UNIVERSITY OF CALIFORNIA SAN DIEGO

Learning to control a Prism bot

A thesis submitted in partial satisfaction of the
requirements for the degree
Master's of Science

in

Engineering Sciences (Mechanical Engineering)

by

Francis Joseph

Committee in charge:

2018

DEDICATION

To my family for their support in everything I do.

EPIGRAPH

Fide et Labore

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

This thesis could not have been completed without the support, advice and facilities provided by Bioinspired Robotics and Design Lab at UCSD. I would like to thank Prof Mike Tolley for providing me guidance through the course of the research and helping me complete my thesis.

I thank Prof Sonia Martinez for providing direction for the research keeping in mind the constraints of the Prism bot.

Prof Nikolay Atanasov's guidance in improving the experiments and results in modeling the problem was very helpful and greatly appreciated.

The research would not be possible without a robot. William Weston-Dawkes provided the design and fabrication method to build the Prism bot. His knowledge and guidance on the research was crucial to building this robust design. I thank him for all his help, guidance and support.

The UCSD Coordinated Robotics Lab were immensely helpful in sharing their resources and providing access to the Optitrack system. I would like to thank Prof Thomas Bewley for giving permission to use this resource.

I would like to thank the Gravish Lab and Prof Nick Gravish for helping me take high speed video of the Prism bot moving.

ABSTRACT OF THE THESIS

Learning to control a Prism bot

by

Francis Joseph

Master's of Science in Engineering Sciences (Mechanical Engineering)

University of California San Diego, 2018

Professor Michael T Tolley, Chair

In this work, a data-efficient method is applied to learn a model of the dynamics of a self-folding robot driven by vibration. These robots can be autonomously fabricated and deployed, but complex dynamics lead to challenges in modeling. Learning from a limited set of observed experiments, a model is developed to control the locomotion of the robot along a desired trajectory. The model is fit assuming a probabilistic Gaussian process and a neural network. The two methods are benchmarked against a differential drive algorithm.

Chapter 1

Introduction

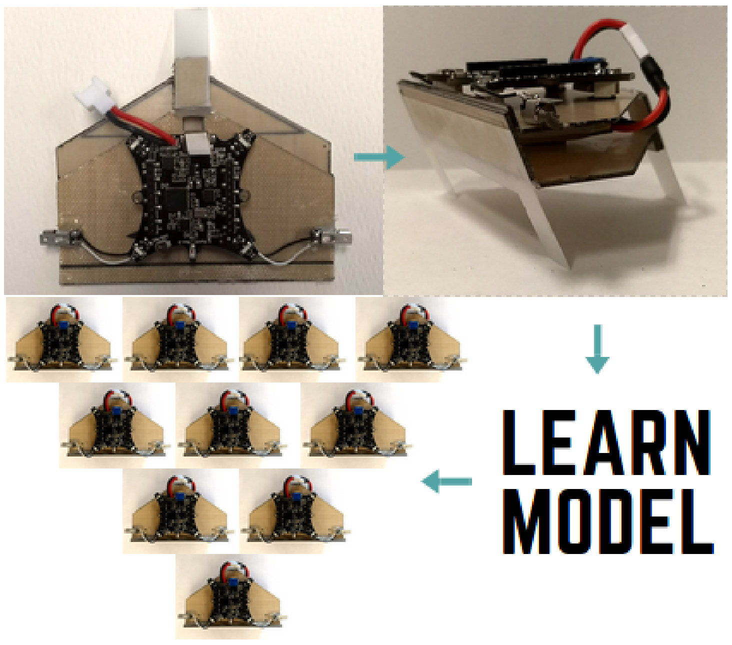


Figure 1.1: Self assembly to swarms

Prism bot is a self assembling, laminate manufactured robot[1]. The laminate design enables the use of rapid manufacturing techniques, which along with the low cost electronics make it ideal for swarms[2]. The long term goal is to have a large number of Prism bots with similar morphology doing different tasks in a coordinated manner. Self-folding[1] provides a method to autonomously deploy these robots. The reason why we can self assemble the entire robot is that we use vibration as the locomotion modality. Vibration based locomotion provides actuators which can be easily incorporated to the structure via adhesives. Vibration based actuation is complex and challenging to model. In this thesis I seek to find a data efficient method to model this locomotion and control the Prism bot.

Modeling vibration based locomotion are possible using finite element analysis but it is computationally intensive which makes simulation time and resource intensive. Understanding the motion using experiments on the real robot comes with the overhead of time required to collect the data. This motivates me to use a method which requires very few samples. I separate the problems of finding the model and the control strategy to give me a better understanding of the motion. On examining the preliminary data I notice a Gaussian trend in the displacement in respect to the inputs to the motors. This motivated me to fit the motion model as a probabilistic dynamic model using a Gaussian Process(GP). I then look to find the performance of a neural network and compare the two function approximations of the dynamics of the Prism bot.

After learning a model I chose to adopt a Markov Decision Process(MDP) to determine a policy for the robot to follow a given trajectory. An MDP is an optimization technique to find a suitable policy, implemented using dynamic programming. I formulate a cost function to follow the trajectory while maximizing speed.

In general, a Prism bot could have the actuators in any position and orientation. In this version of the Prism bot I place the actuators on either side of the robot and perpendicular to the edge. This gives me an intuitive method to steer the robot using differential drive by sending a higher pulse width modulation(PWM) value to a particular motor. I use this simple controller as a benchmark to see how well I perform on learning a method to control the Prism bot.

Chapter 2

Background

Self-folding via laminate manufacturing[3] provides a customizable and rapid fabrication process for building robots. The manufacturing cost and simple design provides practicality to building a large number of these robots. Vibration based locomotion[4] using an eccentric rotating mass is a simple actuation method as it can be directly attached to the housing of the robot structure. Vibration based robots which have angled legs provide increase linear velocity. Previous work[1] has shown that the Prism bot travels quickly with a max velocity of around 0.04m/s.

This locomotion for vertical and horizontal linear (x-z plane) motion[4] helped develop an initial model[1] for the Prism bot. The model assumes that the body is parallel to the ground at all times. The legs are modeled as mass-less rigid rods. The friction was modeled as Coulomb dry friction. The model does provide a good result for longitudinal velocity but is not as effective in predicting angular velocity of the robot.

Differential drive based algorithms for control have been proposed for running similar vibrational robots[5] where one motor is run a higher frequency to provide turning. This performs well when the motors are arranged on either side of the robot. When the motors are staggered this algorithm becomes less intuitive to implement.

Model-based learning methods provide a relatively simple function approximation[6] [7] [8]. They are yet to be shown to perform well in complex locomotion tasks due to their limitations on being generalizable. A lot of work has been done in learning probabilistic dynamics[9] to learn probabilistic forward models and use them to learn feedback controllers for generating a policy for locomotion.

Optimization techniques have been used to improve gait and reducing time for manually finding parameters for gait and controllers. These methods include stochastic gradient descent [10], genetic algorithms [11], and Bayesian optimization [12] for optimizing a control policy. These methods are data efficient but show poor results in encoding surface properties. These methods also show better results on slower moving robots.

Deep reinforcement learning such as Q-learning [13], actor-critic methods [14], and policy gradients [15] are able to learn complex skills for simulated robotic locomotion. The high sample complexity of purely model-free algorithms make them difficult to use for learning on a real robot. Since the collection of the data must be done on the real robot, collecting enough samples for the robot to learn is not practical. There are a few reinforcement learning algorithms which have been shown to work with a few demonstrations[16] but those are for highly specific tasks requiring expert demonstration.

In this work I show that I can learn models on a real robot with very little data. I can then use this model to provide controlled locomotion of the robot. I demonstrate path following of the robot and benchmark this method against a differential drive algorithm.

Chapter 3

System Description

3.1 Structure

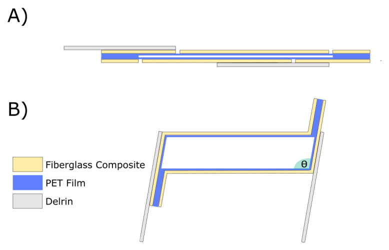


Figure 3.1: Prism bot: Layers and design (Combined support and design layers) Adhesive layers omitted.

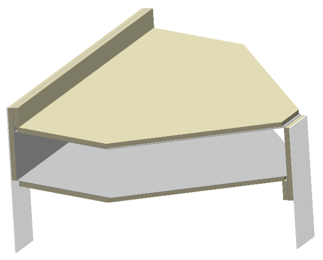


Figure 3.2: Prism bot structure: structure without electronics.

Our robot design consists of nine stacked material layers comprising of four separate materials types: fiberglass composite, delrin, high-temperature PET film, and high-temperature silicone glue-on-a-roll adhesive (Fig. 3.1). We are able to fold the robot into the defined shape by having suitable patterns on the surface using a laser cutter. We developed the design using the foldable robotics package/popupCAD.

3.2 Electronics and software

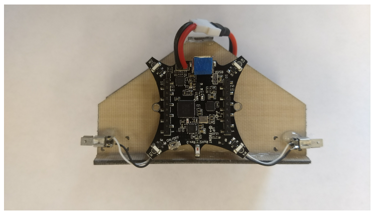


Figure 3.3: Top view of the Prism bot

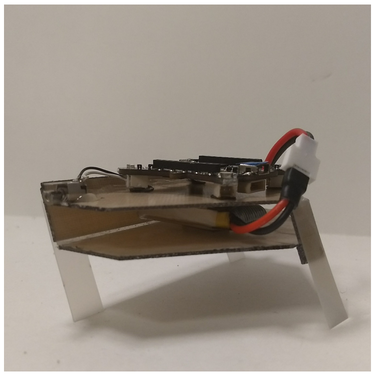


Figure 3.4: Left side view

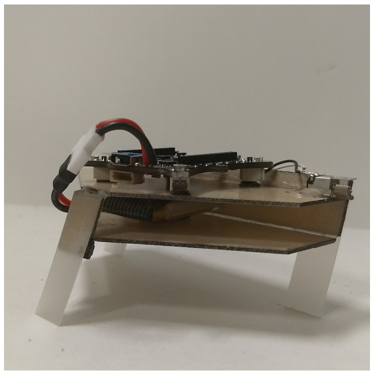


Figure 3.5: Right side view

The Prism bot is actuated using eccentric rotation masses. This makes the requirements for the electronics similar to that seen in a quadcopter. A quadcopter has four motors run through a board which is light and compact as quadcopters need to optimize for weight and size. I leveraged the developments in this field and used the electronics seen in the Crazyflie Nano 2.0 quadcopter. I have a STM32F405(ARM) MCU with modified firmware to run the motors on the Prism bot. The Prism bot has a Bluetooth LE and a 125 channel radio with the capability to communicate with a range of around 1km (line of sight) with the help of the Crazyradio PA connected to the computer. I have an IMU which has a 3 axis gyro, a 3 axis accelerometer, a 3 axis magnetometer and a high precision pressure sensor. I run two motors with eccentric masses which can run upto 10400rpm. A 250mA, 3.7V single cell battery can run the robot for about 20mins continuously.

A new firmware has been flashed to the MCU. The firmware deserializes the PWM values it receives via the radio and sends it to the respective motors. I used the Multiwii

Serial Protocol to send messages to the board.

I run the whole experiment through ROS leveraging the ROS infrastructure to send, record and debug commands to the robot. A ROS node is run on a computer which determines a suitable PWM value to be sent to each motor on the Prism bot. A ROS driver transmits the data to the Prism bot. The position of the robot is tracked using the Optitrack cameras. The position is streamed using the VRPN (Virtual Reality Peripheral Network) protocol. I deserialize the position through another ROS node. Since both the commands and position are transmitted via ROS I can store the messages in ROS using one clock. This helps to synchronize the position of the robot and the commands sent to the robot.

Chapter 4

Technical Approach

4.1 Experimental Setup

The Prism bot had four spherical reflective markers for detection by the Optitrack system. The markers were placed at different heights and at asymmetric locations such that the markers can be discerned as unique points of the rigid body. I have four cameras which are brought close to each other to decrease the volume to only what is required. The LED on the cameras are reduced so that the light from one camera doesn't give false detections to the other cameras. I saw that the infrared cameras detect the balls really well in low ambient light conditions and had limited false detections from other reflective surfaces (Fig: 4.1).

I collected the positions in x, y, z and orientation as a quaternion from the Optitrack system. I determined v_x, v_y and v_z from the first derivative of the position. I calculated the pitch (ϕ_p), yaw (ϕ_y) and roll (ϕ_r) angle by converting the quaternion to euler angles.

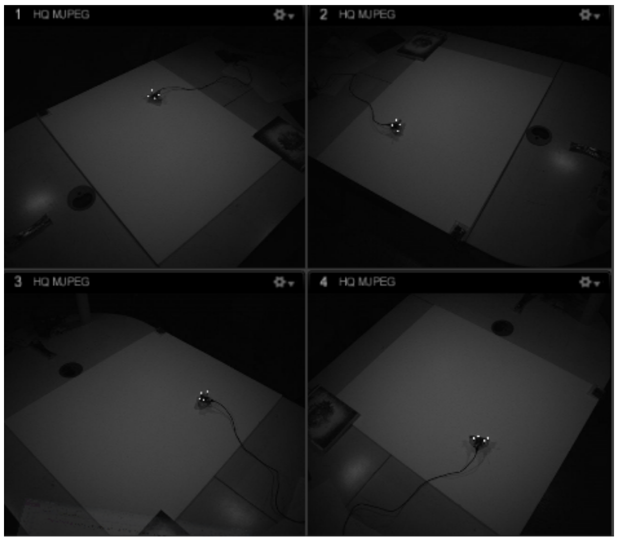


Figure 4.1: Optitrack detecting the reflective balls

I provide different PWM values to the two motors on the Prism bot. The position and orientation of the robot defines the state of the robot.

4.2 System Properties

I initially looked into aspects such as inertia and how fast the robot reacts to each command. I also look into the repeatability of the robot ie whether the same inputs produce the same outputs.

The commands are sent at different frequencies. The Optitrack system has a mean error of around 0.5mm per marker. The motion of the robot limits the frequency at which I can send commands. If I send commands at a very high frequency I will not be able to accurately detect the actual motion of the robot for that input. The VRPN data and the commands are sent asynchronously via two separate ROS nodes. I assume that the time of flight to send the commands for my experimental environment is negligible. The time at which I send the command from the computer is considered as the time the robot receives the command. The packets are assumed not to be lost during transmission. The latency from the Optitrack system through the VRPN transmission is also considered negligible.

4.3 Data processing

I store the data from the ROS nodes in a bag file. The command and position data streams run asynchronously. I synchronize the data using the system clock of the system on which ROS core is run. The raw data provides me with the PWM inputs to the motors and the position and orientation of the robot. The orientation of the robot is in the form of a quaternion which is converted to Euler angles.

The data is then further processed to be used in training a model for the forward motion of the robot. The position of the robot is converted from the global frame to the robot frame. Where the x axis is oriented in the forward direction of the Prism bot, y axis is to the left and z axis is vertically up. The angular orientation is measured with respect to the x axis measured in the anticlockwise direction.

4.4 Model

4.4.1 Gaussian Probabilistic Model

I notice that when I transform the motion of the robot to the robot frame, the data follows a Gaussian trend. I fit the data for discretized inputs to a forward probabilistic model of x, y and rotation about the z axis in the robot frame. The input PWM values are discretized between 0 and 30,000 with an interval of 10,000 between each discrete input. The angle has to be constrained to be between $-\pi$ to π radian to ensure we get the right variance and mean. I use the method of moments to fit the data to the respective discretized Gaussian.

$$x_{t+1} = x_t + f(x_t, u_t) \tag{4.1}$$

where, $f(x_t, u_t)$ is a Gaussian distribution which I fit for a discrete set of inputs.

$$f(x_t, u_t) \sim \mathcal{N}(\mu, \sigma) \tag{4.2}$$

The Gaussian models are fit for each set of discretized inputs for change in x, y and yaw. The change in the z direction is negligible and is not considered in modeling. Similarly the roll and pitch are negligible that they are also not modeled. The knowledge of the motors PWM value is all that is required for the Gaussian to predict the change in x, y and yaw in the robot frame.

4.4.2 Neural Network

I ran a multilayer neural network[6] to give me a function $f(x_t, u_t)$ as shown in the equation below. The multilayer neural network had 2 hidden layers with tanh activation function. I trained the dynamics model for 75 epochs, using the Adam optimizer with a learning rate of 0.001 and a batchsize of 1000. I map the function as shown in the equation below:

$$x_{t+1} = x_t + f(x_t, u_t) \tag{4.3}$$

$f(x_t, u_t)$ is the function I approximate using the neural network. The output of the network is $(\Delta x, \Delta y, \Delta\phi_y)$.

The inputs to the neural network are the inputs to the two PWM values to the two motors. The discretized inputs in the training set are between 0 and 30,000 with 10,000 intervals. The inputs are divided by 10,000 before they are sent to the network. I found this has similar results as normalization of the data. This is just simpler but cannot be used if we have other inputs to the network such as velocity, etc. Since I made the assumption that the inertia of the system is low, the current velocity of the robot is not important in the model. The output is mapped to the change in x, y and yaw in the robot frame. The change in z is negligible as there is little movement in the z axis. There is very little amount of pitching or rolling in the movement of the robot hence these changes are not mapped as well.

4.5 Simulator

I developed a simulator to quickly test whether the learned models predicts a general trend in the movement of the robot. This qualitative assessment provides me with a good method to iterate through multiple models.

The simulator developed using python also provides a test bed to find open loop policies for various trajectories which can later be tested on the robot.

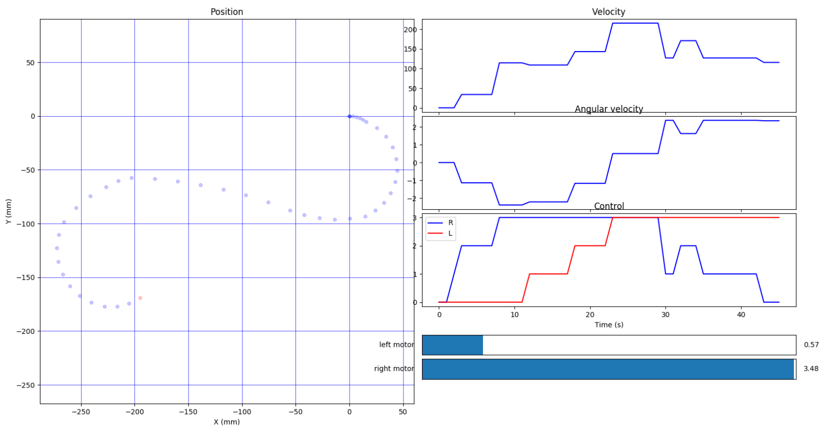


Figure 4.2: Simulator showing movement of the robot using the learned model with a slider for inputs to each motor. The range of the slider is from 0 to $3.5 * 10^4$

4.6 Applying a controller

I apply a controller to see how well I can control the robot with available model or if further improvements need to be made. I benchmark the quality of the model with a differential drive algorithm for following a trajectory with feedback to determine the efficacy of my study.

4.6.1 Differential drive

I use a differential drive algorithm to provide a benchmark of what I can achieve with a controller with no knowledge of the dynamics of the system. The implementation of the algorithm is mentioned in Algorithm 1.

Algorithm 1 Differential drive algorithm for trajectory following

```
1: procedure DIFFERENTIAL DRIVE (robot (x, y, yaw ($\phi_y$)), way points [$x_w$,$y_w$])
2:     while not at goal do
3:         Get the closest point to trajectory
4:         Find $e_d$ the perpendicular distance between the robot and the trajectory
5:         Find $e_\theta$ angle between the line and the robot
6:         if robot to the right of trajectory
7:         then $e \leftarrow e_d *$ scale_factor $+ e_\theta$
8:         else $e \leftarrow e_d *$ scale_factor $- e_\theta$
9:         left leg PWM $\leftarrow$ PWM$_{\text{nom}}$ $- e *$ gain
10:        right leg PWM $\leftarrow$ PWM$_{\text{nom}}$ $+ e *$ gain
11:    return PWM value for the left and right motor
```

4.6.2 MDP trajectory following using the learned model

I use an MDP[17] to determine a control policy for the model which I learned. I formulate the MDP with a cost for each state and find the best policy for the next 10 steps. I initialize the policy with a random policy and then iterate using two steps until

I converge. The first step is to evaluate the policy and the second step is to improve the policy.

Policy evaluation:

$$J^{\pi}(i) := g(i, \pi(i)) + \sum_{j=1}^{n} P_{ij}^{\pi(i)} J^{\pi}(j)] \tag{4.4}$$

Policy improvement:

$$\pi'(i) = \arg \min_{u \in U(i)} \left[g(i, u) + \sum_{j=1}^{n} P_{ij}^{\pi(i)} J^{\pi}(j) \right] \tag{4.5}$$

where, π is the policy, u is the control, $g(i, u)$ is the state cost, P_{ij} is the transition probabilities and $J(i)$ is the cost vector.

The state cost is given by,

$$g(i, u) = e_{cte} * w_1 + e_\theta + e^{w_2/\left(\text{PWM}_{\text{left}} + \text{PWM}_{\text{right}}\right)} \tag{4.6}$$

where, e_{cte} is the cross track error which is the shortest perpendicular distance to the figure 8 trajectory. e_θ is the orientation error of the robot with respect to trajectory. I use weights w_1 and w_2 to give a suitable method to add the errors. I add the PWM values I send to each motor and take its inverse as I want to ensure that the robot is moving as fast as possible and not concerned about power consumption. I have seen from the test data that increasing the PWM values increase the speed of the robot.

I choose a discount factor γ as 0.9 to consider future state costs. I get the motion model for the MDP from the model that I learned on the robot. I reevaluate the MDP on every update of the robot's position.

Chapter 5

Results

5.1 Response of the robot

The response of the robot was very quick and moved immediately on actuation. I verified this using the Optitrack system. I plotted the velocity and commands versus time (Fig 5.1). I saw that the velocity of the robot increases to its final velocity almost instantaneously. This helps make the assumption that the inertia of the system is very low. In modeling the motion of the robot, the current velocity had very little significance to the change in velocity of the robot.

5.2 Repeatability of the robot

In order to learn the dynamics or forward kinematics of any robot I needed to ensure the robot is repeatable or predictable. This means on sending a set of inputs the

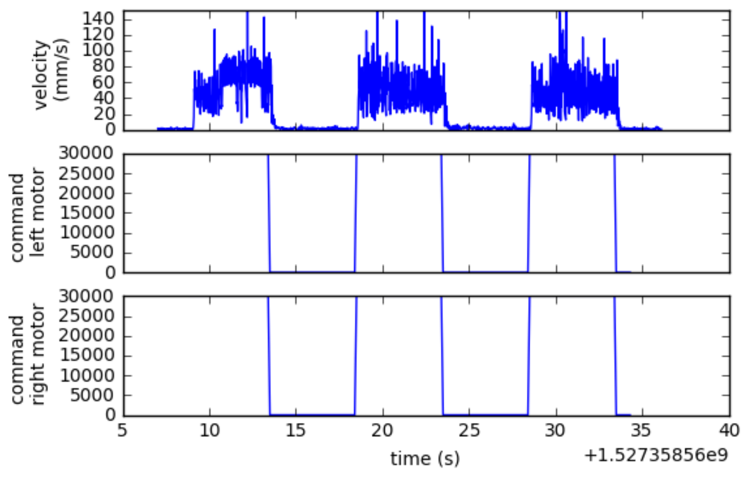


Figure 5.1: Dynamic response of the system.

robot will show a similar behavior with a high probability. This was essential to set up the learning experiment and to help any approach converge to a good function approximation of the motion of the robot. I did three trials, by sending the same command to the same robot starting from the same point. I plot the motion in figure 5.2, this plot shows that the robot demonstrates repeatability. The robot's trajectory is predictable for about 100mm. This ensured that I can predict the trajectory for the robot with a reasonable probability for around 0-2 seconds.

5.3 Models

The change in x, y and yaw was plotted in histograms (Fig 5.3). I saw that when I transform the changes in the state of the robot (x, y, yaw) to the robot frame the data had a Gaussian trend(Fig). This lead me to find how well a probabilistic dynamic model in the form of a Gaussian(Fig 5.3) would perform over a neural network. The simplicity

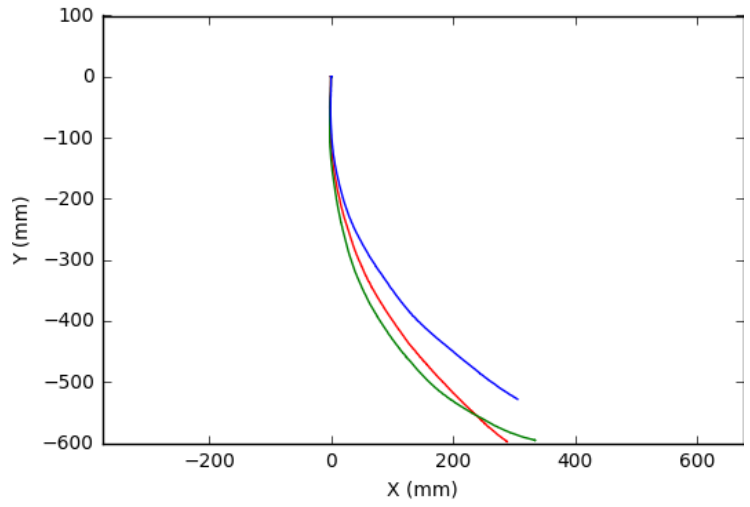


Figure 5.2: Position of robot versus time. The same command is sent to the robot starting from the same position. We see the repeatability of the robot to follow the same trajectory over short time horizons.

of the Gaussian function and the small amount of data required to fit the function helped me learn a model quickly. The changes in x, y and yaw showed a definite trend. The angle must be placed between $-\pi$ and π to be processed in the Gaussian fitting algorithm. The simulator provided reasonable qualitative understanding of how the robot would move for different inputs to the robot.

I then use the same data and learned a model using a neural network as a function approximation of the motion model. The learned model showed a decrease in the loss function which is the mean squared error. The model showed higher accuracy for data collected at higher frequency but this did not correlate with performance on the actual robot. This could be because at higher frequencies the robot moved less and the data made it seem that the model was more accurate. I used tensorboard to plot the learning of the model [18]. The accuracy and the loss function are more for a qualitative understanding of whether the model is being learned and I was not able to infer any quantitative results

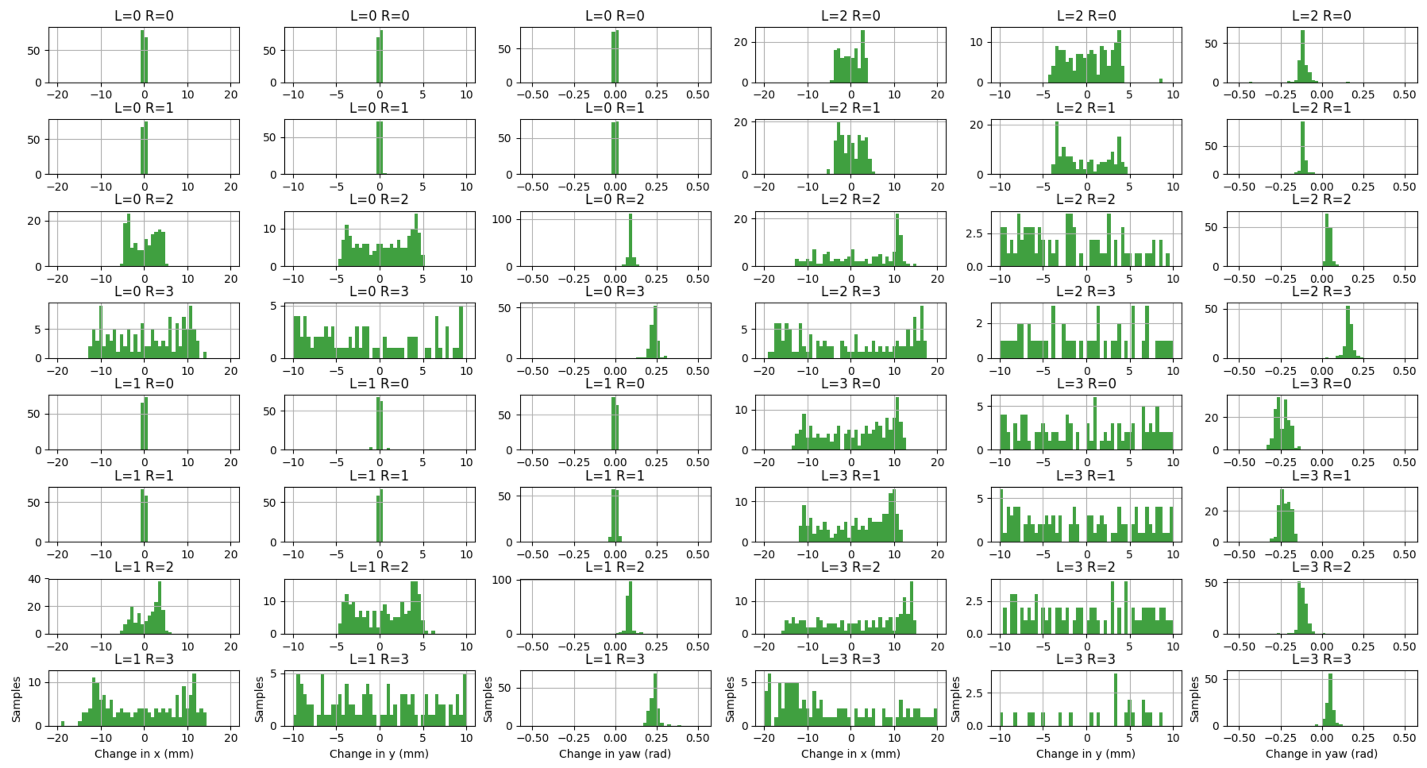


Figure 5.3: Distribution in the global frame for Δx, Δy and Δ yaw. The title L, R on top of each graph corresponds to the PWM value sent to the left and right motor times 10^4 (Eg:L$*10^4$)

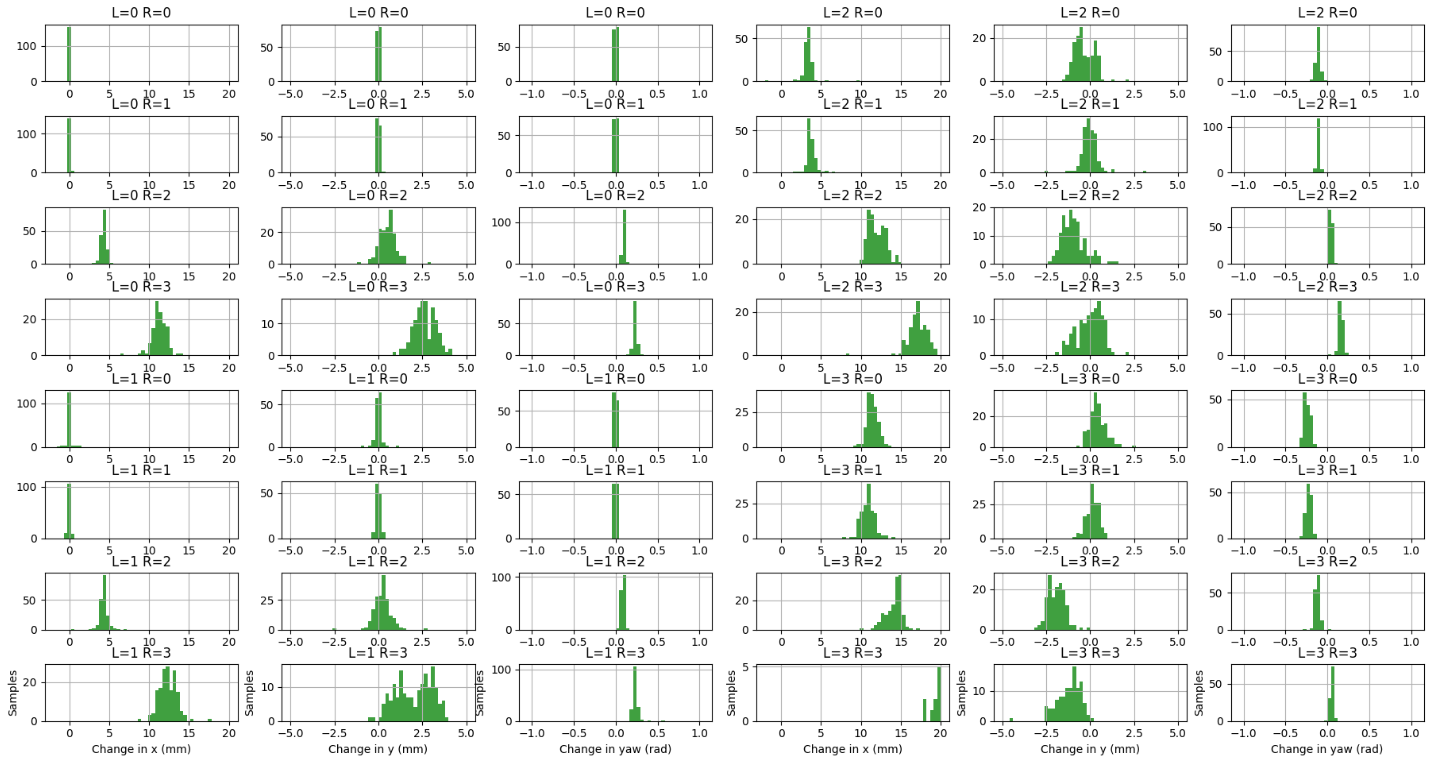


Figure 5.4: Distribution in the robot frame for Δx, Δy and Δ yaw.The title L, R on top of each graph corresponds to the PWM value sent to the left and right motor times 10^4 (Eg:L$*10^4$)

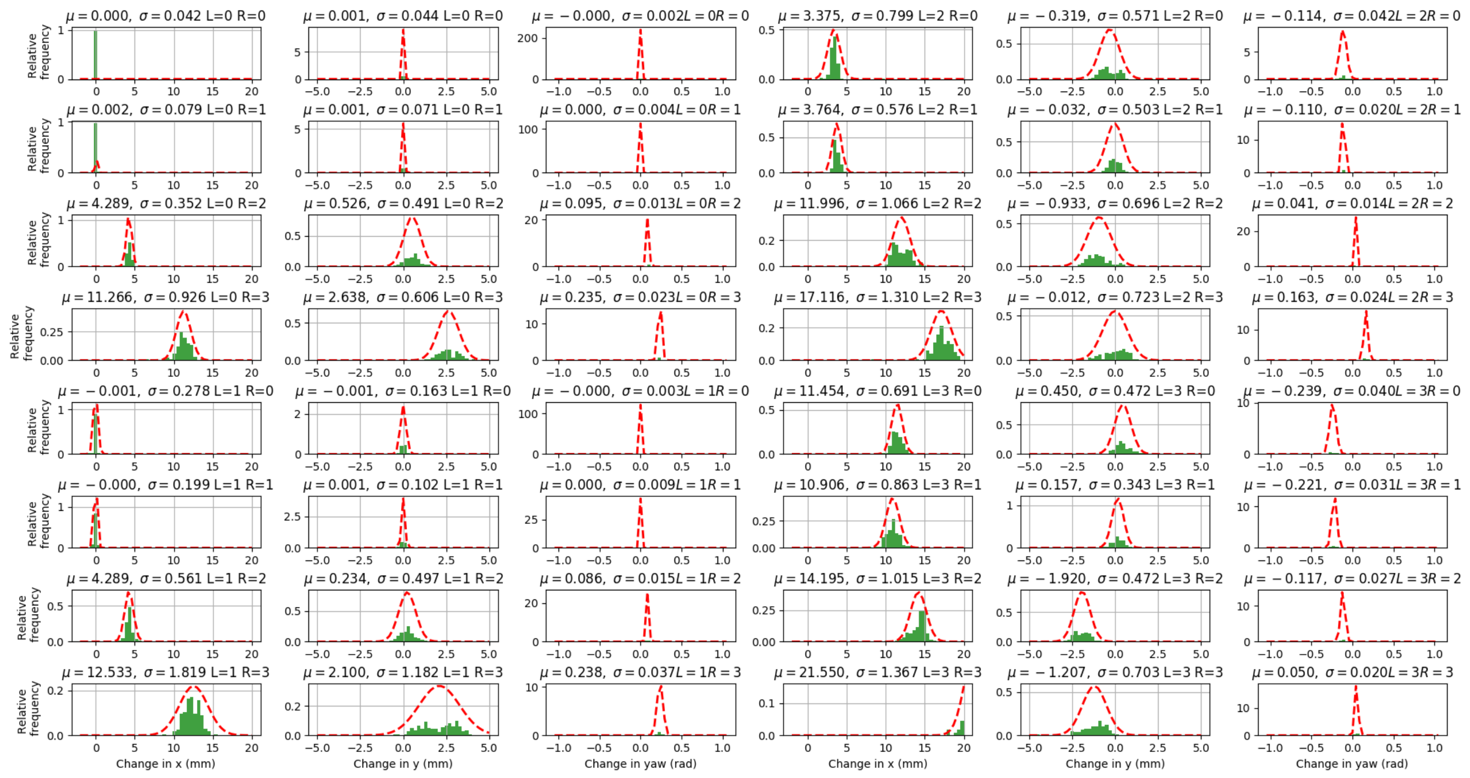


Figure 5.5: Fitting the distribution in the robot frame for Δx, Δy and Δ yaw with a Gaussian function.The title L, R on top of each graph corresponds to the PWM value sent to the left and right motor times 10^4 (Eg:L$*10^4$). The mean and standard deviation is also mentioned on the title of each graph.

Left Motor (PWM)	Right Motor (PWM)	Δx mean (mm)	std	Δy mean (mm)	std	Δyaw mean (rad)	std
0	0	0.000	0.042	0.001	0.044	-0.000	0.002
0	10000	0.002	0.079	0.001	0.071	0.000	0.004
0	20000	4.289	0.352	0.526	0.491	0.095	0.013
0	30000	11.266	0.926	2.638	0.606	0.235	0.023
10000	0	-0.001	0.278	-0.001	0.163	-0.000	0.003
10000	10000	-0.000	0.199	0.001	0.102	0.000	0.009
10000	20000	4.289	0.561	0.234	0.497	0.086	0.015
10000	30000	12.533	1.819	2.100	1.182	0.238	0.037
20000	0	3.375	0.799	-0.319	0.571	-0.114	0.042
20000	10000	3.764	0.576	-0.032	0.503	-0.110	0.020
20000	20000	11.996	1.066	-0.933	0.696	0.041	0.014
20000	30000	17.116	1.310	-0.012	0.723	0.163	0.024
30000	0	11.454	0.691	0.450	0.472	-0.239	0.040
30000	10000	10.906	0.863	0.157	0.343	-0.221	0.031
30000	20000	14.195	1.015	-1.920	0.472	-0.117	0.027
30000	30000	21.550	1.367	-1.207	0.703	0.050	0.020

Table 5.1: Gaussian models fit for discretized set of inputs.

in terms of accuracy or loss.

5.4 Performance

The trajectory following task had to be complex enough to show that the robot could tackle most trajectories. I chose to do the figure 8 as it has a straight path, a left turn and a right turn. The figure 8 trajectory which I made had turns with variable radius of curvature from very small to large. This makes the test practical and a good metric to test the performance of the robot.

I made the robot follow this trajectory and calculate the error in terms of distance from the desired trajectory. I also calculate the error in orientation from the orientation I

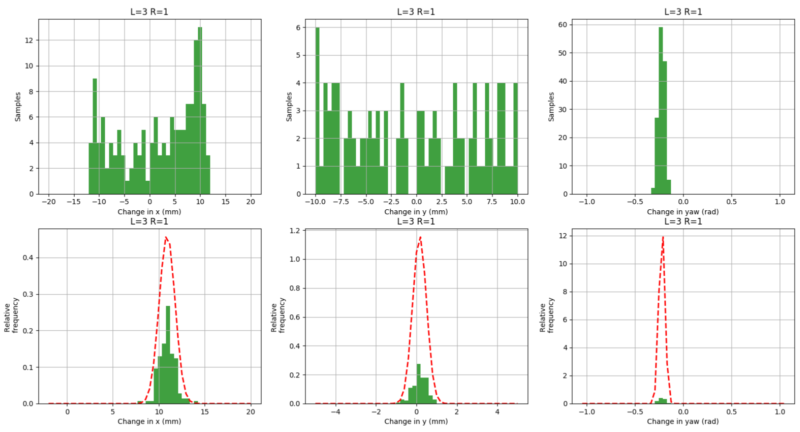
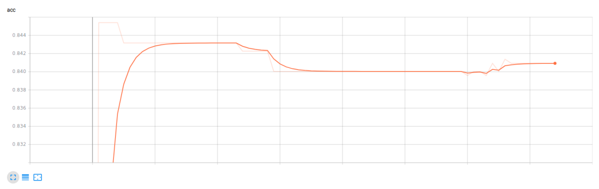


Figure 5.6: Gaussian probabilistic dynamic model of change in robot state in the robot frame



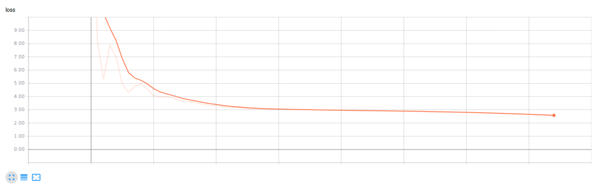


Figure 5.7: Model accuracy

Figure 5.8: Model loss function

desired in the trajectory.

I plotted the actual position of the robot trying to follow the trajectory using the differential drive algorithm (Fig 5.9), the Gaussian probabilistic model (Fig 5.10) and the model learned using a neural network (Fig 5.11).

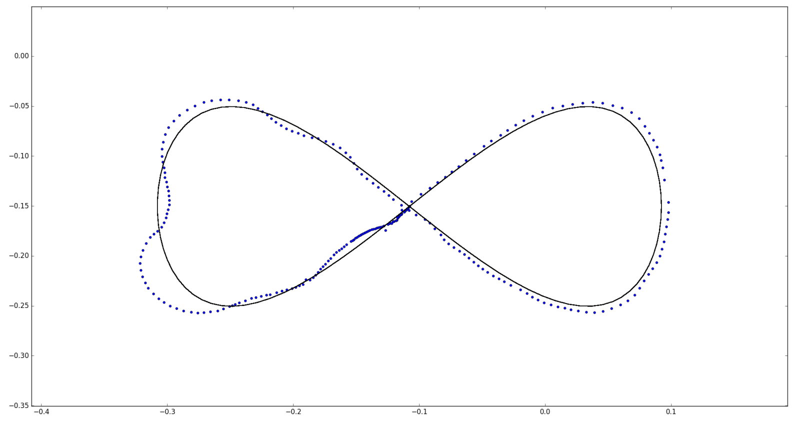


Figure 5.9: Trajectory following using differential drive algorithm (P controller).

Method	Trajectory	RMS Error Distance (mm)	RMS Error Orientation (rad)
Differential drive	Figure 8	6.67	0.63
MDP(Gaussian)	Figure 8	10.32	1.37
MDP(Neural Net)	Figure 8	7.94	1.05

Table 5.2: Error from the desired trajectory in terms of distance from desired trajectory.

I saw the performance by each algorithm and tabulated the root mean square error for each algorithm(Table 5.2). I saw that the differential drive algorithm performed the best after a lot of tuning. The differential drive algorithm has to be tuned again if we changed any small feature of the robot. This was seen when I had broken a leg and

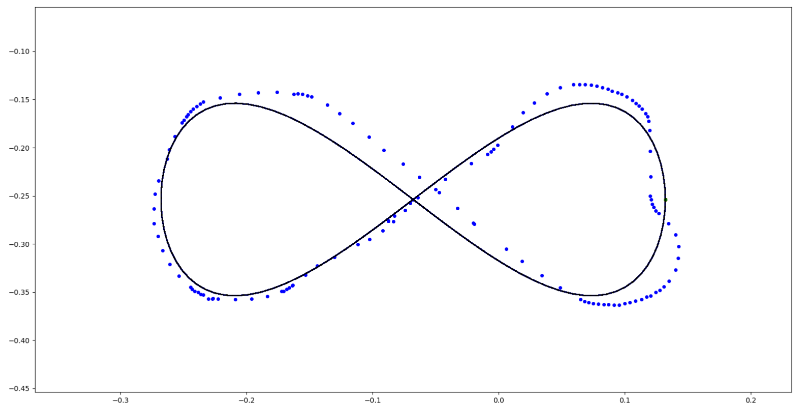


Figure 5.10: Trajectory following using mdp algorithm with Gaussian model.

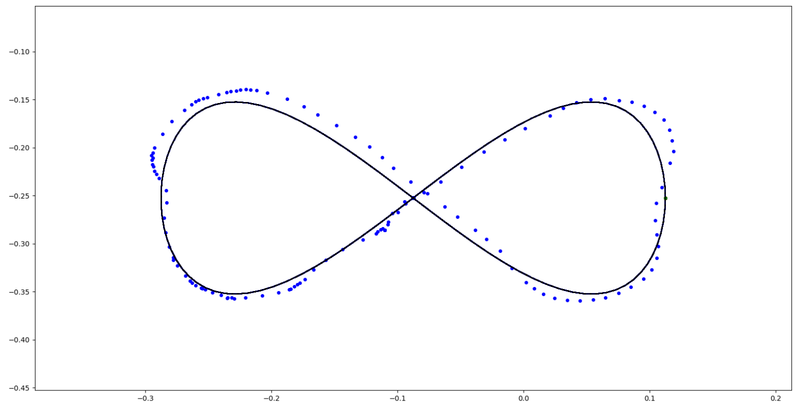


Figure 5.11: Trajectory following using mdp algorithm with Neural network model.

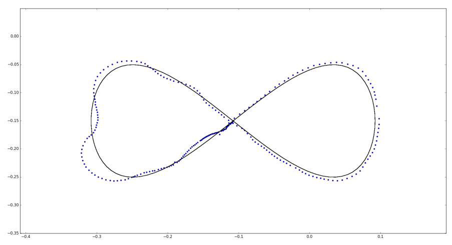


Figure 5.12: Tuned differential drive algorithm

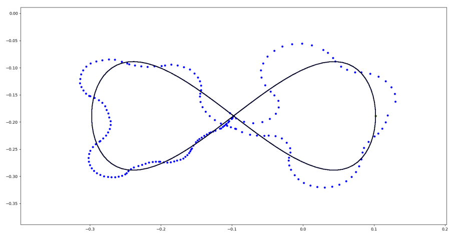


Figure 5.13: Same tuning parameters but small change in leg of the robot

replaced it with a new leg of the same size and shape. This also shows the sensitivity of vibration based locomotion to the robot. The figure 5.13 shows that on simply using previously tuned values on a similar robot the differential drive algorithm does not perform well. Although the neural network model didn't perform the best in the metric, it was the easiest to get the robot to follow the trajectory path.

Chapter 6

Conclusions

In this work I showed that I can control the movement of the robot with no knowledge of the dynamics of the system. I looked into developing a controller based on the model which I learned. My data efficient method to control the robot showed that I can follow a trajectory in an acceptable range and benchmarked it against a differential drive algorithm.

This result is acceptable as the main goal for this method is to control robots with more complex actuation strategies with motors placed in different positions and orientations. The belief is that my method is agnostic to the position and orientation of the actuators on the Prism bot that makes my method applicable to any morphology of the Prism bot.

The biggest drawback to my approach is that I still don't have a method to optimize the design and placement of the actuators. I am only able to learn how to control the robot irrespective of the orientation and position of the rotating masses. Thus I still have to build robots with motors mounted in various position to find the various interesting locomotion modalities.

Chapter 7

Future Work

The feet of the robot contribute considerably to the movement of the robot. I do not track the movement of the legs as a state of the robot. I have seen using slow motion video capture of the legs that the locomotion moves minimally on each cycle of the motor then lurches periodically. On sensing the shape of the leg I could more accurately model the movement of the robot and define the exact state of the robot.

The goal is to control the robot, it would be ideal to directly learn a controller by using some data efficient methods to learn controllers on a real robot such as PILCO[9]. PILCO is a method that uses probabilistic dynamic model and explicitly incorporate model uncertainty in the long-term plan. I can then compute policy gradients to improve the policy. There are certain smoothness assumptions which usually cannot be taken for robots which make contact to surfaces. This could be an issue and Deep PILCO[19] could be used to overcome this limitation.

The Optitrack system provides a reliable feedback of the exact position of the robot with sub millimeter precision. The downside is that the setup is quite expensive and cannot be used in all environments. I could use the on-board IMU to provide some state estimation especially since the robot is almost constantly parallel to the ground during its motion. Using a state estimator I could remove the requirement of the Optitrack and there by learn motion models without the Optitrack. This leads to the possibility of the robot updating its controller based on feedback of the motion of the robot using the state estimator.negative.

In this work, I use a robot with the rotating masses on either side of the robot. There are interesting motions which are seen by mounting the motors in different orientations.

After understanding the locomotion we could apply distributed algorithms[20] which use very limited amount of sensing information[2] from the environment and can be computed on the robot.

Chapter 8

Summary

This thesis could not have been completed without the support, advice and facilities provided by Bioinspired Robotics and Design Lab at UCSD. I would like to thank Prof Mike Tolley for providing me guidance through the course of the research and helping me complete my thesis.

I thank Prof Sonia Martinez for providing direction for the research keeping in mind the constraints of the Prism bot.

Prof Nikolay Atanasov's guidance in improving the experiments and results in modeling the problem was helpful and greatly appreciated.

The research would not be possible without a robot. William Weston-Dawkes provided the design and fabrication method to build the Prism bot. His knowledge and guidance on the research was crucial to building this robust design. I thank him for all his help, guidance and support.

The UCSD Coordinated Robotics Lab were immensely helpful in sharing their

resources and providing access to the Optitrack system. I would like to thank Prof Thomas Bewley for giving permission to use this resource.

I would like to thank the Gravish Lab and Prof Nick Gravish for helping me take high speed video of the Prism bot moving.

Appendix A

Code Repositories

1. Crazyflie firmware:

   https://github.com/UCSDBioinspired/crazyflie-firmware

2. Crazyflie ROS drivers:

   https://github.com/whoenig/crazyflie_ros.git

3. Prism Bot ROS controllers:

   https://github.com/UCSDBioinspired/prismbot_cmd

4. Data processing and model training:

   https://github.com/UCSDBioinspired/scripts

5. Optitrack streaming:

   https://github.com/UCSDBioinspired/optitrack

Bibliography

[1] W. P. Weston-Dawkes, A. C. Ong, M. Ramzi, A. Majit, F. Joseph, and M. T. Tolley, "Towards Rapid Mechanical Customization of cm-scale Self-Folding Agents," *2017 IEEE International Conference on Intelligent Robots and Systems (IROS)*, pp. 4312–4318, 2017.

[2] V. Krishnan and S. Martínez, "Spatial self-organization in multi-agent swarms via distributed computation of diffeomorphisms," *SIAM Journal on Control and Optimization*, 2016. Under review.

[3] S. Felton, M. Tolley, E. Demaine, D. Rus, and R. Wood, "A method for building self-folding machines," *Science*, vol. 345, no. 6197, pp. 644–646, 2014.

[4] F. Becker, S. Boerner, V. Lysenko, I. Zeidis, and K. Zimmermann, "On the mechanics of bristle-bots - modeling, simulation and experiments," in *ISR/Robotik 2014; 41st International Symposium on Robotics*, pp. 1–6, June 2014.

[5] P. Vartholomeos and E. Papadopoulos, "Analysis, design and control of a planar micro-robot driven by two centripetal-force actuators," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pp. 649–654, IEEE, 2006.

[6] A. Nagabandi, G. Yang, T. Asmar, G. Kahn, S. Levine, and R. S. Fearing, "Neural network dynamics models for control of under-actuated legged millirobots," *arXiv preprint arXiv:1711.05253*, 2017.

[7] R. Lioutikov, A. Paraschos, J. Peters, and G. Neumann, "Sample-based information-theoretic stochastic optimal control," in *Proceedings of 2014 IEEE International Conference on Robotics and Automation*, pp. 3896–3902, IEEE, 2014.

[8] M. C. Yip and D. B. Camarillo, "Model-less feedback control of continuum manipulators in constrained environments," *IEEE Transactions on Robotics*, vol. 30, pp. 880–889, Aug 2014.

[9] M. P. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, (USA), pp. 465–472, Omnipress, 2011.

[10] R. Tedrake, T. W. Zhang, and H. S. Seung, "Learning to walk in 20 minutes," in *Proceedings of the Fourteenth Yale Workshop on Adaptive and Learning Systems*, vol. 95585, pp. 1939–1412, Beijing, 2005.

[11] S. Chernova and M. Veloso, "An evolutionary approach to gait learning for four-legged robots," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, pp. 2562–2567 vol.3, Sept 2004.

[12] D. Lizotte, T. Wang, M. Bowling, and D. Schuurmans, "Automatic gait optimization with gaussian process regression," in *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, (San Francisco, CA, USA), pp. 944–949, Morgan Kaufmann Publishers Inc., 2007.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[14] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning.," *CoRR*, vol. abs/1509.02971, 2015.

[15] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on Machine Learning* (F. Bach and D. Blei, eds.), vol. 37 of *Proceedings of Machine Learning Research*, (Lille, France), pp. 1889–1897, PMLR, 07–09 Jul 2015.

[16] Y. Duan, M. Andrychowicz, B. Stadie, O. J. Ho, J. Schneider, I. Sutskever, P. Abbeel, and W. Zaremba, "One-shot imitation learning," in *Advances in neural information processing systems*, pp. 1087–1098, 2017.

[17] R. Bellman, "A markovian decision process," *Indiana Univ. Math. J.*, vol. 6, pp. 679–684, 1957.

[18] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden,

M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[19] Y. Gal, R. McAllister, and C. E. Rasmussen, "Improving pilco with bayesian neural network dynamics models," in *Data-Efficient Machine Learning workshop, ICML*, 2016.

[20] N. Atanasov, J. L. Ny, and G. Pappas, "Distributed algorithms for stochastic source seeking with mobile robot networks," *ASME Journal of Dynamic Systems, Measurement, and Control (JDSMC)*, vol. 137, no. 3, pp. 031011–031011–9, 2015.