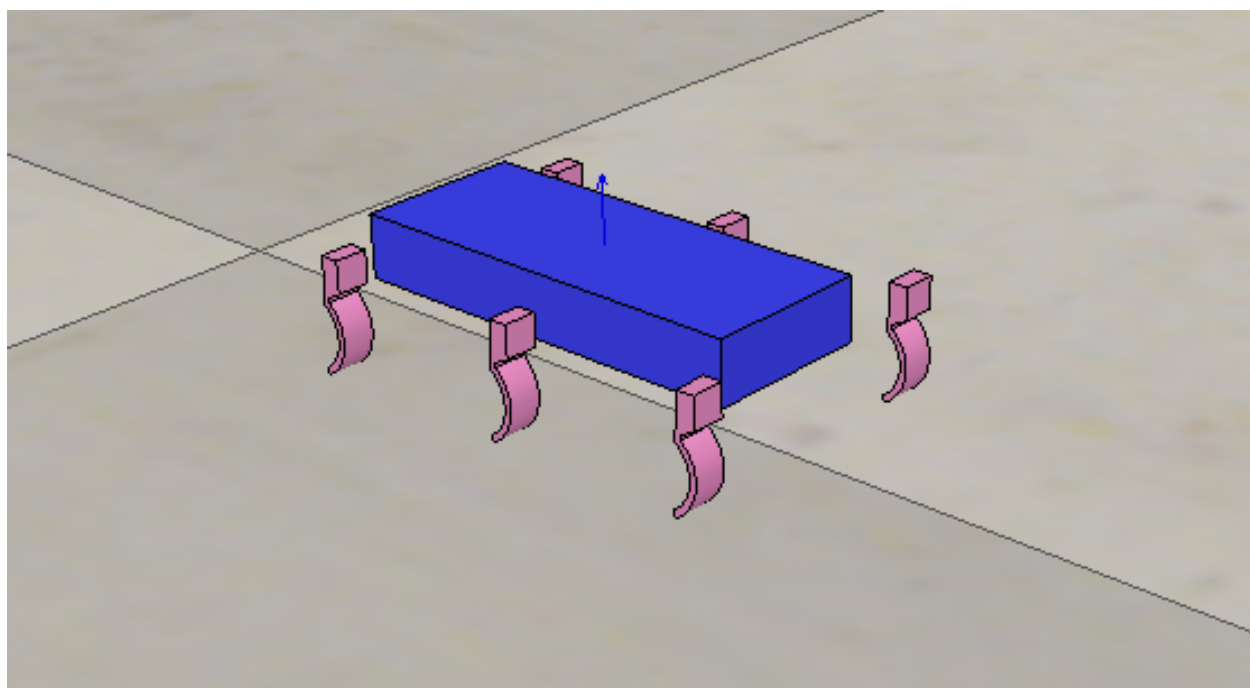


Figure 4.1: Picture of the simulated VelociRoACH.

## 4.4 OMPL

The Open Motion Planning Library [18] (OMPL). Is an open source repository containing state-of-the-art motion planning algorithms. To interact with OMPL, I am currently using VREP's API, which makes it very easy to interact with and test with in VREP at the cost of being very difficult to interact and test with outside of VREP. In order to use OMPL through VREP's interface, I am using a customized version of the "pose3D" state sampler (which incorporates the objects position and orientation) and state validation to find collision-free paths in which the robot crawls along the ground. The next few subsections will go into more detail about the custom state sampler and the state validation as well as how OMPL is being used.

### Custom State Sampler

Since the robot is restricted to be on the ground of the simulation, the search effectively covers a 2D surface in 3D space. From this, given a 3D search space for valid states, I randomly sample points in a in the 2D xy plane then project the robot to lie in the 3D. A cartoon of this process is shown if Fig. 4.2. With the projected plane, I rotated the robot such that the forward direction and the left direction of the robot is consistent with the projected robot.

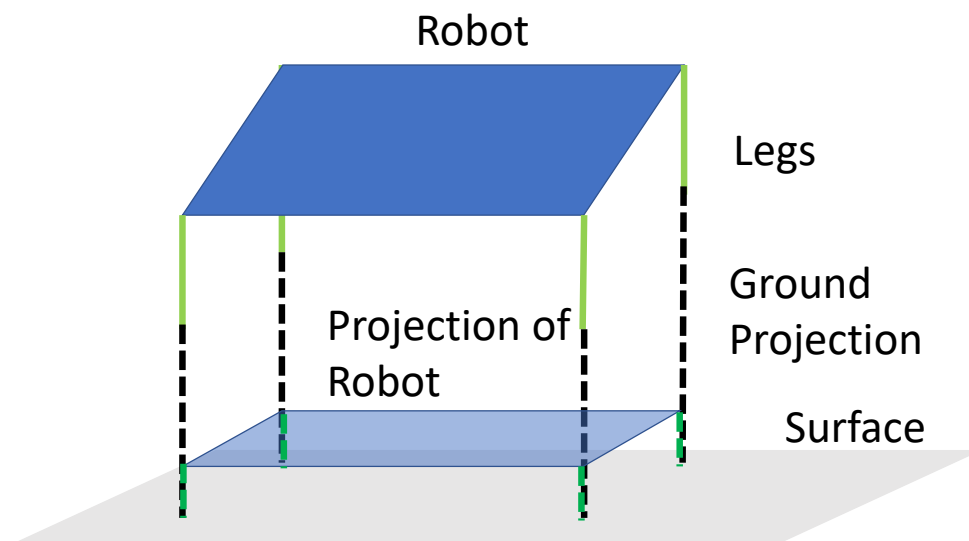


Figure 4.2: Example of projection of the robot onto a 2 dimensional surface.

## Custom State Validation

In order to test if there exists a valid path between two points, one needs to check whether the states along the path are valid. In order to do custom state validation, the model checks if the following conditions are met to see if a state is valid:

- If the robot is not colliding with any of the meshes.
- If the robot body is reasonably close to the ground.
- If the all of the robot legs are close to the ground (instead of one of the legs being over a pit for example).

## Using OMPL

From OMPL I use probabilistic roadmaps (PRM) [8] to do the path planning with the custom samplers and state validation. The reason why I am using PRM instead of some of the more common path finding algorithms such as rapidly-exploring random trees (RRT) [9] is because the VREP-OMPL interface only supports a small subset of the path planning algorithms if one uses a custom state sampler and state validator. PRM happens to be supported, while RRT is not. While other algorithms are supported such as Sparse Roadmap Spanners (SPARS) [4], in general they perform slower and less optimally than PRM.

## Chapter 5

# Software Based Experiments

Before discussing the experiments, this report will cover a version of this system that can be realized entirely through software. Since this system only relies on software, it is much easier and faster to design and run experiments than ones that rely on data from the physical world. The block diagram for software based experiments is shown in Fig. 5.1.

To replace the PCL mesh reconstruction, I instead use simplified meshes generated from OpenSCAD [13], an open source software for creating 3D objects. The next section will go into more detail about how I use OpenSCAD.

### 5.1 OpenSCAD

The primary way I have been using OpenSCAD is to generate simplified environment meshes such as the one in Fig. 5.2. OpenSCAD allows one to draw custom polyhedrons and

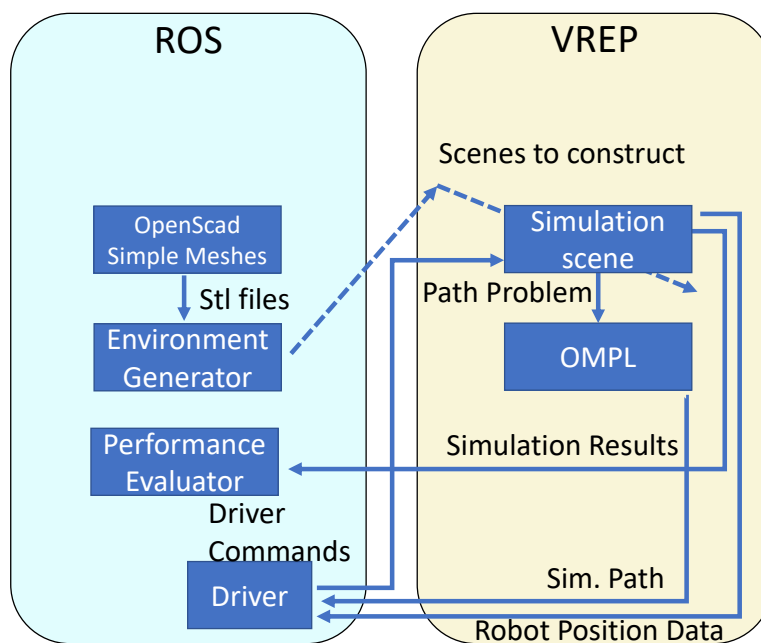


Figure 5.1: Simplified block diagram.

build environments using simple shapes such as spheres, cubes, and triangles. In addition, OpenSCAD can also generate height maps, which is another way to approximate a 3D environment. Consult the Appendix in section C for more information.

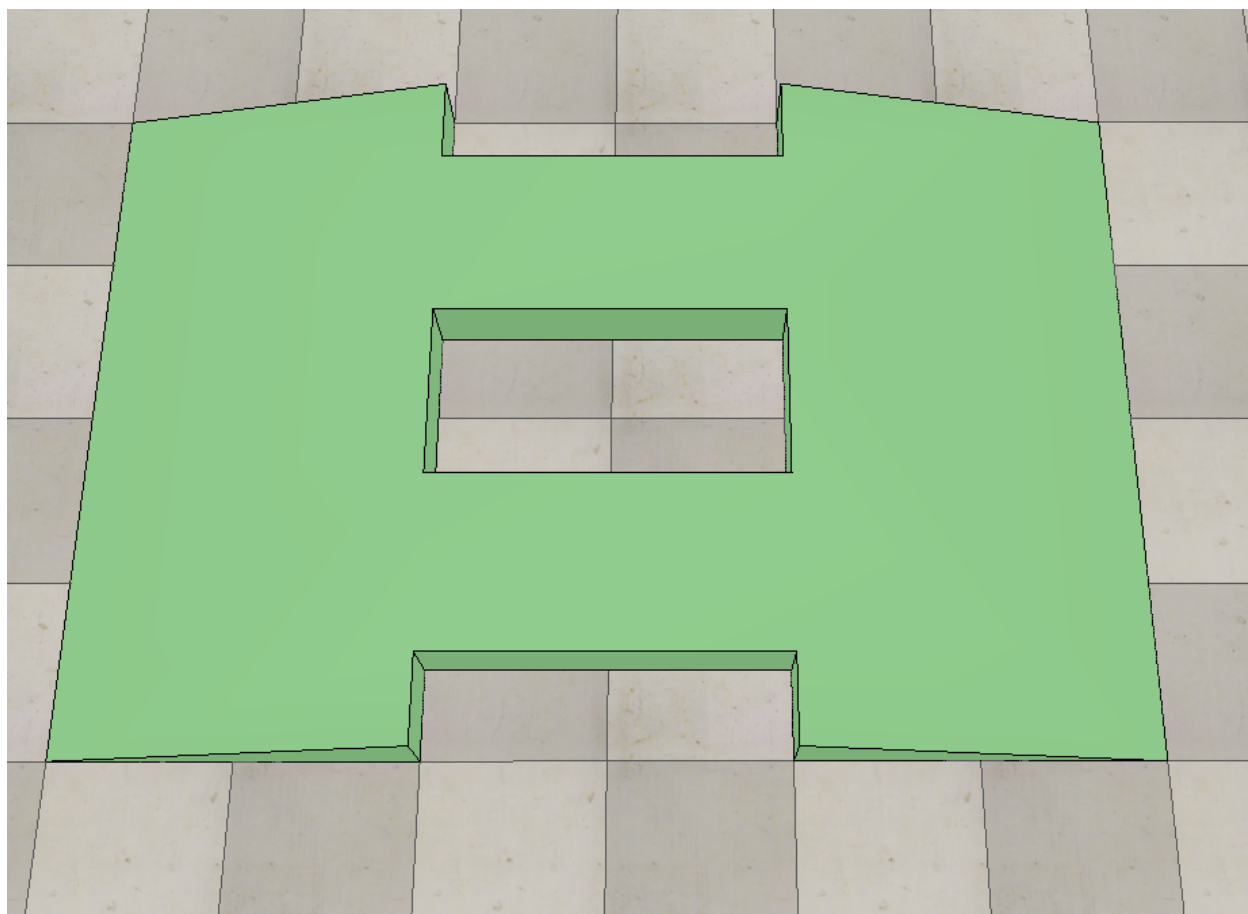


Figure 5.2: The double bridge, an example of a simplified mesh generated with OpenSCAD.



## Chapter 6

# Experiments

The purpose of these experiments is to test the efficiency of this system in various scenarios. In this report, there are a total of three experiments designed to try to characterize the system in a noiseless software-based system, a software-based system with artificial height variation, and a system that incorporates data from the physical world. For each of these systems, the goal is to design an environment in which the goal is for a robot to go from a start position to a goal position. For the planning of each path, the planner is only aware of the physical environment layout without any idea about the underlying dynamics or control model of the robot. However, despite this, the experiments will show that it is still possible to discern which paths are relatively safer and more energy efficient with the framework described in this report.

## 6.1 Picking the lowest energy path

The goal of the first experiment is to see how the system performs on a completely artificial environment. For this environment, I used a double bridge setup in Fig. 6.1. For a more detailed description of the environment used, see the appendix at section C. The goal of this experiment is for the robot on the left to navigate to a point within 5 cm of the robot on the right. Note that the robot on the right is a placeholder and has no collision physics. This experiment was to test how OMPL will generate different paths and their relative efficiencies. Since for this experiment the bridges are symmetrical, the top and bottom bridges should be chosen with roughly equal probability and the robots should traverse both bridges with roughly equal success rates and energy expenditure. In order to evaluate the relative successes of the paths, I generated a path using OMPL, had the robot run along the simulated path, then calculated the total energy that the robot expended as described in section 3.4.

## 6.2 Picking rough terrain

For this experiment, the goal was to see how the system performs on a system where artificial height variation is introduced. The reason behind adding the terrain roughness is to increase the uncertainty in trajectories beyond what geometric planner can predict. For this experiment, I am using a double bridge setup with bridges with artificial height variation. The setup can be seen in Fig. 6.2, a closeup shot of the bumpy terrain can be seen in Fig.

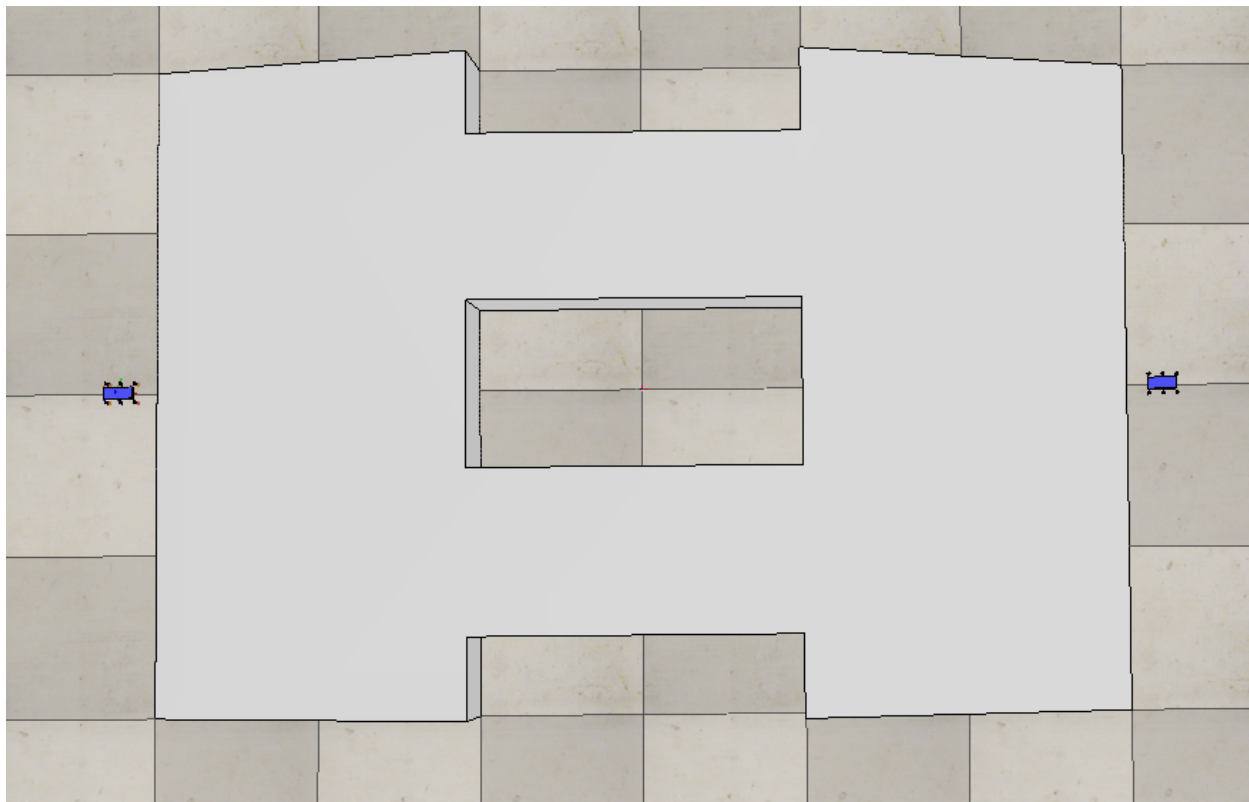


Figure 6.1: The setup for the experiment described in Section 6.1.

6.3 and a more detailed description of how exactly I generate the bumps is in the appendix at section C. The height of all of the bumps in the top and bottom bridge are randomly selected from the uniform distribution  $[0, 1.5 \text{ cm}]$  and  $[0, 1.0 \text{ cm}]$  respectively. Intuitively, the top path should be more difficult to cross, and the goal of this experiment is to gain some quantitative understanding on exactly how much harder the top path is to cross than the bottom one.

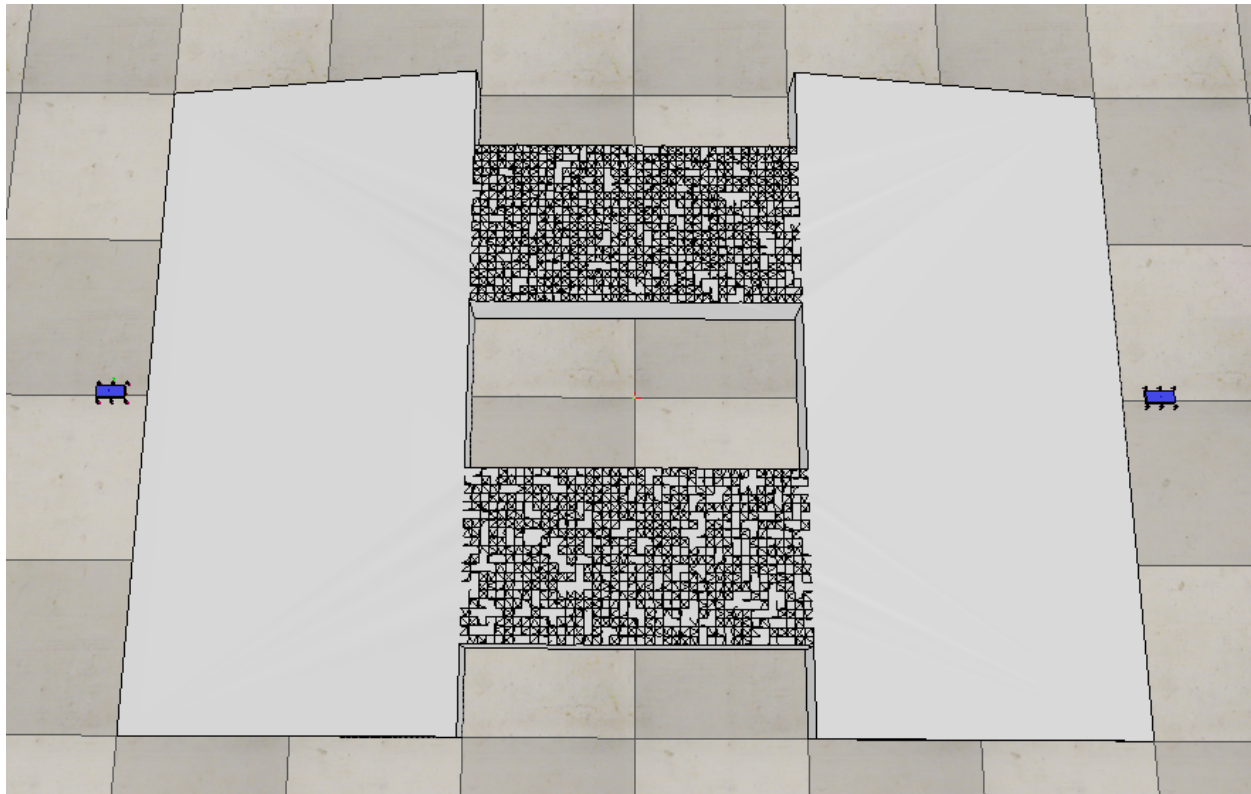


Figure 6.2: The setup for the experiment described in Section 6.2.

### 6.3 Running the robot on an environment generated from the physical world

For the last experiment, the goal was to test on a system from the physical world without extra artificial height variation. For this experiment, I have a setup of a single bridge made of plywood supported by cinder blocks using paper to smooth the transitions between sheets of plywood. The setup of the experiment can be seen in Fig. 6.4 and 6.5. Then I used the Kinect to generate a mesh to use for simulation in VREP which can be seen in Fig. 6.6. Note for this simulation, I used the path planning algorithm STRIDE [6] as in this case it worked better than using PRM.

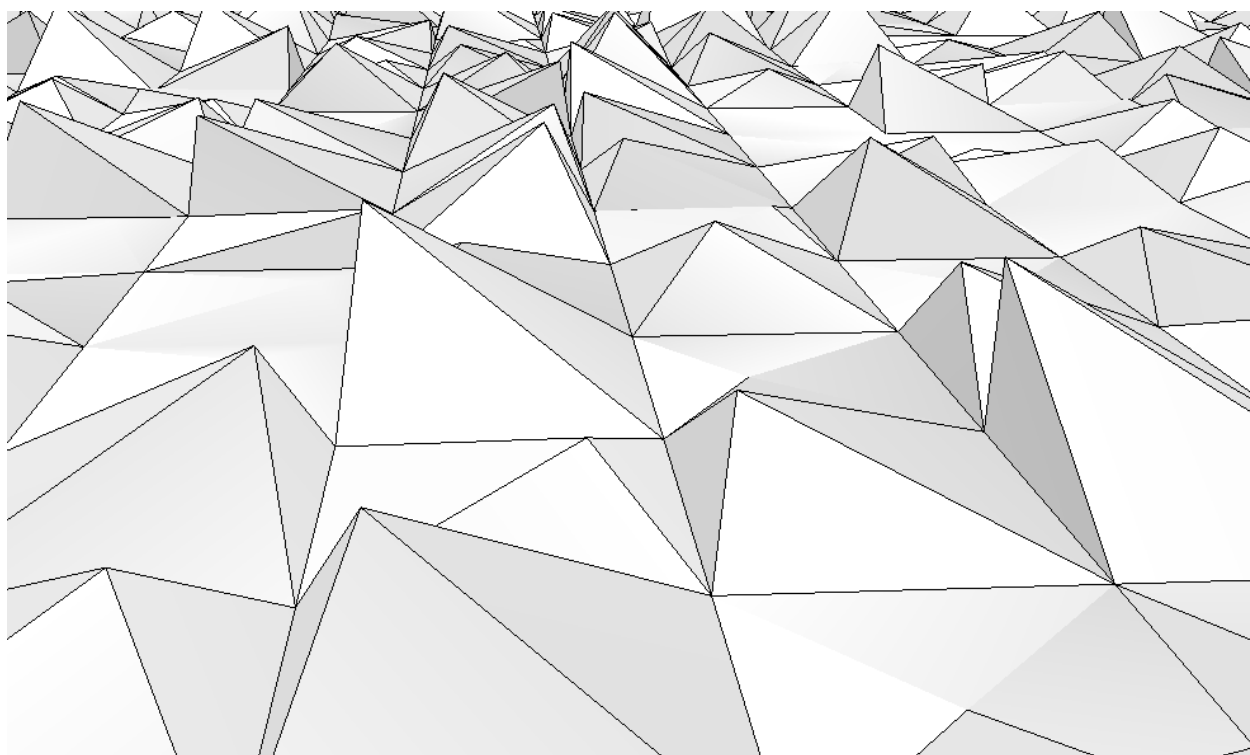


Figure 6.3: Close up rendering of the bumps used for the experiment described in section 6.2.



Figure 6.4: Picture of the setup in 6.3.





Figure 6.5: Close up of the ramp in 6.3.

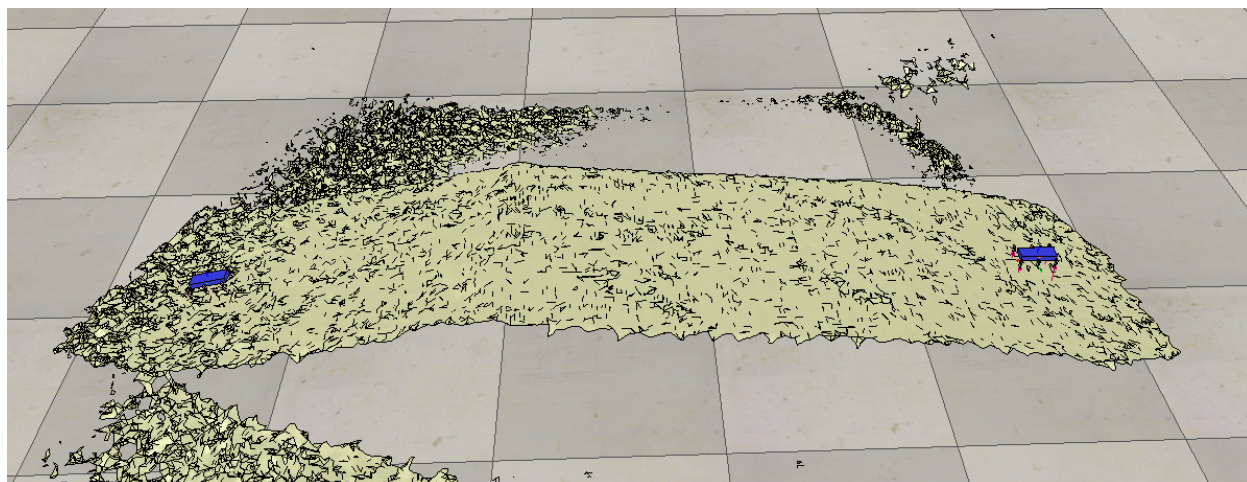


Figure 6.6: Screenshot from the physics simulation in 6.3. The robot is starting from the right and trying to reach the position marked by the dummy robot (blue) on the left.

# Chapter 7

## Results

For the results, the goal is to quantify how well the robots could successfully traverse the paths generated in the experiments. In order visualize the results better, the paths returned by OMPL are black while the paths the robot actually took are in blue. In the results data, I kept track of whether a path was successfully traversed and if it was traversed how much energy was used. For considering whether or not a path was successfully traversed, I checked to see whether or not the robot reached its destination within 60 seconds of simulation time. Since for the majority of trials the robot reached the destination in under 30 seconds, the failure cases were only ones in which the VelociRoACH fell off of the bridge. Finally, when quoting the amount of energy used, the statistics are only collected from the successful traversals. In this manner I can characterize which paths can be traversed efficiently and safely as well as get a general sense of how OMPL performs without incorporating information about the robot's dynamics.



## 7.1 Picking the lowest energy path

For this experiment I ran OMPL PRM until it generated 20 paths for each of the top and bottom paths. The results represent the control case, where there is a smooth terrain for both paths and are summarized in Table 7.1. Note the amount of energy is a relative unit, so only comparisons in energy used are meaningful.

From the results, it seems that the results from running the robot along the top and bottom path is functionally identical. This makes sense since the paths are symmetric. One thing to note is that some of the randomized paths are poor and give roughly 20% higher energy expenditure costs than normal.

	top bridge	bottom bridge
number of trials	20	20
successful traversals	18	19
average energy expended	1.082	1.111
std. dev energy expended	0.0596	0.0909
minimum energy expended	1.009	1.009
maximum energy expended	1.192	1.371

Table 7.1: Results of the experiment described in section 6.1.

## 7.2 Picking rough terrain

For this experiment I ran OMPL until I had a total of 25 paths for both the top and bottom paths. The results are summarized in Table 7.2. Note the amount of energy is a relative unit, so only comparisons in energy used are meaningful.

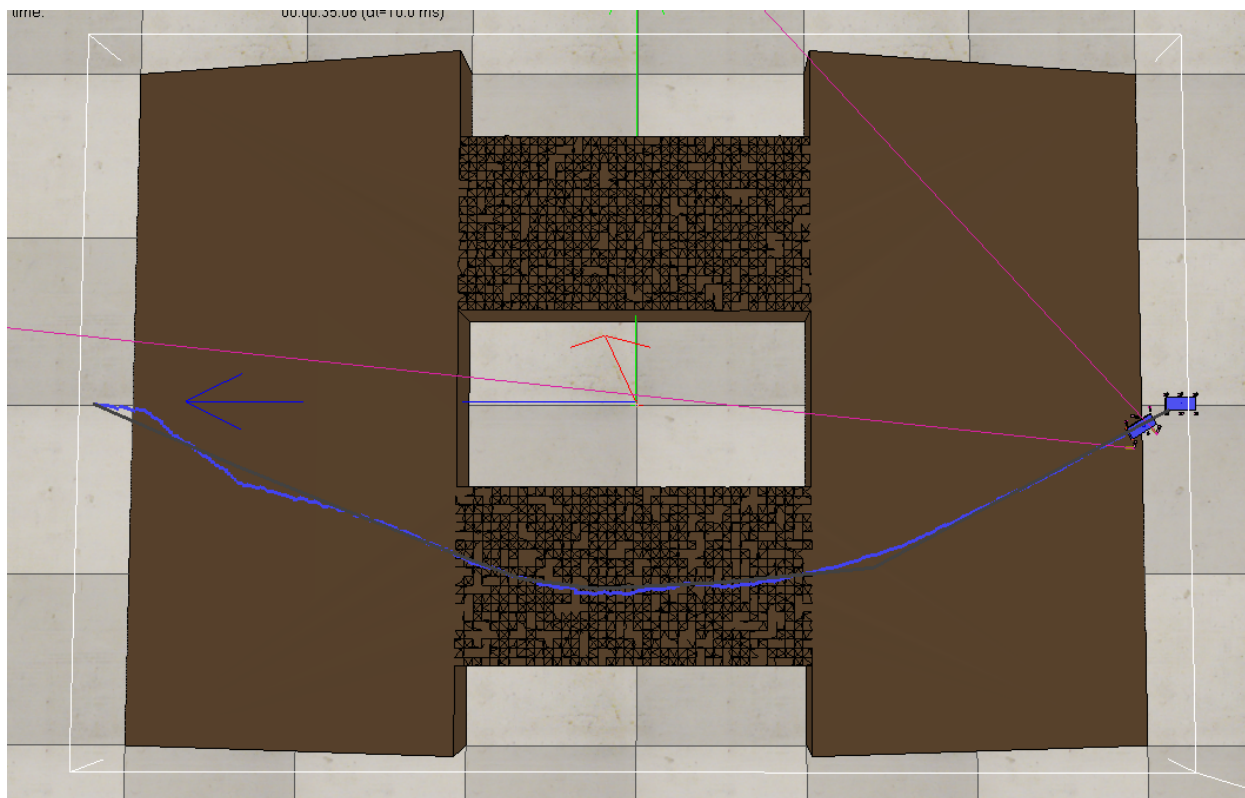


Figure 7.1: Example path generated from section 6.2. In this case the robot succeeds in traversing the path.

In general, when the robot successfully traversed the paths it had smooth paths that were centered on the bridge, and the unsuccessful paths were ones in which the path was too close to the edge. Examples of successful and unsuccessful paths that were traversed can be seen in Fig. 7.1 and 7.2.

From the results, it seems clear that the top path is much worse than the bottom path. Both the failure rate and the energy cost is roughly twice as large for the top path as opposed to the bottom path. The failure rate for this experiment is much higher than the previous experiment due to the terrain roughness making it much harder for the differential drive

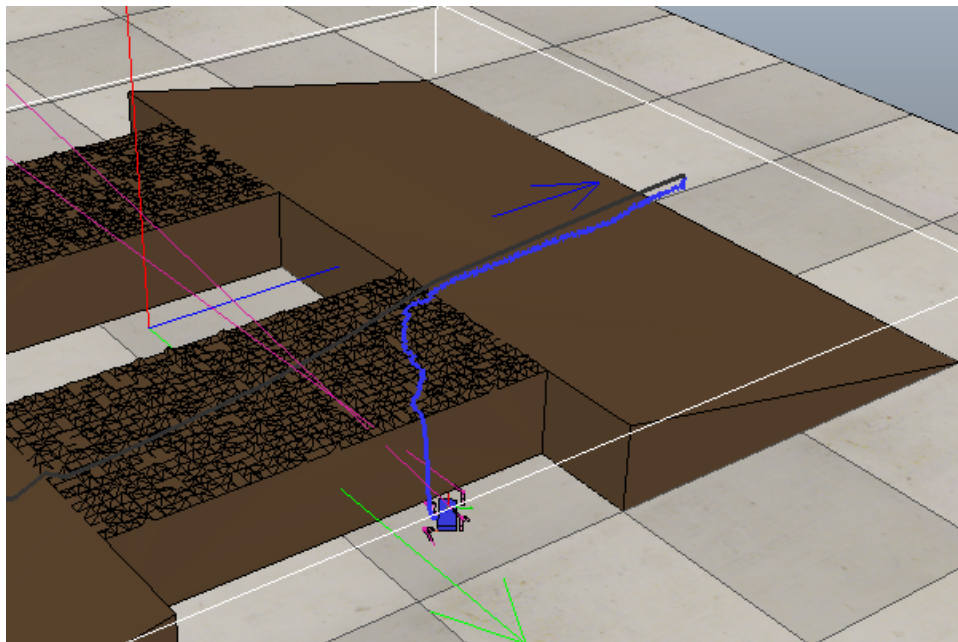


Figure 7.2: Example path generated from section 6.2. In this case the robot fails in traversing the path.

controller to stay close to the intended path.

	top path	bottom path
number of trials	25	25
successful traversals	19	23
average energy expended	1.210	2.424
std. dev energy expended	0.1133	1.7228
minimum energy expended	1.058	1.288
maximum energy expended	1.452	7.154

Table 7.2: Results of the experiment described in section 6.2.

### 7.3 Running the robot on an environment generated from the physical world

For this experiment I placed the robots 2m apart from each other and ran 25 trials. The results are summarized in Table 7.3. From looking at the results, the path planning algorithm returned a fairly high success rate. Indicating that it is possible to drive the roach across the bridge in simulation. The lowest energy and safest paths tended to be ones that had paths that were closer to the center of the bridge and relatively straight like the paths in Fig. 7.3. Conversely, paths that tended to have sharper turns and were closer to the edge of the bridge tended to fail, such as the path in Fig. 7.4.

parameter	value
number of trials	25
successful traversals	20
average energy	0.938
std. dev. energy	0.329
min. energy	0.656
max. energy	2.061

Table 7.3: Results of the experiment described in section 6.3.

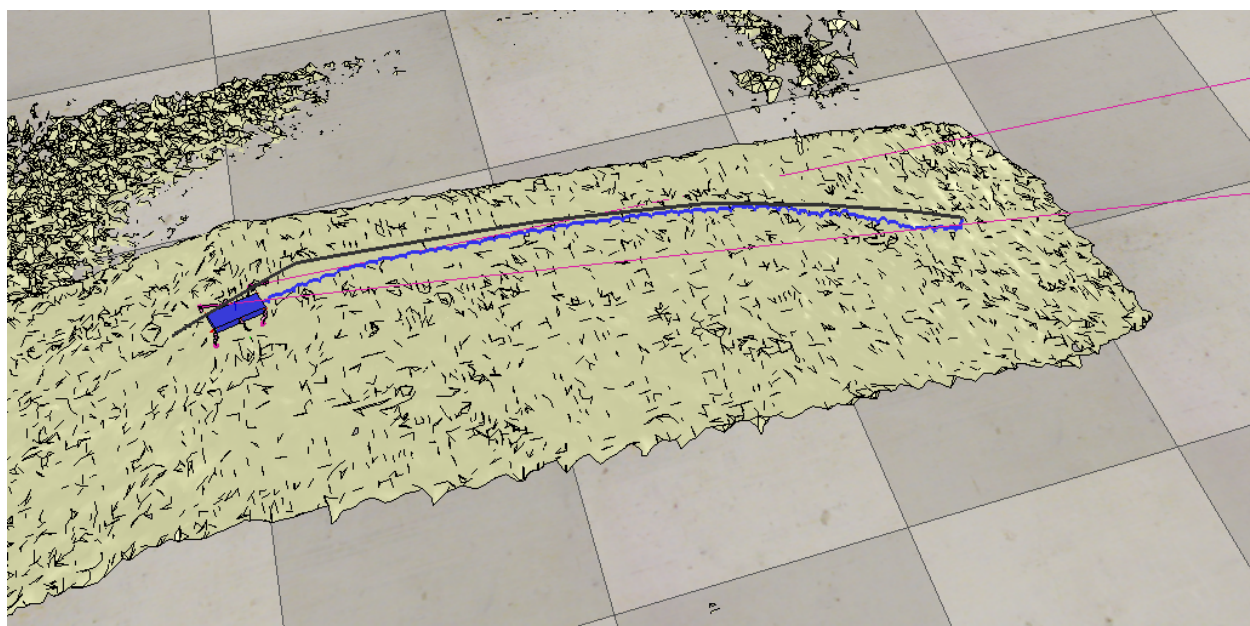


Figure 7.3: Example path generated from section 6.3. In this case the robot succeeds traversing the path.

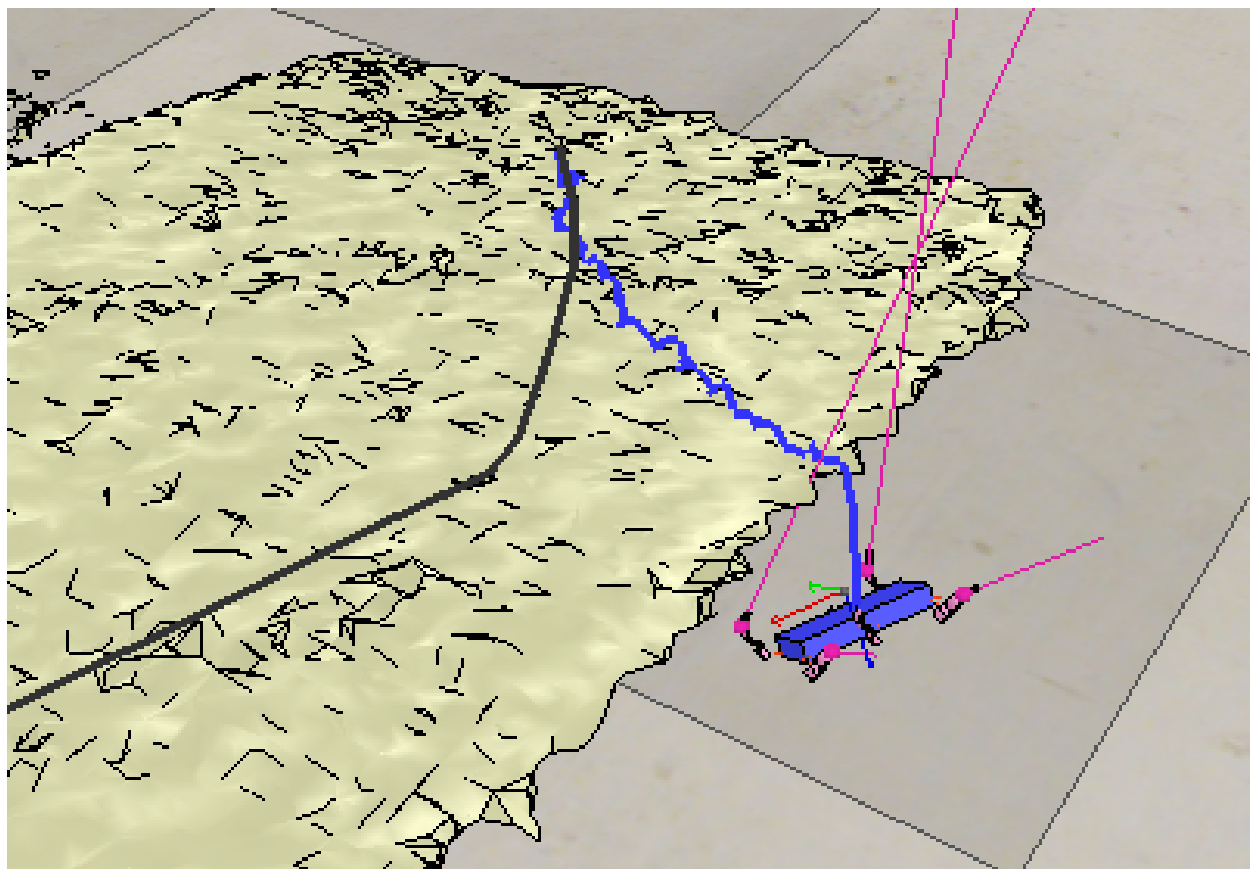


Figure 7.4: Example path generated from section 6.3. In this case the robot fails traversing the path.

# Chapter 8

## Conclusion

### 8.1 Limitations to the system

The main bottleneck to this system is the vast amounts of computation time required to run the physics simulation. The experiments relying on purely simulated data take on the order of 1 minute and the experiments on a physical environment take on the order of 3 minutes, making them too slow to run in real-time. However, this system is heavily parallelizable, as given a path or an environment one can have multiple instances of a single environment being run at once.

Another limitation to this system is it relies heavily on the accuracy of the physics simulation models. As physical tests were not conducted, it is still an open question how accurate the physics simulations are at predicting the behavior of real physical robots.

## 8.2 Future Work

One avenue of work would be to implement the red arrows as seen in Fig. 1.1. With this, it will be possible to run physical tests using this framework, allowing for the validation of the simulation results.

Another possibility would be to move the OMPL outside of VREP, perhaps into ROS or another system entirely. The reason for this is that while using VREP's OMPL interface is easy to write, it is very limiting in the sense of what one can change. These limitations prevented me from using more common planning algorithms like RRT to instead use only supported algorithms like PRM. Additionally, due to how VREP is built it is difficult to do version control on the code, making it more difficult to implement changes to the algorithm from a software perspective.

## 8.3 Overall Conclusion

This report has shown a framework that incorporates physics simulations in a feedback loop to aid in real-time robot planning. By using this framework, one can run experiments using simulated and physical environments using a planner that does not need any knowledge about the robot's dynamics. Furthermore, one can run these experiments using code that can control both the physical and the simulated robots.





## Appendix A

# Modeling the turning radius of the robot

For this section, let  $\omega_l, \omega_r$  be the angular velocities of the left and right sides of the robot respectively. For each data point in Fig. 3.4, I ran the VREP simulated robot and at each time step took its position and rotation. For calculating total rotation of the robot, I looked at the change in rotation at each time step and summed them. I used a similar process for calculating the distance travelled. The result of a sample run can be seen in Fig. A.1 Then to get the curvature of travel, given the robot travelled a total of distance  $d$  and turned a total number of  $n$  revolutions the curvature of the path taken was  $2\pi n/d$ . Note that this formula assumes that the robot travelled in perfect circles, which was not the case. The robot tended to drift somewhat during all of the tests. However, since the purpose of this test was to determine a reasonable empirical model, I ignored this error and the resulting P-controller seems to work under this assumption.

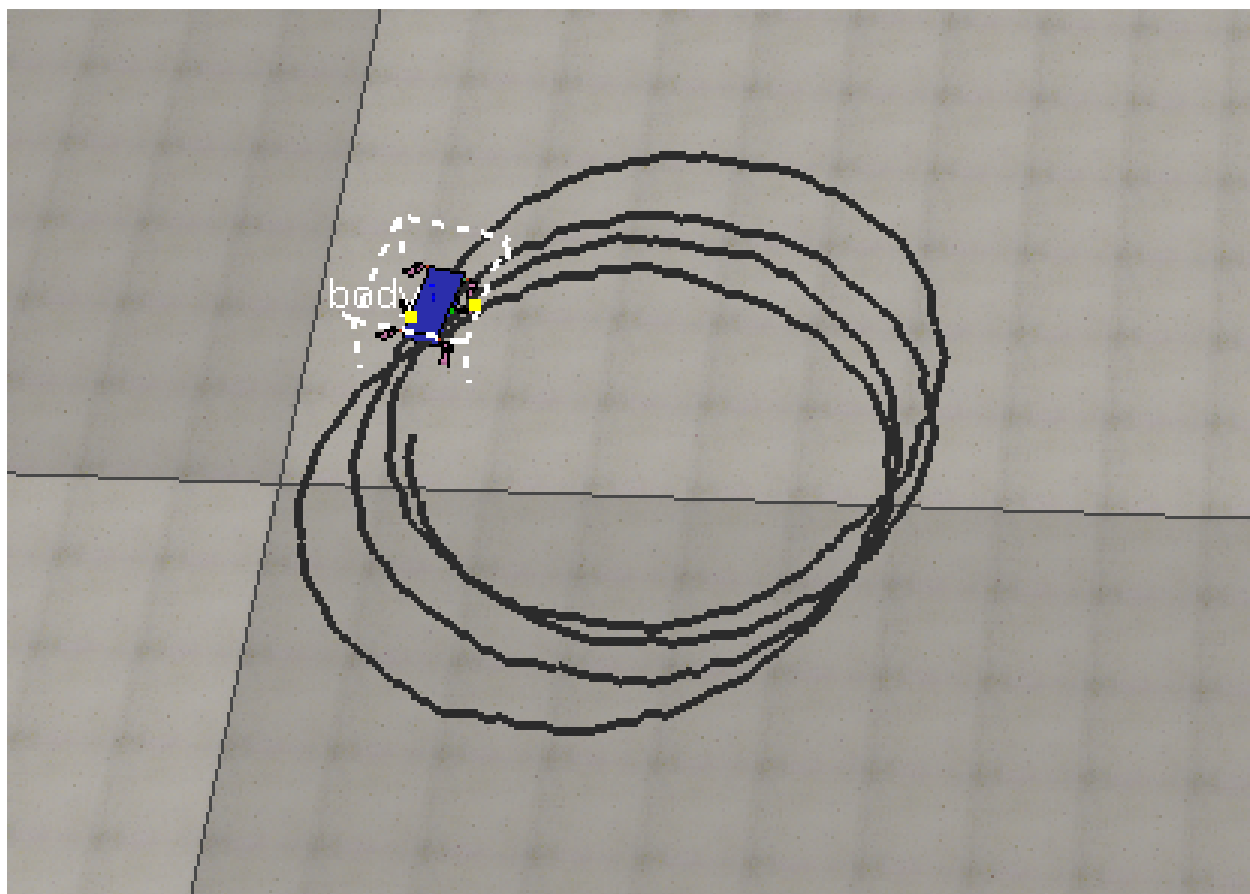


Figure A.1: Example run that was used to generate data for Fig 3.4.

## Appendix B

### Sample \*.yaml configuration file.

Below is a sample \*.yaml configuration file that sets up a sample path planning scene over a double bridge described in section C. An annotated screenshot of resulting scene in VREP can be found in Fig. B.1.

```
1. scene_file: "empty.ttt"
2. objects: [
3.   {
4.     model_filename: "ros_clock.ttm"
5.   },
6.   {
7.     model_filename: "stl_object.ttm",
8.     stl_filename: "double_bridge.stl",
```

```
9.     position: [0.0, 0.0, 0.0],
10.     orientation: [0.0, 0.0, 0.0, 1.0],
11. },
12. ]
13. robots: [
14.   {
15.     model_filename:
16.       "6leg_crawler_non_threaded.ttm",
17.     position: [0.0, -1.6, 0.045],
18.     orientation: [0, 0.0, 0, 1],
19.   },
20.   {
21.     model_filename:
22.       "6leg_crawler_dummy.ttm",
23.     position: [0.0, 1.6, 0.045],
24.     orientation: [0, 0.0, 0, 1],
25.   },
26. ]
```

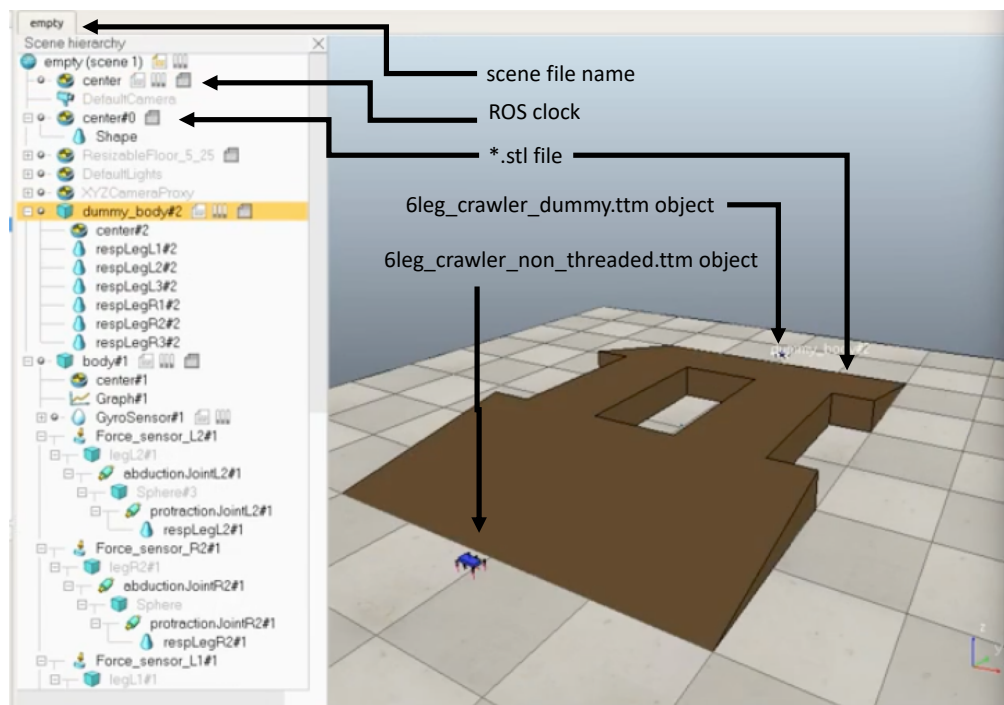


Figure B.1: Annotated screenshot of the scene described in B.

# Appendix C

## OpenSCAD shapes

In this section of the appendix will go over some of the main shapes I use in the simplified meshes.

### C.1 Double Bridge

I chose to use the double bridge model because it is geometrically simple to describe and offers path planning algorithms two distinct ways to traverse from one side to the other.

The double bridge can be thought of as two mirrored right triangular prisms that are joined by some bridge component, as seen in Fig. C.1. As for the bridge itself, it is composed of two identical cuboids spanning the gap as shown in Fig. C.2.

Keeping with the dimensions described in figures C.1 and C.2, the dimensions of the bridge are in Table C.1.

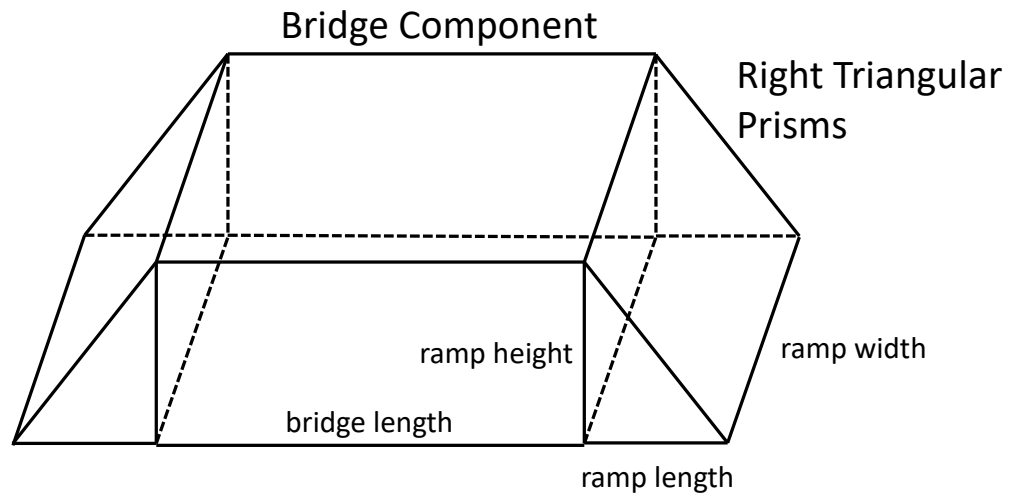


Figure C.1: Conceptual drawing of the ramps of the double bridge.

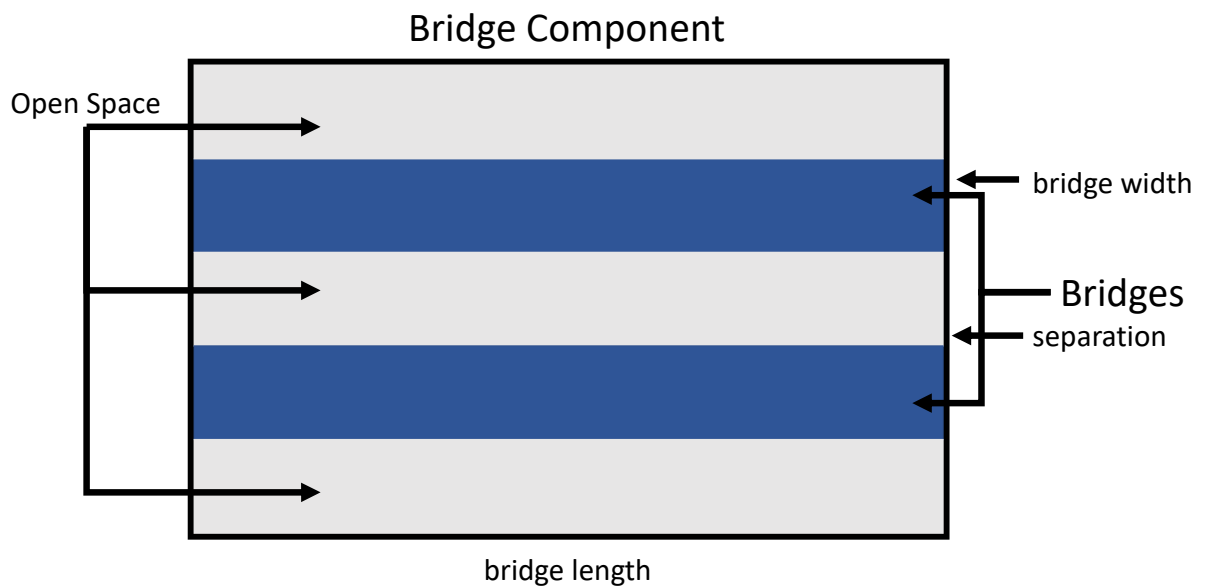


Figure C.2: Conceptual drawing of the bridges of the double bridge.



bridge dimension name	length (m)
ramp length	1.0
ramp width	2.0
ramp height	0.2
bridge length	1.0
bridge width	0.5
separation	1.0

Table C.1: Dimensions of the bridge.

## C.2 Introduction of random bumps

For the test introducing height variation into the bridges, I used the same double bridge except added some pyramid-shaped bumps to the bridges. For each of the bumps I used a pyramid with a square base with the peak at some random height in the range  $[0, max\_height]$  and selected the location of the peak to be over the center of the pyramid. A schematic of the random pyramid can be seen in figure C.3. For each bridge there is a  $20 \times 40$  grid of these pyramid bumps across the bridge with varying maximum heights. This makes each pyramid base  $2.5 \text{ cm} \times 2.5 \text{ cm}$ .

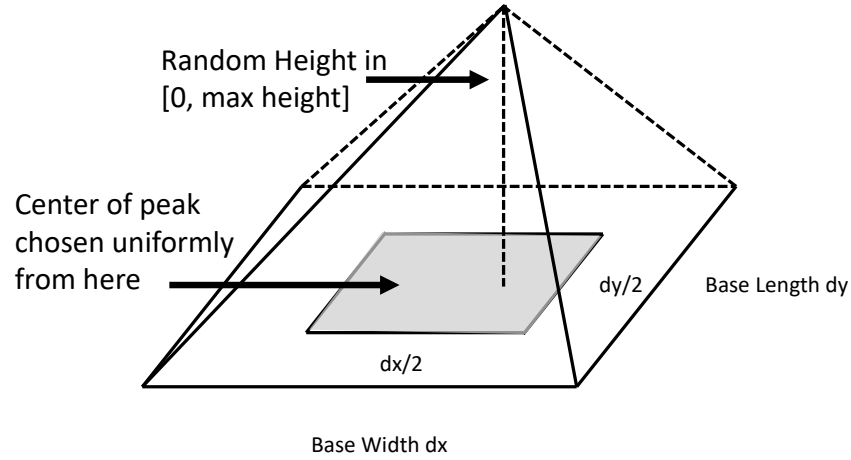


Figure C.3: Drawing of the bumps used when constructing the double bridge with artificial height variation.

# Bibliography

- [1] Mossaad Ben Ayed, Lilia Zouari, and Mohamed Abid. “Software In the Loop Simulation for Robot Manipulators”. In: *Engineering, Technology & Applied Science Research*. Vol. 7. 5. Springer Fachmedien Wiesbaden, 2017, pp. 2017–2021.
- [2] Austin Buchan. *bml\_vrep*. 2016. URL: [https://github.com/biomimetics/bml\\\_vrep](https://github.com/biomimetics/bml\_vrep) (visited on 04/24/2018).
- [3] Ignasi Clavera, Anusha Nagabandi, Ronald S. Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. “Learning to Adapt: Meta-Learning for Model-Based Control”. In: *CoRR* abs/1803.11347 (2018). arXiv: 1803.11347. URL: <http://arxiv.org/abs/1803.11347>.
- [4] David Coleman, Ioan Alexandru Sucan, Mark Moll, Kei Okada, and Nikolaus Correll. “Experience-Based Planning with Sparse Roadmap Spanners”. In: *CoRR* abs/1410.1950 (2014). arXiv: 1410.1950. URL: <http://arxiv.org/abs/1410.1950>.

- [5] S. Demers, P. Gopalakrishnan, and L. Kant. “A Generic Solution to Software-in-the-Loop”. In: *MILCOM 2007 - IEEE Military Communications Conference*. Oct. 2007, pp. 1–6. DOI: 10.1109/MILCOM.2007.4455268.
- [6] B. Gipson, M. Moll, and L. E. Kavraki. “Resolution Independent Density Estimation for motion planning in high-dimensional spaces”. In: *2013 IEEE International Conference on Robotics and Automation*. May 2013, pp. 2437–2443. DOI: 10.1109/ICRA.2013.6630908.
- [7] Duncan W. Haldane, Kevin C. Peterson, Fernando L. Garcia Bermudez, and Ronald S. Fearing. “”Animal-inspired Design and Aerodynamic Stabilization of a Hexapedal Millirobot””. In: *ICRA* (2013).
- [8] Lydia E. Kavraki, Petr Svestka, Jean-Claude Latombe, and Mark H. Overmars. “Probabilistic Roadmaps for Path Planning in High-Dimension Configuration Spaces”. In: *IEEE Transactions on Robotics and Automation* 12.4 (Aug. 1996), pp. 566–580.
- [9] J. J. Kuffner and S. M. LaValle. “RRT-connect: An efficient approach to single-query path planning”. In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*. Vol. 2. 2000, 995–1001 vol.2. DOI: 10.1109/ROBOT.2000.844730.
- [10] René Linssen, F. Uphaus, and J. Mauss. “Software-in-the-Loop at the junction of software development and drivability calibration”. In: *16. Internationales Stuttgarter*

- Symposium*. Ed. by Michael Bargende, Hans-Christian Reuss, and Jochen Wiedemann. Wiesbaden: Springer Fachmedien Wiesbaden, 2016, pp. 451–465.
- [11] Zoltan Csaba Marton, Radu Bogdan Rusu, and Michael Beetz. “On Fast Surface Reconstruction Methods for Large and Noisy Datasets”. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. Kobe, Japan, May 2009.
- [12] Duy Nguyen-Tuong and Jan Peters. “Model learning for robot control: A survey”. In: 12 (Apr. 2011), pp. 319–40.
- [13] *OpenSCAD*. 2018. URL: <http://www.openscad.org/index.html> (visited on 04/25/2018).
- [14] Andrew Pullin. *Simulations*. 2018. URL: <https://wiki.eecs.berkeley.edu/biomimetics/Internal/Simulations> (visited on 04/26/2018).
- [15] Coppelia Robotics. *VREP*. 2018. URL: <http://www.coppeliarobotics.com/index.html> (visited on 04/24/2018).
- [16] Radu Bogdan Rusu and Steve Cousins. “3D is here: Point Cloud Library (PCL)”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China, May 2011.
- [17] Russell Smith. *ODE*. 2015. URL: <http://www.ode.org/> (visited on 04/24/2018).

- [18] Ioan A. Sutan, Mark Moll, and Lydia E. Kavraki. “The Open Motion Planning Library”. In: *IEEE Robotics & Automation Magazine* 19.4 (Dec. 2012). <http://ompl.kavrakilab.org>, pp. 72–82. DOI: 10.1109/MRA.2012.2205651.
- [19] Gabriele Vandi et al. “Development of a Software in the Loop Environment for Automotive Powertrain Systems”. In: *Energy Procedia* 45 (2014), pp. 789–798.
- [20] Zummy. 2016. URL: <https://wiki.eecs.berkeley.edu/biomimetics/Main/Zummy> (visited on 04/24/2018).