DEVELOPING AN ANTICIPATORY CONTROLLER TO IMPROVE PERFORMANCE OF A SNAKE-LIKE ROBOT IN UNSTRUCTURED TERRAIN

A Thesis
Presented to
The Academic Faculty

By

Ian Tomkinson

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Mechanical Engineering

Georgia Institute of Technology

December 2018

DEVELOPING AN ANTICIPATORY CONTROLLER TO IMPROVE PERFORMANCE OF A SNAKE-LIKE ROBOT IN UNSTRUCTURED TERRAIN

Approved by:

Dr. Daniel I Goldman, Advisor
School of Physics
Georgia Institute of Technology

Dr. Frank Hammond
School of Mechanical Engineering
Georgia Institute of Technology

Dr. David Hu
School of Mechanical Engineering
Georgia Institute of Technology

Date Approved: December 7, 2018

Belief can mean the difference between a fear of failure and the courage to try. On a team or in a family, belief makes each individual stronger and also fortifies the group as a whole.

Coach K

To my family

ACKNOWLEDGEMENTS

I want to thank the members of the Complex Rheology and Biomechanics Lab at Georgia Tech and the Biorobotics Lab at Carnegie Mellon for making this work possible. On the CMU side, thanks to Dr. Howie Choset for collaborating with our group, and a special thanks to Guillaume Sartoretti for assisting with the development of the control scheme. I'd like to thank Dr. Dan Goldman for advising and supporting me during my studies, as well as Dr. Hammond and Dr. Hu for serving on the committee. Thanks to Dr. Jennifer Rieser for extensive mentorship and guidance with designing the experiments, assisting to analyze the Optitrack data, and offering insights into the results through her work in simulation. I'd also like to think Yasemin and Enes Aydin for their help with technical challenges developing the gantry system, as well as Zachary Goddard for his extensive contributions to the gantry development. Thanks to Ross, Xiaobin, and Zachary for assisting me learn the ropes during my first summer in the lab. Finally, a very special thanks to Jonathan Gosyne, Kehinde Aina, Seth Daly, Mason Murray-Cooper, Daniel Soto, and Christian Hubicki for making the CRAB lab an exciting and fun place to work through engaging discussions about everything ranging from advanced control theory to Survivor strategy.

TABLE OF CONTENTS

LIST OF FIGURES

SUMMARY

Limbless robots have the potential to help with many possible applications from search and rescue to surveillance. However, their performance in unstructured environments does not currently match that of living systems. In order to understand how collisions with obstacles effect locomotion, we studied the interactions between a snake robot and vertical posts. Under normal open loop control where the robot is controlled by the serpenoid equation presented in [1], we observe that the collision between the robot and the post will often cause the robot to reorient and deviate away from its open loop trajectory by an angle θ which we call the scattering angle. Drawing insights from previous experimental results and simulations [2], we hypothesize that a model of the interaction between the robot and the posts can be used to implement a simple control scheme to control the orientation of the robot using only minimal onboard sensing. These robot-obstacle collisions are characterized by running many experiments to systematically sample all possible contact conditions between the robot and the post. To assist with the collection of this data, an automated gantry system was developed to conduct experiments without any human input. This allows us to model and understand the behavior of the robot at contact. Using this model, we develop an anticipatory control scheme to correct for the scattering that results from the collision with the posts. Contact sensing at the head of the snake measures the location and duration of contact with the pegs. The controller uses this measurement to predict the magnitude and direction of the steering behavior and steer the snake to correct for the scattering. Finally, we experimentally validate this controller for a single post as well as a row of five evenly spaced posts and find that the controller reduces the distribution of scattering angles caused by the post and offers further insight into the nature of the robot-obstacle collision.

CHAPTER 1
INTRODUCTION

1.1 Robots in Unstructured Environments

Autonomous navigation through unstructured environments is quickly becoming one of the most exciting new areas in the field of robotics. While robots are traditionally used in industrial applications such as assembly line automation where precise control of position is necessary, this new relatively new field of robotics seeks to take robots out of the laboratory and in to the real world. From space exploration, field robotics, autonomous driving, drones, and bipedal robots, roboticists are quickly changing the paradigm of what kinds of problems robots can solve. These novel systems have the potential to completely revolutionize many different industries including transportation, agriculture, search and rescue, military surveillance, and package delivery. While this may appear to be a wide range of seemingly unrelated fields, the fundamental questions behind solving these problems are the same:

1. How does a robot pick the ”best” path through the environment?
2. What does it mean for a path to be ”best”?
3. How can a robot adapt its path as it learns more about our environment?
4. How do we control the robot to move along this path?

Modern robotics and controls attempts to address these challenging issues through relatively new approaches like optimal control, robust control, artificial intelligence, and machine learning. However, elegant solutions to these difficult problems already exists in biological systems. In particular, animals like snakes, lizards, and cockroaches are particularly adept at maneuvering through a wide range of environments. Therefore, our approach to addressing some of these challenging questions is to study biological and robotic systems in parallel.

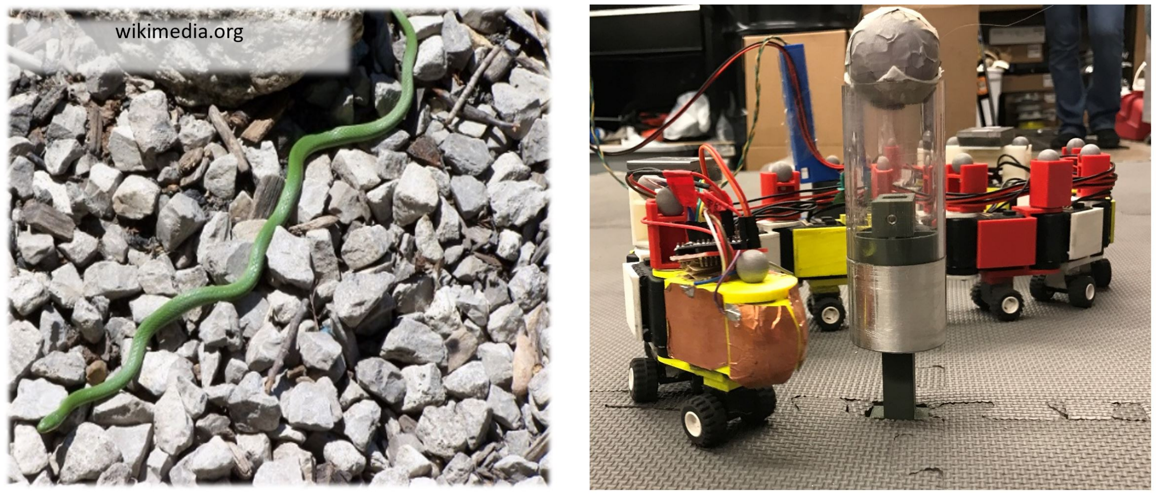


Figure 1.1: Comparison of a biological snake and our robophysical model. The lefthand figure shows a living snake moving through an unstructured environment, and the right-hand figure shows our attempt to capture aspects of this behavior in a controlled laboratory environment.

1.2 Drawing Inspiration from Biology

While animals are able to easily navigate through environments with rocks, sticks, and other obstacles, robotic systems are unable to match their performance. Fig. 1.1 illustrates the type of heterogeneous environment robots may encounter in the field. The performance of living snakes in these types of environments has been studied in [3]. However, with live animal experiments, we have very little control over the behavior of the animal, we can only observe what they chose to do. Therefore, studying living systems and robotic systems in parallel allows us to systematically vary robot parameters and observe the effect of varying these parameters on the robot's behavior. We hope that studying the biological system will offer us insights into how to design and program a better robot, and studying the robotic system will offer us insight into how living systems can achieve such fluid locomotion.

1.3 Robophysics

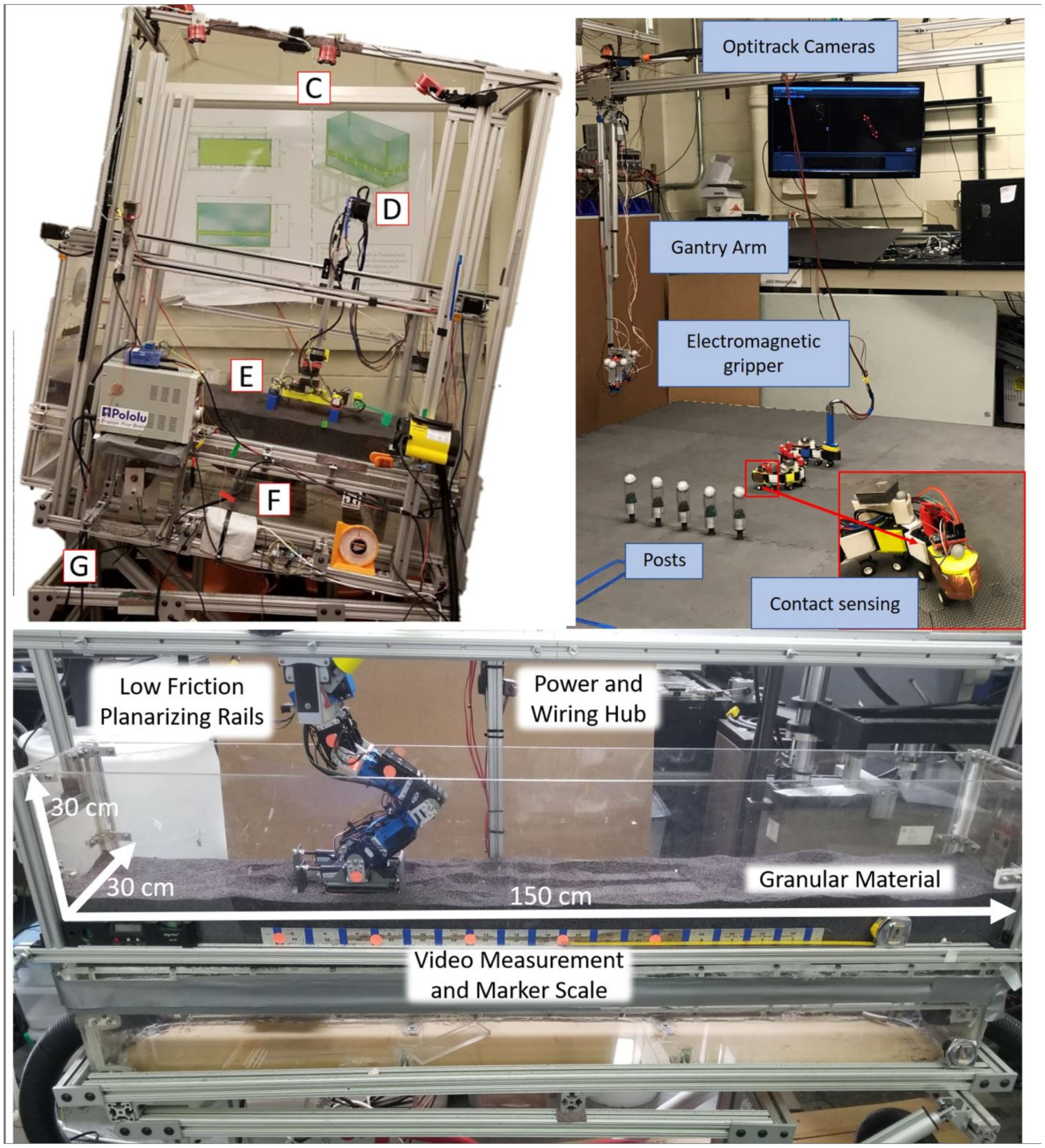


Figure 1.2: Examples of automated systems in the CRAB lab which use Robophysics to model interactions between a range of different robots and environments. Images courtesy of the CRAB lab.

This idea of systematically changing robot parameters to see how they effect the robot's behavior is crucial to what we call Robophysics [4]. In particular, we are interested in using

Robophysics to study the interaction between the robots and the environment. Instead of designing a controller that treats obstacles as a disturbance to the system and rejects them, or using simultaneous localization and mapping (SLAM) to detect and avoid obstacles, we seek to develop a model for the interaction between the robot and its environment through a combination of systematic experimental testing and simulation. A model of these interactions can then be utilized by a simple controller to use these interactions between the robot and the environment advantageously. The main drawback to this approach, however, is that these models often require large amounts of experimental data to develop and validate. In order to assist with data collection, automated gantry systems can be extremely useful. Fig. 1.2 shows three automated gantry systems developed to collect large amounts of experimental data and understand how robotic systems respond to different environments. Fig. 1.2-A shows a system designed to test robots climbing up slope on granular media. Fig. 1.2-B shows the gantry system designed to automate experiments modeling the interactions between a snake robot and a row of vertical posts. The development of this system is topic of the first part of this thesis. Finally, Fig. 1.2-C shows a system to study bipedal walking on granular materials [5].

1.4 Objective

The following two objectives are addressed in the two main sections of this thesis:

1. Develop an automated gantry system and perform systematic experimental testing to generate a model for the interactions between a snake robot and vertical posts.

2. Use this model of the environment to create an anticipatory controller to control the orientation of the robot based on the interaction of the robot and the obstacle.

The remainder of the thesis is organized as follows: Chapters 2-5 address the mechanical design, electronics, and software used to create the gantry system. Chapters 6-10 describe the snake robot, controller development, controller results, and modifications to the controller to improve performance.

Part I

Automated Gantry System

CHAPTER 2
GANTRY SYSTEM OVERVIEW AND MECHANICAL DESIGN

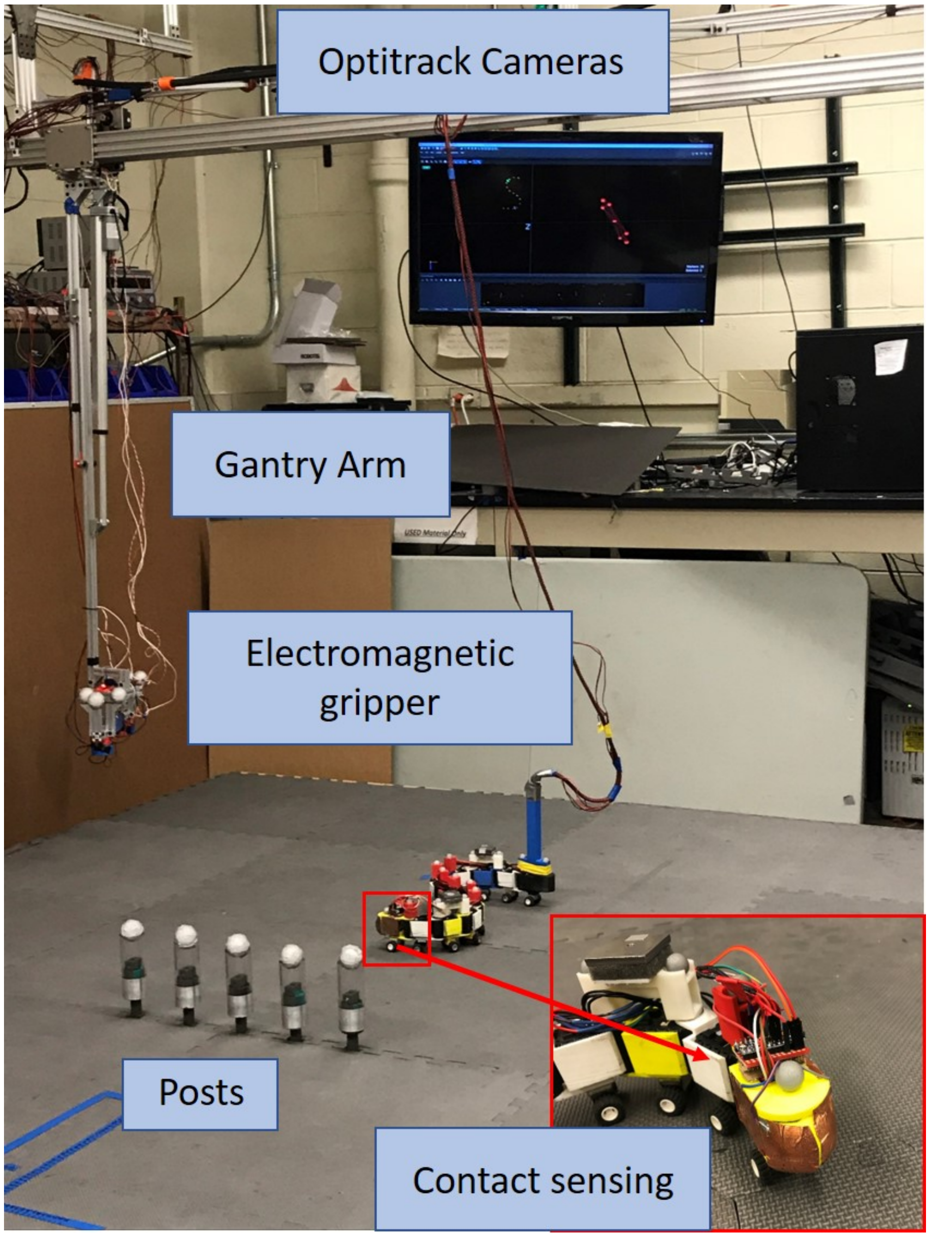


Figure 2.1: Overview of the gantry system. The key components (cameras, gantry arm, electromagnetic gripper, posts, and contact sensing) are highlighted.

2.1 Gantry Motivation

Previously, experiments to study the interactions between the snake robot and the posts relied on the snake robot being manually placed on the ground by a graduate student using a ruler. Due to the large amount of experimental data required to create an experimental model of these interactions, developing an automated system to conduct these experiments is a very logical solution. Building on the work of [6], where a smaller gantry system was built to study locomotion on sandy slopes, we seek to demonstrate that these gantry systems can also function on a much larger scale. However, we will demonstrate in this thesis that many novel issues arise when attempting to build an automated system at a much larger system. For example, two recurring challenges in the mechanical, electrical, and software development of the gantry system are reliably identifying Optitrack markers on the snake and making consistent, quality contact between the the snake robot and the gantry system. The first part of the thesis highlights and addresses these challenges and their solutions. Despite these challenges, we created a fairly robust automated gantry system that was able to conduct thousands of snake robot experiments. Furthermore, the gantry system offers higher accuracy in placing the robot with the desired position and orientation. This position and orientation accuracy can be now be tested in a systematic way so that we can develop an understanding of the uncertainty associated with these parameters. The remainder of Chapter 2 provides an overview of the tasks the gantry system will automate, important design criteria for the system, and a high level mechanical overview of the system. Chapter 3 provides insight into the software we wrote to automate these tasks. Chapter 4 describes the electronics necessary for the sensing and actuation required by the gantry system. Finally, Chapter 5 describes the performance of the system. To see the system in action, please refer to the video referenced in this section.

2.2 Gantry Overview

The gantry allows for the control of X, Y, Z and orientation of the electromagnetic gripper so that the snake can be picked up from any position on the mat. The X and Z directions are controlled by two stepper motors, and the Y axis uses a Frigelli motor to raise and lower the electromagnetic gripper. Two electromagnets allow the gantry to pick up the snake robot by making contact with magnetic plates on the robot. Finally, additional features are developed as needed to improve the safety and robustness of the system.

2.3 Experimental Overview

Our primary purpose in developing this system was to automate the experiments presented in [2]. In this work, the snake was placed manually on the ground which was time consuming and potentially inaccurate. Like the system presented in [6], our automated gantry system can conduct many experiments with minimal human supervision. This allows for the large amounts of data to be collected relatively easily, and demonstrates our ability to produce results consistent with [2]. In order to automate these experiments, the following steps are implemented.

1. Initialize the snake
2. Begin snake locomotion
3. Move the gantry arm in parallel with the snake
4. Stop the gantry and snake after five complete undulations
5. Determine the position and orientation of the snake robot
6. Position the gantry directly above the snake's magnetic contact plates
7. Turn on the electromagnets

8. Lower the gantry arm until contact is made with the plates
9. Raise the snake until it is clear of the posts
10. Reorient the snake and move it to the next desired head initial position
11. Lower the snake onto the mat
12. Turn off magnets, raise the gantry arm, and move the gantry off to the side of the mat

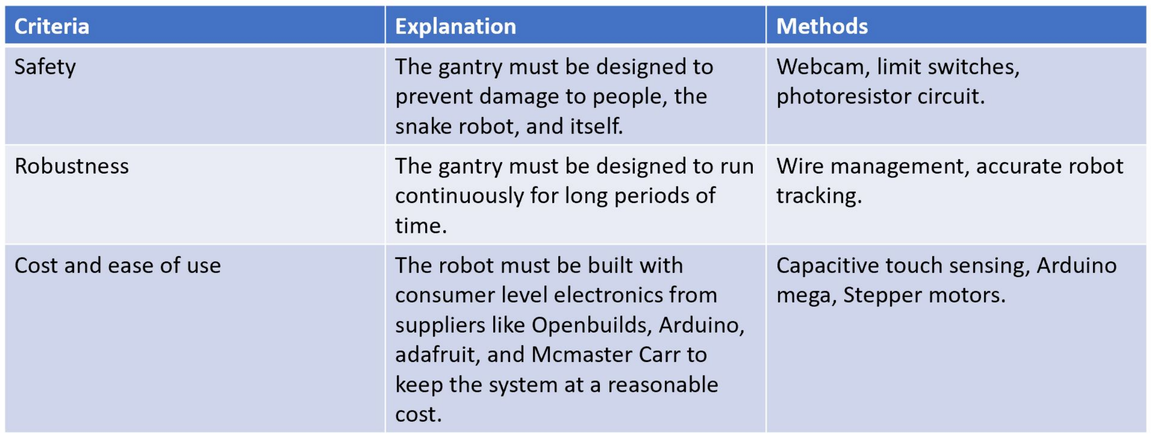
2.4 Objectives

Based on these steps, we determined the following objectives as crucial to being able to successfully automate the experiments.

1. The gantry will need to be able to pick up the snake in a variety of different positions and orientations.
2. The gantry cameras will need to have a complete view of the entire mat in order to capture the Optitrack markers for the snake.
3. The gantry will need to be able to accurately place the snake at a variety of initial positions and orientations.
4. The gantry will need to be built with effective wire management so that the snake does not become stuck during experiments.

2.5 Design Parameters

Based on the steps and objectives described above, we determined that safety, robustness, cost, and ease of use are important design criteria for the system to meet. The following sections describe in detail why they were selected as important, and how the design sought to address the criteria. These criteria are summarized in Fig. 2.2 below.



Criteria	Explanation	Methods
Safety	The gantry must be designed to prevent damage to people, the snake robot, and itself.	Webcam, limit switches, photoresistor circuit.
Robustness	The gantry must be designed to run continuously for long periods of time.	Wire management, accurate robot tracking.
Cost and ease of use	The robot must be built with consumer level electronics from suppliers like Openbuilds, Arduino, adafruit, and Mcmaster Carr to keep the system at a reasonable cost.	Capacitive touch sensing, Arduino mega, Stepper motors.

Figure 2.2: Summary of the key design criteria for the gantry system, and how they are taken into account by the design.

2.5.1 Safety

We designed the gantry system to be operated with no human intervention in order to collect large amounts of experimental data. As a result, many precautionary measures were taken in order to prevent damage to people, the snake robot, or the gantry system. These safety features are detailed below:

1. We installed a Logitech C920 HD Pro Webcam (Fig. 2.3) above the setup, giving a complete view of the snake robot and gantry system at all times during experiments. Using OBS studio, we can easily stream a live feed from the experiment to Twitch TV so that the system can be monitored remotely while it is running. Additionally, OBS studio also gives us the option to record video, which can be used to go back and detect what caused an experiment to fail.

Figure 2.3: We mounted this 1080p HD Logitech webcam above the gantry system in order to stream experiments to Twitch, as well as record HD video of important experiments.

2. The computer that controls the gantry system is equipped with TeamViewer, which means that we can log in remotely and end an experiment if needed, as well as control the gantry arm, monitor experiment progress in the command window, and begin recording video with OBS studio.

3. We installed limit switches (Fig. 2.4) in the x, y, and z directions to prevent the gantry from grinding against the end of the rails, or pushing the snake into the ground.

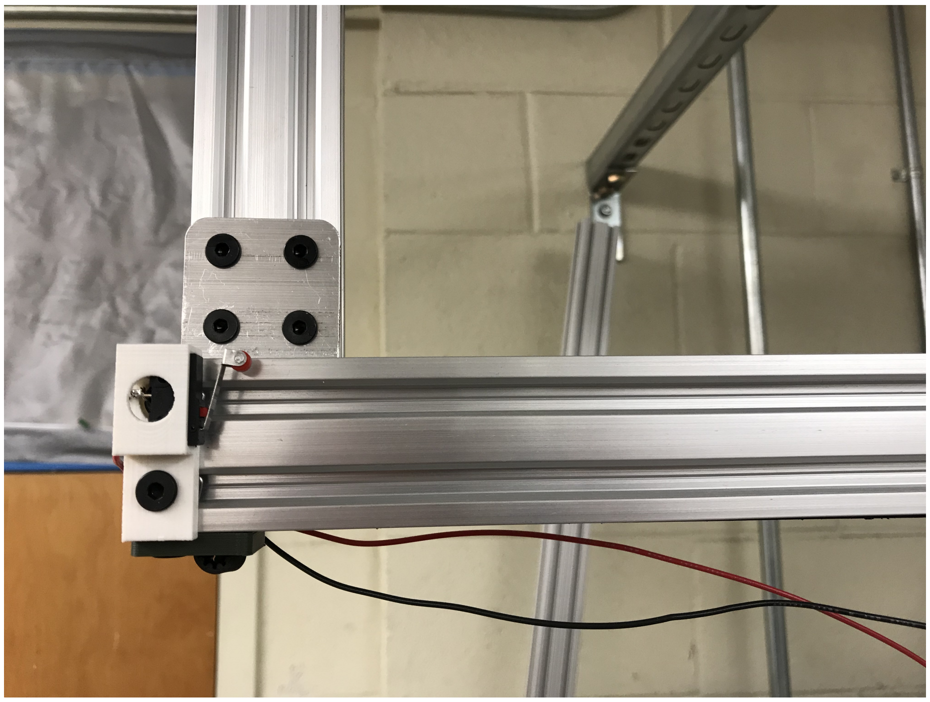


Figure 2.4: Four limit switches are mounted at the ends of the X and Z support rails. These switches turn off the X and Z actuation once the gantry has reached the edge of its desired range of motion by triggering a GPIO pin on the Arduino Mega. This prevents damage to the Stepper motors which control the gantry X and Z position.

4. We built and installed two circuits with photoresistors around the electromagnets on the magnetic gripper of the gantry. These circuits allow the user to estimate when contact has been made based on the amount of light detected by the sensors. The gantry code checks to see if successful contact was made before raising the gantry arm, and if no contact is detected, the program ends. The hardware and software for these systems are described in greater detail in Chapters 3 and 4.

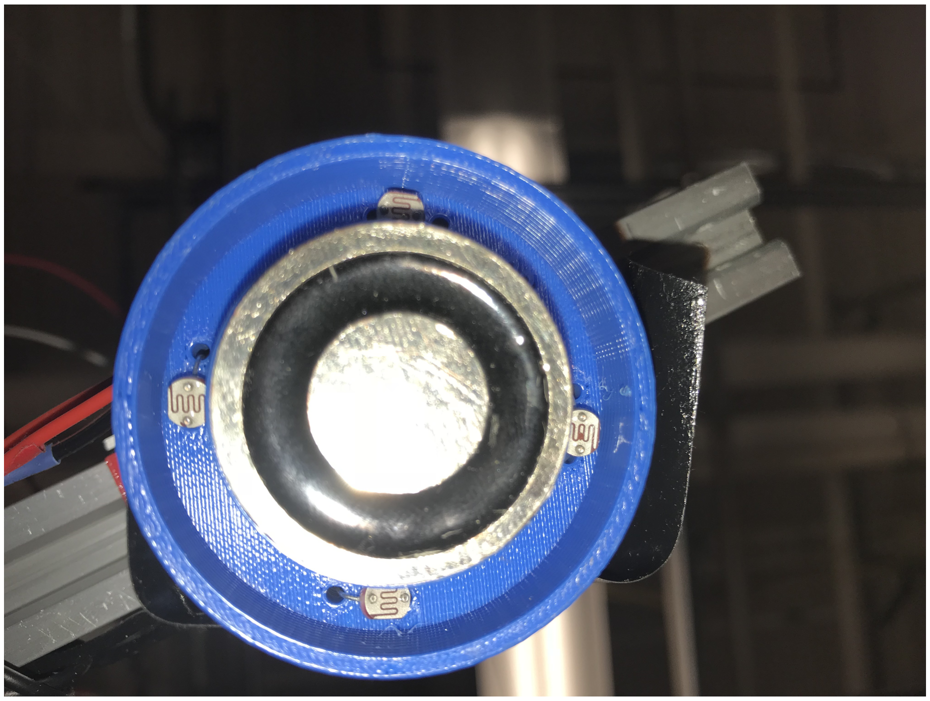


Figure 2.5: This simple photoresistor circuit determines the proximity of the electromagnet to the magnetic plate on the snake. A threshold value is determined for what constitutes successful contact, allowing the program to exit in the case of poor contact with the snake.

5. At the beginning of each experiment, the code pings all of the motors to ensure they are still connected. If the computer cannot connect to a motor because an error or warning light has been triggered, the program exits.

6. The power switches(Fig. 2.6) for all of the components of the gantry system are easily accessible and clearly labeled so that components can be easily powered down in emergencies.

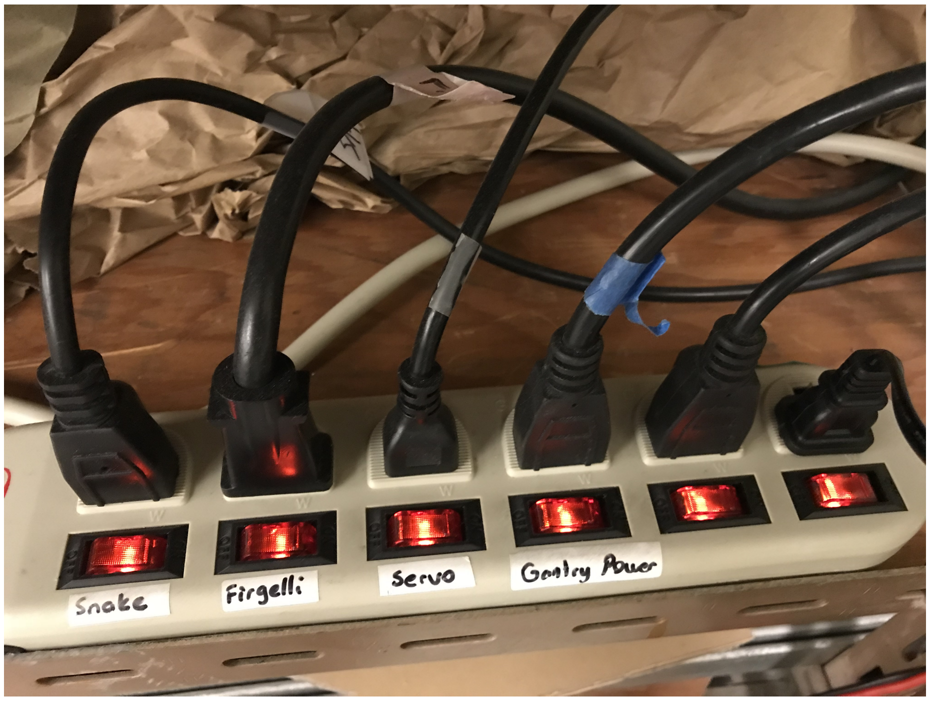


Figure 2.6: This power rail allows for easy access to the power supplies of all of the gantry components in case of emergencies.

2.5.2 Robustness

In order for the gantry system to complete many experiments without failure, it is important for the system to be robust. The two most significant challenges to the robustness of the system were mitigated by developing effective wire management and producing reliable Optitrack data.

1. Wire management:

   Wire management for a system of this size is crucial since there are many mechatronic systems that need to be powered, and numerous moving parts. In order to prevent gantry cables from dragging on the ground, we installed two drag chain cable carriers along the X and Z rails of the gantry well above the snake robot and the gantry arm. These carriers protect the cables from getting caught on the upper rails of the gantry while the system is moving, and keep wires out of the path of the snake. Another concern was the wires leading to the snake getting caught on the posts or getting caught on the gantry arm. These cables were run along the upper gantry rails then hung down directly over the snake. The gantry arm moves in parallel with the snake during experiments so that the snake cable is always hanging directly above the snake. The individual cables are bundled together using braided cable to prevent catching on the gantry.

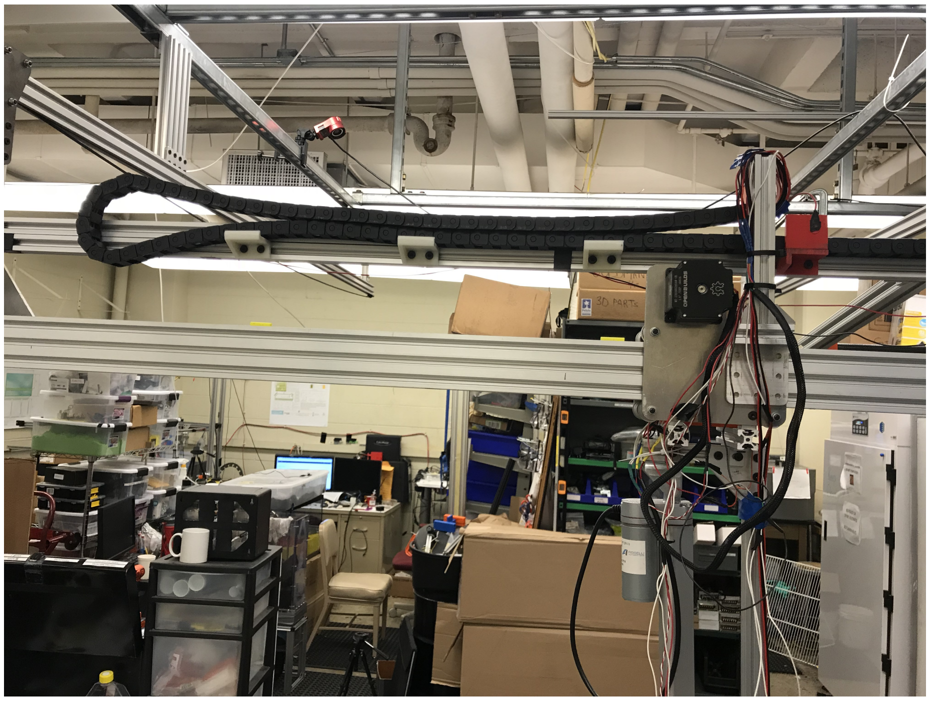


Figure 2.7: Cable carrying drag chains were installed on the X and Z rails of the gantry in in order to protect the wires from moving parts. 3D printed brackets support the chain as it moves in the positive and negative direction, allowing the chain to fold back on itself as the gantry moves.

Even with these precautions in place, it became apparent that power cables for the snake robot were applying forces on the snake robot and influencing the experimental data. As a result, a 3D printed tail module (Fig. 7.3) was designed so that the cables point directly up from the snake, instead of dragging behind the robot. Furthermore, instead of anchoring the snake cable to a fixed point on the gantry rail above, a simple system was built to allow the cable to slide freely in the Z direction to prevent pulling on the robot. A carbon fiber rod is mounted behind the gantry railing, and the cables are attached to a metal ring surrounding the rod. If the power cables become tensioned as the robot moves, instead of exerting a torque on the robot from above, the ring is free to slide along the rod so that the cable stays above the robot. While there is certainly more work that could be done in cleaning up the wires for aesthetics, these simple changes were sufficient to be able to collect hundreds of experiments of reliable data without any wire related issues.

2. Optitrack Data:

   The other challenging problem that needed to be overcome for a robust system was getting reliable Optitrack data for locating the gantry and snake during experiments and analyzing the experiments afterwards. Even with 6 Optitrack cameras, markers were often dropped due to the position of the gantry arm. In order to improve this data, the cameras were rearranged so that they were below the gantry arm, but there were some areas of the mat (like directly in front of the mat) where the Optitrack data was weak. This is because the focus of the camera placement was to have reliable locations for the electromagnet plates so that the gantry can effectively pick up the snake. Additionally, the markers on the snake were modified in order to be more visible to the Optitrack system. Marker holders were designed to raise the spherical markers above the surface of the snake and wires leading to the head of the robot. The marker holders on the magnet plates were custom printed to allow for the creation of rigid bodies. Early on, it proved almost impossible to calculate the center of mass of the snake from the raw marker data since markers were often incorrectly identified. It was much easier for Motive to look for two unique rigid bodies, and locate them in order to find the snake. Each magnet holder is composed of the 3D printed base, two spherical Optitrack markers, a layer of foam, then the metal plates which are picked up by the electromagnets. Reflective tape is used to create two unique rigid bodies, which are defined in the motive software. Similarly, a rigid body is created on the gantry end effector in order to track its location. Six large Optitrack markers are arranged symmetrically around the center of mass of the end effector so that moving to the location of the snake is a simple as commanding the gantry to move to the average of the X and Z coordinates of the snake rigid bodies.

2.5.3 Ease of Use and Cost

Additionally, the gantry was designed so that it could be assembled quickly with cost effective components. Besides the Optitrack camera system, (which is necessary for accurate tracking), components were selected that allowed the system to be developed quickly with relatively low cost parts. Many of the design decisions were made to emulate the system developed in [6] so that we could take advantage of the knowledge and expertise already in the lab about these systems. A majority of these components are readily available from McMaster Carr or OpenBuilds. For example, the V-Slot rails and Tee Nuts from Open Builds allow the upper gantry frame to easily be constructed with minimal machining. There is significant tips and advice for developing these kinds of automated systems available on openbuilds.com. In addition to all of the hobbiest level supplies, this website also has a strong community dedicated to creating do it yourself laser cutters, CNCs, and automated drawing machines that rely on very similar parts, frames, and actuation methods that we used to develop our system.

CHAPTER 3
GANTRY SOFTWARE OVERVIEW

3.1 Software Overview

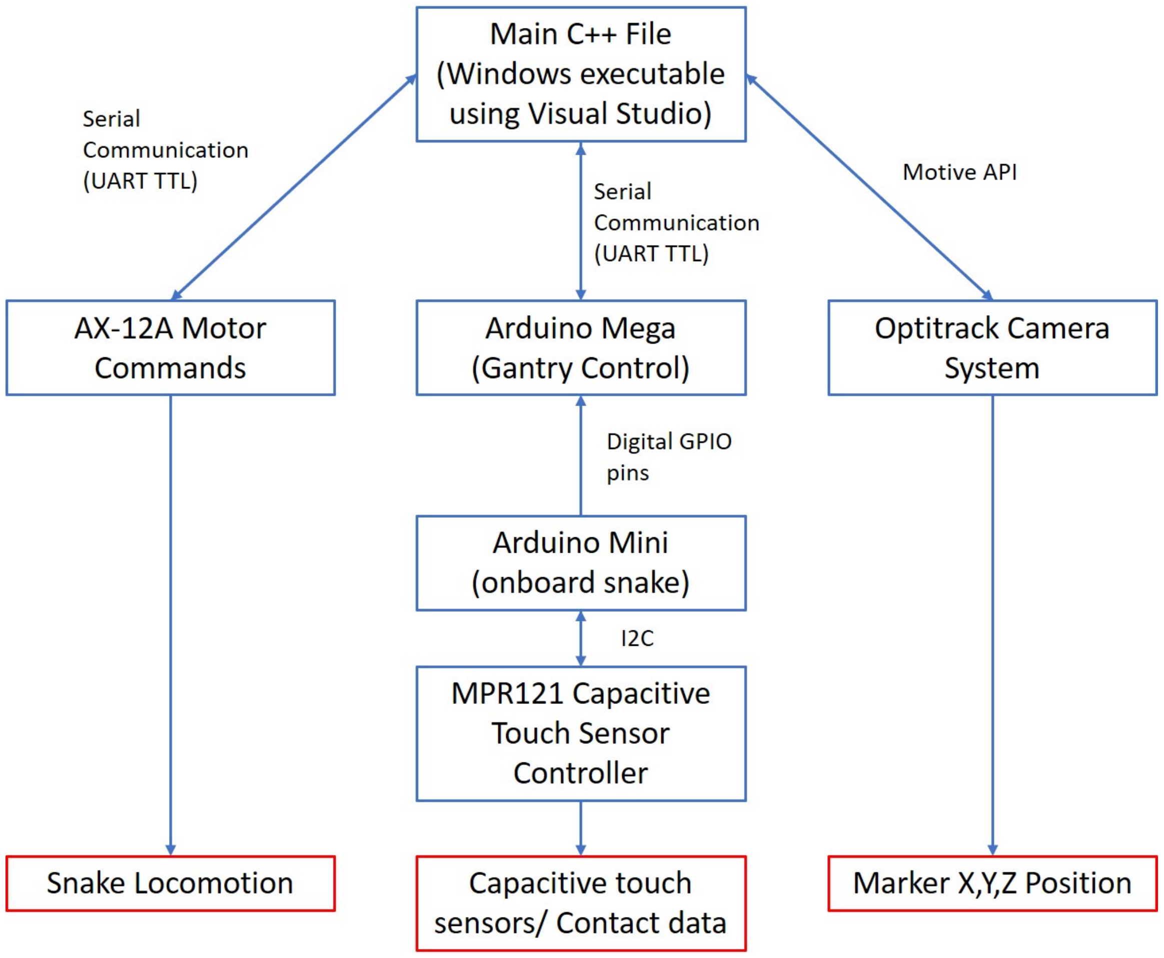
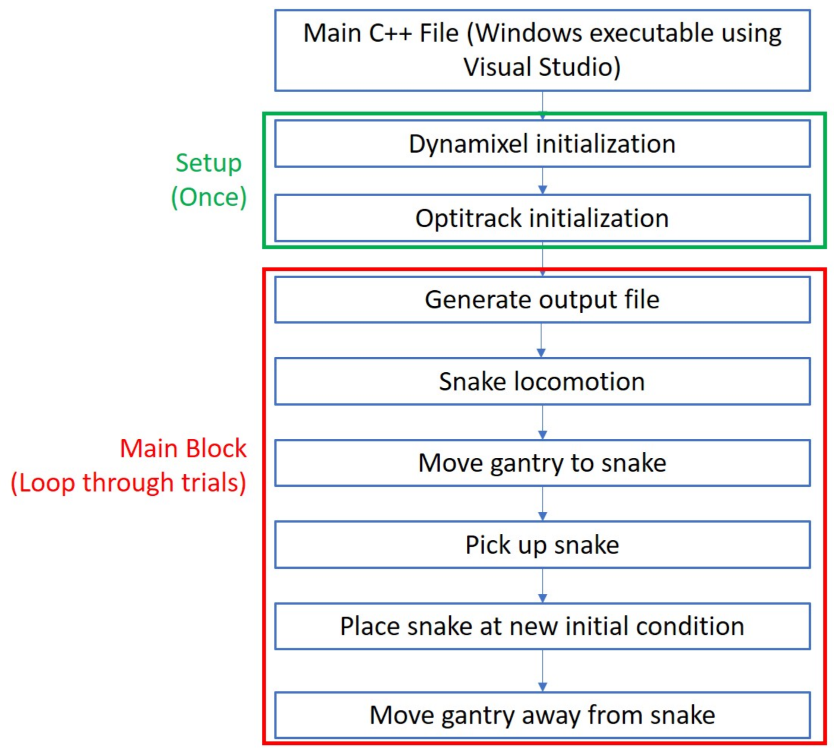

Main C++ File (Windows executable using Visual Studio)
Serial Communication (UART TTL)
Serial Communication (UART TTL)
Motive API
AX-12A Motor Commands
Arduino Mega (Gantry Control)
Optitrack Camera System
Digital GPIO pins
Arduino Mini (onboard snake)
I2C
MPR121 Capacitive Touch Sensor Controller
Snake Locomotion
Capacitive touch sensors/ Contact data
Marker X,Y,Z Position

Figure 3.1: Overview of the three main components which are interfaced through the software. The Main C++ file is written as a Windows executable file, and interfaces with the AX-12A Dynamixel motors, the Arduino Mega, and the Optitrack camera system. Each of these components is broken down further in this section.

The software for the gantry system can be divided into three main programs: the Arduino Mega code, the Arduino mini code, and the main C++ code written as a windows executable file in C++. We selected C++ as the primary programming language because it allows us to interface easily between the Optitrack camera system, the gantry code on the Arduino Mega, and the AX-12A motors through the USB2Dynamixel. Each of these systems has strong compatibility with C++ as shown in (Fig. 3.1). The key takeaway from this figure is how all of the different gantry components communicate with each other in the software. The C++ file communicates with the AX-12A dynamixel motors through a commercially available USB2Dynamixel via serial communication, which will be detailed in Section 7. The C++ file communicates with Arduino Mega via UART serial communication which is broken down further in Section 3.4. Finally, the Optitrack camera system interfaces with the main C++ file via the Motive API which is detailed in Section 3.4.

3.2 Main C++ windows executable file



Main C++ File (Windows executable using Visual Studio)
Setup (Once)
Dynamixel initialization
Optitrack initialization
Main Block (Loop through trials)
Generate output file
Snake locomotion
Move gantry to snake
Pick up snake
Place snake at new initial condition
Move gantry away from snake

Figure 3.2: High level overview of the primary project code written with C++.

The structure of the main C++ program is overviewed in Fig. 3.2. The setup block (green) is only executed once. This block initializes the camera system, connects to all of the motors, and asks for user input on how many experiments to run and where to store the data. The second block is the heart of the program, and is looped through for every trial of data to be collected. The details of how each of these steps is not important yet as they will be explained in detail in the remainder of this section. Our approach in developing the software for this challenging problem was to break the problem down in to a series of tasks that need to be accomplished, and break these tasks down further into manageable subtasks. This simple approach allowed us to develop solutions to one simple subtask at a time, and test this subtask independently to ensure it is functioning correctly before integrating it with the rest of the code. For now, we will ignore the control of the snake motors as that is addressed in Part II, and focus on the Optitrack camera system and the Arduino gantry control.

3.3 Optitrack Camera System

A reliable vision system is crucial for both having quality experimental data and being able to automate the experiments. In particular, having good marker tracking is essential to just about every step in the main loop of Fig. 3.2. For example, during snake locomotion, reliable tracking data for every segment of the snake is needed so that the scattering angle can be determined in post processing. To move the gantry to the snake, we rely heavily upon knowing the position and orientation of both the snake and gantry. To place the snake accurately, we need to have a global reference frame so that we know where the snake should be placed in reference to the posts. Therefore, in order to acchieve the reliable tracking needed to complete these tasks, we selected hardware and software by Optitrack which can be purchased at https://optitrack.com/products/motive/.

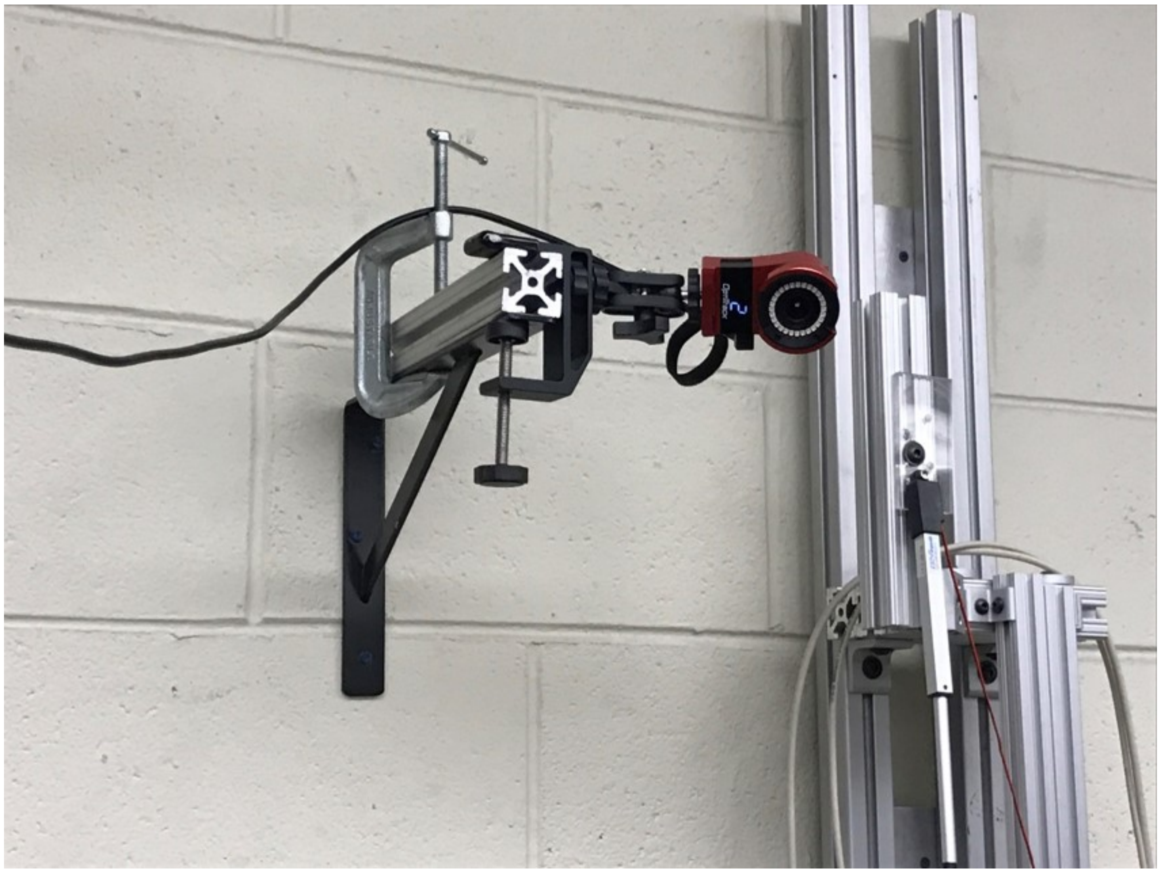


Figure 3.3: One of the six Optitrack cameras mounted around the gantry system. The cameras were mounted so that the snake and gantry could be viewed from a variety of angles, while trying to minimize having the camera's view obstructed by the gantry arm.

3.3.1 Optitrack Camera Hardware

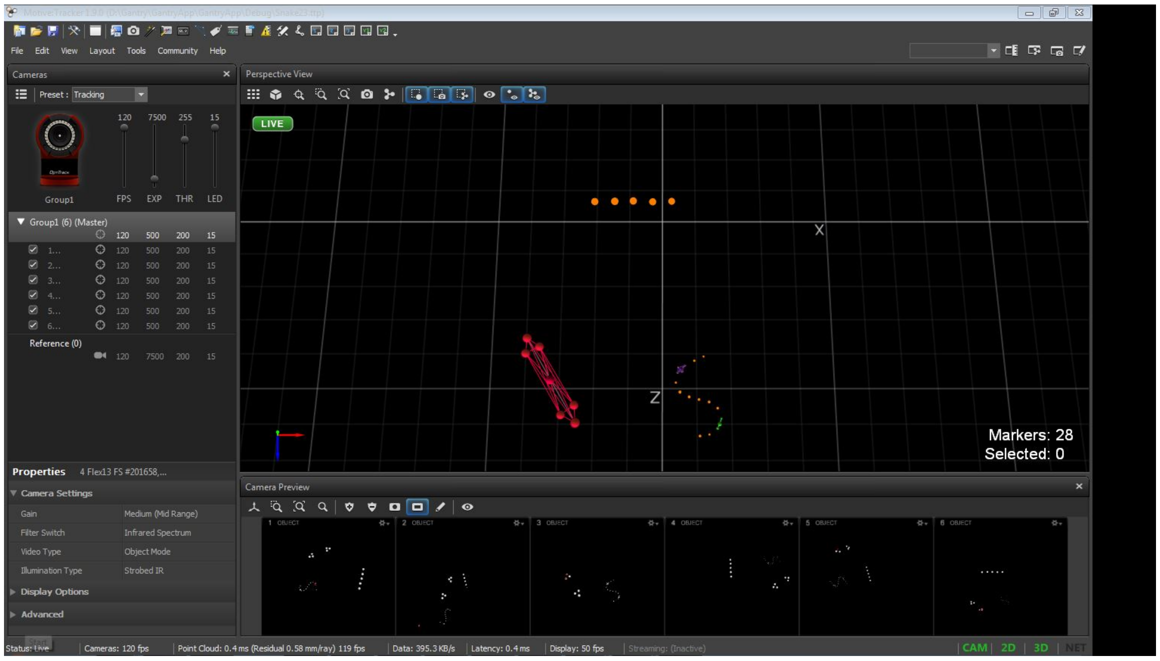
Six Optitrack Flex 13 cameras are mounted above the gantry system are used to track the position of the snake, posts, and gantry. We selected the Flex 13 because it balances high performance at a relatively low cost point compared to other models (about one thousand dollars each). The prime series offers higher frame rates, but the lowest price for these cameras is double that of the Flex 13, and we have observed in previous experiments that 120 frames per second is sufficient for our application.

The number of cameras was increased from four to six when the gantry system was being developed. The Optitrack software requires a marker to be viewed from multiple cameras in order to determine its location in 3D space. Prior to the gantry system, four cameras were sufficient to have clean, reliable tracking data for the snake robot during experiments. Adding the gantry system, however, complicates camera placement significantly because cameras can easily have their view of markers blocked as the gantry moves to different areas of the mat. Even with six cameras, the quality of the tracking data has been one of the greatest challenges in this project. This challenge is addressed in part II as it drove several innovations on the design of the snake to make it more visible to the camera system. Theoretically, this problem can easily be solved by adding more cameras, but it would require installing an additional hub, and would be expensive. Our setup uses the OptiHub 2, which can connect to up to 6 cameras via USB camera cables.

In order to track the location of objects that we care about, Optitrack markers and reflected tape are used to generate markers on the snake, gantry, and posts. Each segment of the robot has at least one Optitrack marker, which allows us to follow the trajectory of each segment. Reflective tape is wrapped around five ping pong balls, which are mounted on the top of each of the five posts. Finally, six ping pong ball markers are attached to the gantry system. These six markers form what is called a rigid body which is used to track the end effector of the gantry (more on this in the software section).

The shape of the gantry gripper arm went through many iterations to reach its current design. The markers are raised and extended beyond the electromagnets in order to be more easily visible to the cameras above. The overall form factor, however, needed to remain compact since early iterations of this design frequently got caught on the snake's power cable when the gantry went to pick up or put down the robot. Finally, the shape of the rigid body was intentionally designed to be symmetric so that its centroid lines up with the middle of the gripper arm.

3.3.2 Optitrack Camera Software



Preset : Tracking

	FPS	EXP	THR	LED
Group1	120	7500	255	15
Group1 (6) (Master)	120	500	200	15
1...	120	500	200	15
2...	120	500	200	15
3...	120	500	200	15
4...	120	500	200	15
5...	120	500	200	15
6...	120	500	200	15
Reference (0)	120	7500	200	15

Properties 4 Flex13 FS #201658,...

Camera Settings

Gain	Medium (Mid Range)
Filter Switch	Infrared Spectrum
Video Type	Object Mode
Illumination Type	Strobed IR

Markers: 28
Selected: 0

Status: Live | Cameras: 120 fps | Point Cloud: 0.4 ms (Residual 0.58 mm/ray) 119 fps | Data: 395.3 KB/s | Latency: 0.4 ms | Display: 50 fps | Streaming: (Inactive)

Figure 3.4: An illustration of how the Motive software is used in our setup. The large red rigid body represents the gantry, the five large orange markers are the posts, the green and purple rigid bodies are the magnetic plates on the snake, and the remainder of the orange markers are the segments of the snake.

The camera hub directly interfaces with a software called Motive shown in Fig. 3.4. Motive offers extensive tools for tracking markers, and only a fraction of its vast capabilities are used in this project. Extensive documentation and support on this software is provided here https://optitrack.com/support/software/motive-tracker.html.

The first step of using this software is calibrating the cameras. In order to calibrate the cameras, a calibration wand and calibration square are required. The camera calibration procedure is briefly summarized in the following steps:

1. Remove all markers from the view of the cameras. Anything that is reflective can

easily be picked up by the cameras. Check the view for each individual camera in grayscale mode and object mode.

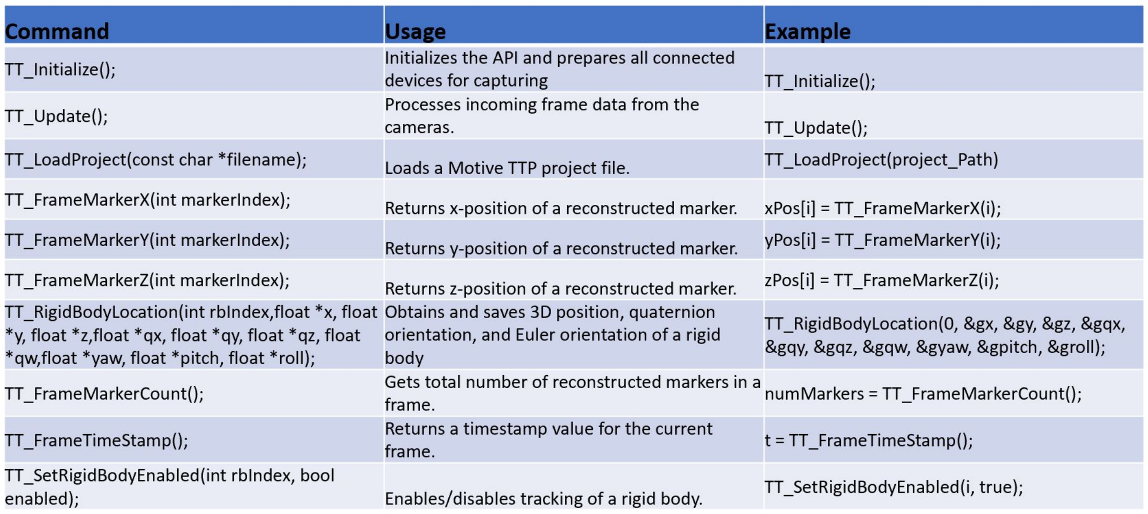
2. If there are any markers which cannot be obscured or removed, mask them with the Optitrack Software.
3. Begin calibration, and specify a location to save the Optitrack project.
4. Wave the calibration wand in front of each camera, ensuring to fully and evenly cover its entire view.
5. Wave wand over the entire mat to ensure complete coverage.
6. Finish Calibration, and save the project.
7. Set the ground plane so that the X axis is parallel to the array of posts, and clearly in view of all of the cameras.

Another important feature of the Optitrack Software is the creation of rigid bodies. Creating a rigid body allows you to specify the multiple points on an object and track its geometric centroid. We use rigid bodies to track the position of the gantry as well as the position of the magnetic plates on the snake. A couple of rules must be followed in order to specify a rigid body, which are important to consider when placing markers on an object.

1. The distance between markers must always remain fixed (this is necessary from the definition of a rigid body).
2. A rigid body must contain three or more markers. More than three is preferable since it will continue to track the object if a marker is temporarily dropped.
3. The three markers must not all lie on the same plane. In order to assist the software, we recommend making rigid bodies very distinct from each other. In our experience, rigid bodies with similar geometries can easily be confused.

Defining two rigid bodies on the snake significantly simplified the calculation necessary to find the magnetic contact plates. Instead of trying to sort the markers on the snake and estimate the correct location of the magnetic plates, we simply define rigid bodies on these magnetic plates and let the Optitrack software do the work.

In practice, Motive is used very infrequently for this project besides for calibrating the cameras and defining rigid bodies. Instead, we use the Motive API because it allows the camera information to be integrated with the rest of our code and called in real time. A very detailed set of documentation on the Motive API is available online at the Motive API Function Reference webpage [7]. Frequently used functions are summarized in table 3.5 below, and then we explain how they are used in each part of the main C++ code to communicate with the cameras.



Command	Usage	Example
TT_Initialize();	Initializes the API and prepares all connected devices for capturing	TT_Initialize();
TT_Update();	Processes incoming frame data from the cameras.	TT_Update();
TT_LoadProject(const char *filename);	Loads a Motive TTP project file.	TT_LoadProject(project_Path)
TT_FrameMarkerX(int markerIndex);	Returns x-position of a reconstructed marker.	xPos[i] = TT_FrameMarkerX(i);
TT_FrameMarkerY(int markerIndex);	Returns y-position of a reconstructed marker.	yPos[i] = TT_FrameMarkerY(i);
TT_FrameMarkerZ(int markerIndex);	Returns z-position of a reconstructed marker.	zPos[i] = TT_FrameMarkerZ(i);
TT_RigidBodyLocation(int rbIndex,float *x, float *y, float *z,float *qx, float *qy, float *qz, float *qw,float *yaw, float *pitch, float *roll);	Obtains and saves 3D position, quaternion orientation, and Euler orientation of a rigid body	TT_RigidBodyLocation(0, &gx, &gy, &gz, &gqx, &gqy, &gqz, &gqw, &gyaw, &gpitch, &groll);
TT_FrameMarkerCount();	Gets total number of reconstructed markers in a frame.	numMarkers = TT_FrameMarkerCount();
TT_FrameTimeStamp();	Returns a timestamp value for the current frame.	t = TT_FrameTimeStamp();
TT_SetRigidBodyEnabled(int rbIndex, bool enabled);	Enables/disables tracking of a rigid body.	TT_SetRigidBodyEnabled(i, true);

Figure 3.5: Frequently used functions called from the Motive API.

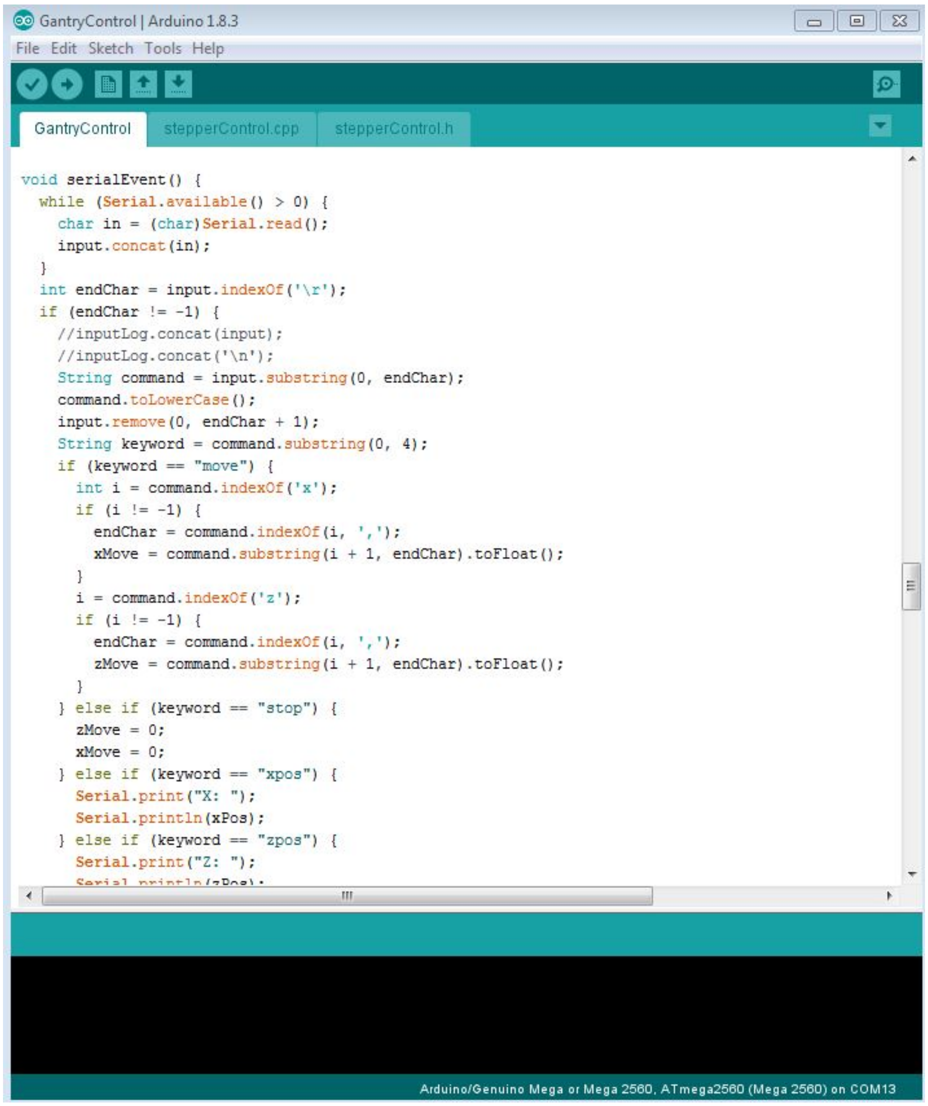
The initialize and load project commands are used during the Setup block to prepare the camera system for data acquisition. The update command is used frequently throughout the entire code to ensure that the data is being pulled from the most recent frame available. The X,Y,Z, marker count, marker commands are used during the snake locomotion loop to pull the location of every marker, then write them to an excel spreadsheet.

The rigid body location command is used to calculate the positions of the gantry and two snake contact markers in the ”pick up snake” sub-task. The orientation of the snake is calculated from the location of the two snake contacts using inverse tangent. This angle is sent to the servo motor so that the the electromagnets line up above the magnetic contact plates. Using the average value of the two snake contact markers, the program calculates the distance between the gantry and the snake, then commands the stepper motors to travel that distance. We noticed that, due to slight slipping in the timing belt, the gantry does not travel exactly the distance commanded. Therefore, to ensure that we will accurately arrive at the snake’s exact location, the gantry actually moves towards the snake twice. The first ”coarse” adjustment moves the gantry very close to the magnetic plates. At this point, the gantry position is requested again, and the distance between snake and gantry is sent to the motors again. This second ”fine” position adjustment corrects from the error caused by the slipping in the timing belt.

Finally, Set Rigid Body Enabled is used to turn the rigid bodies on and off. We noted in the previous paragraph how important the rigid bodies were for finding and picking up the snake. There are, however, times when we do not want the rigid bodies to be active. During the snake locomotion loop, camera data must be taken and recorded quickly since this data is requested every time a position is commanded to the snake motors. Tracking rigid bodies is relatively time costly since the software must correctly identify the markers on the rigid body and calculate their centroid. For this reason, during the snake loop, only raw marker data is tracked, and when the rigid body data is requested to pick up the snake, the snake has stopped moving and timing is not important.

This section illustrates how the Optitrack data is used to automate the tasks in 3.2 by informing how to move the gantry system. The next section explains how the gantry system is controlled by the Arduino Mega, and the electronics necessary to execute these commands.

3.4 Arduino Gantry Control



```
void serialEvent() {
  while (Serial.available() > 0) {
    char in = (char)Serial.read();
    input.concat(in);
  }
  int endChar = input.indexOf('\r');
  if (endChar != -1) {
    //inputLog.concat(input);
    //inputLog.concat('\n');
    String command = input.substring(0, endChar);
    command.toLowerCase();
    input.remove(0, endChar + 1);
    String keyword = command.substring(0, 4);
    if (keyword == "move") {
      int i = command.indexOf('x');
      if (i != -1) {
        endChar = command.indexOf(i, ',');
        xMove = command.substring(i + 1, endChar).toFloat();
      }
      i = command.indexOf('z');
      if (i != -1) {
        endChar = command.indexOf(i, ',');
        zMove = command.substring(i + 1, endChar).toFloat();
      }
    } else if (keyword == "stop") {
      zMove = 0;
      xMove = 0;
    } else if (keyword == "xpos") {
      Serial.print("X: ");
      Serial.println(xPos);
    } else if (keyword == "zpos") {
      Serial.print("Z: ");
      Serial.println(zPos);
```

Arduino/Genuino Mega or Mega 2560, ATmega2560 (Mega 2560) on COM13

Figure 3.6: The Arduino IDE was used to interface with the microcontroller and develop low level functionality for sub-components of the gantry system. Commands are given to the program via serial interrupts, and any of the gantry subcomponents can be directly controlled through the Arduino Serial Monitor.

As shown in Fig. 3.1, the Arduino Mega is essential for handling all of the control of the gantry system, as well as interfacing with the capacitive touch sensors. The Arduino controls the X and Z stepper motors, the servo motor, the electromagnets, the frigeli motor (Y axis), and interfaces with the contact sensing on board with the snake. The remainder of this section will discuss the Arduino software for each of these components in detail, except for the contact sensing data which will be addressed in Section 7.

3.4.1 Arduino Program Overview

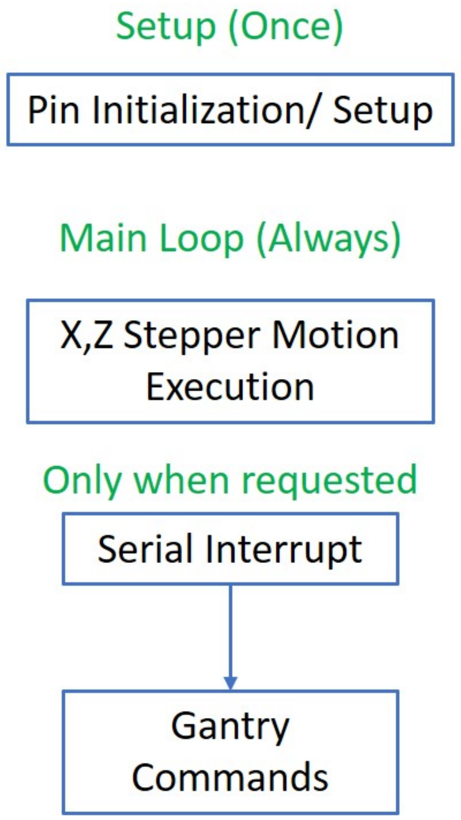


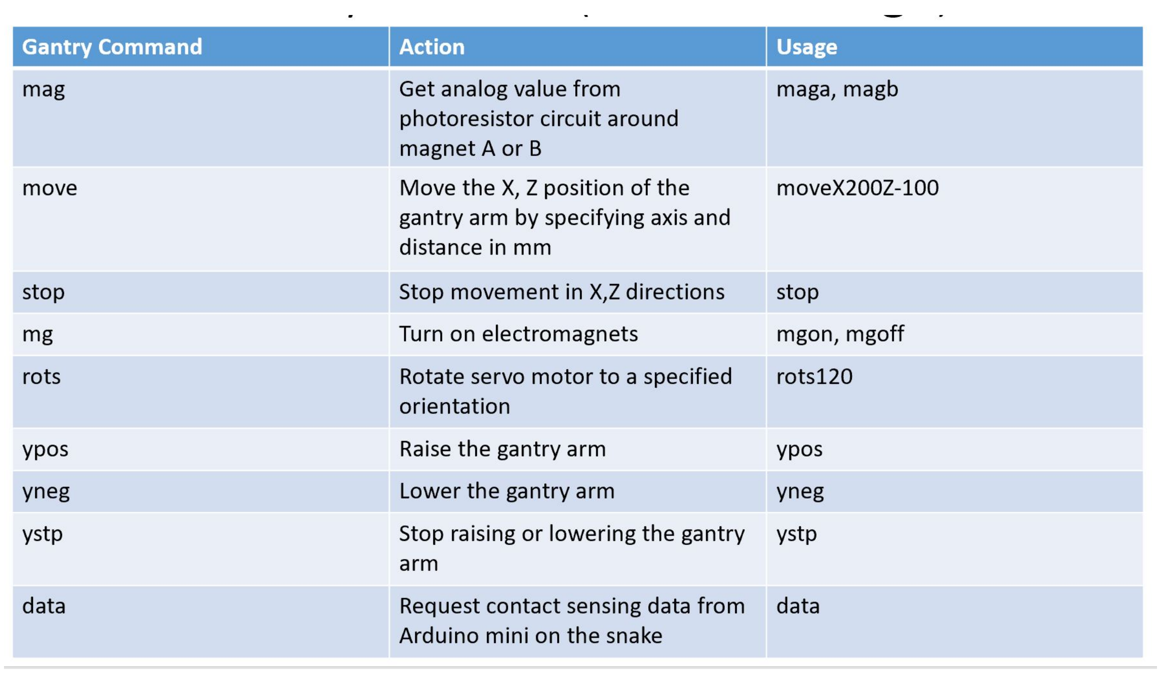
Figure 3.7: Overview of the structure of the arduino program.

Fig. 3.7 overviews the structure of the Arduino program. The first section initializes the pins to be inputs or outputs depending on how they are being used by the subcomponents. The second section executes X and Z axis gantry locomotion for the stepper motors. This section is also the main Arduino code, so always is running as an infinite loop. Without being given any input, this loop does nothing. When the values for zMove or xMove are modified in the serial interrupt, however, the motors are activated. Every time the main loop is executed, a pulse is sent to the stepper motors, and the value of zMove or xMove is decremented until it reaches zero, and the code returns to its normal idle state. It is also important to note that when the gantry finishes executing a X and Z command, the phrase ”done moving” is written to the serial monitor. This is an essential step that drastically improves the speed of certain parts of the main C++ file. The way that the C++ program executes is that as soon as a line of code is executed, it moves on to the next line of code. In many circumstances, however, we want the code to wait until the gantry is done moving before the next lines of code execute. In order to do this, the program enters an infinite while loop while the gantry arm is moving. In this loop, the program is polling the serial monitor, and when the Arduino sends the done moving command to the C++ program, the C++ program brakes the loop and continues execution with the next command to the gantry. This is crucial because we don’t know how long it will take the gantry to reach the snake since it may scatter to many different locations and orientations.

The majority of the gantry control actually occurs within the serial interrupts in the third part of the code. In addition to triggering the X and Z locomotion in the main loop, the serial interrupts also allow for control of the electromagnets, photoresistor circuit, rotary servo, Frigeli motor for Y axis locomotion, as well as interfacing with the contact sensing on the snake. The following section explains how these functions work, and breaks down the electronics necessary to accomplish these tasks (except for the snake contact sensing, which is addressed in detail in part II.)

CHAPTER 4

GANTRY ELECTRONICS



Gantry Command	Action	Usage
mag	Get analog value from photoresistor circuit around magnet A or B	maga, magb
move	Move the X, Z position of the gantry arm by specifying axis and distance in mm	moveX200Z-100
stop	Stop movement in X,Z directions	stop
mg	Turn on electromagnets	mgon, mgoff
rots	Rotate servo motor to a specified orientation	rots120
ypos	Raise the gantry arm	ypos
yneg	Lower the gantry arm	yneg
ystp	Stop raising or lowering the gantry arm	ystp
data	Request contact sensing data from Arduino mini on the snake	data

Figure 4.1: Commonly used commands for controlling the gantry system. Commands are triggered by writing the appropriate key word into the serial monitor, or by sending the commands from the main C++ file to the Arduino using UART communication.

As we highlighted in Chapter 3, the majority of the gantry control is accomplished in the serial interrupt section of the Arduino Mega code. Figure 4.1 lists the commands which can be written to the serial monitor to control the electromagnets, servo motor, Frigelli motor, and photoresistor circuit. As mentioned in Chapter 3, the gantry can be driven directly from the arduino serial monitor, which allows each of these subcomponents to be developed, coded, and tested independently. In this section, we go in depth into how each of these subcomponent works, as well as the hardware associated with working these sensors and actuators.

4.1 Microcontroller Selection

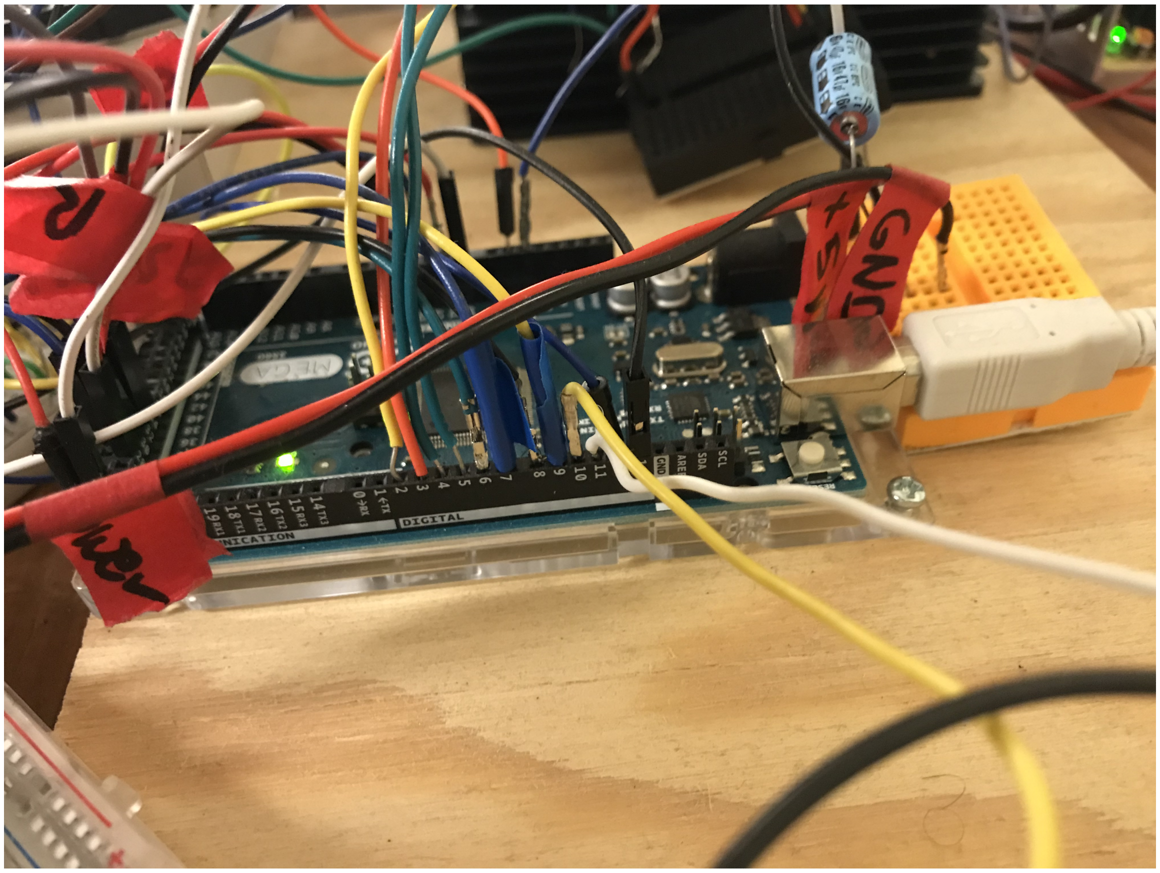


Figure 4.2: An Arduino Mega was selected as our microcontroller based on its low cost, high usability, and large number of available pins.

An Arduino Mega was selected as the microcontroller to control the gantry system. Arduinos are used frequently in our lab due to the low price point (about 36 dollars for a Mega). Additionally, the Arduino IDE is extremely easy to learn for someone without experience working on microcontrollers, and plenty of documentation and support is available online. Communication protocols such as UART, I2C, and SPI are much easier to work with compared to similar cost microcontrollers like the TI MSP432. The Mega in particular has a much larger pinout selection, which was crucial since this Arduino must coordinate with many different components. The Arduino Mega has a total of 54 GPIO pins and 16 Analog input pins, and a majority of the pins on the board are currently being used by our setup.

4.2 Sensors

A wide variety of sensors are required in order to automate the experiments in a robust manner. In addition to the Optitrack cameras which were addressed in the previous section, the primary sensors used for the gantry system are the limit switches and the photoresistor circuits.

4.2.1 Limit Switches

Limit switches are a crucial safety feature to prevent damage to the stepper motors by switching the motors off when the limit of the range has been hit. Additionally, the limit switch associated with the Y-axis allows us to determine the correct height to place the snake. Since the Frigelli motor is controlled with an on-off controller, the Y-axis limit switch is essential to stopping the gantry arm at the appropriate height to pick up the snake robot. While there is no direct command for the limit switches in the program, they are activated immediately on contact. For example, in the case of the X and Z axis limit switches, the main stepper code mentioned in section 3 is constantly checking the states of these switches. If the appropriate limit switch is hit, the X and Z actuation stops. Furthermore, the Y axis limit is crucial to raising and lowering the snake to the ground. The height of this limit switch is easily adjusted by sliding the 3D printed trigger up or down in the open builds V slot linear rail. Setting this height is essential to successfully making reliable contact between the magnetic plates on the snake and the electromagnets. In addition to the limit switch, a clever design of the 3D printed connection between the gantry arm and the electromagnetic gripper allows for some compliance between these two parts of the gantry. Briefly, two pieces of foam are inserted in this blue connection piece: one piece below the open builds rail and one above the open builds rail. These foam pieces allow the gripper to be lowered so that there is some force between the electromagnets and the magnetic plates,

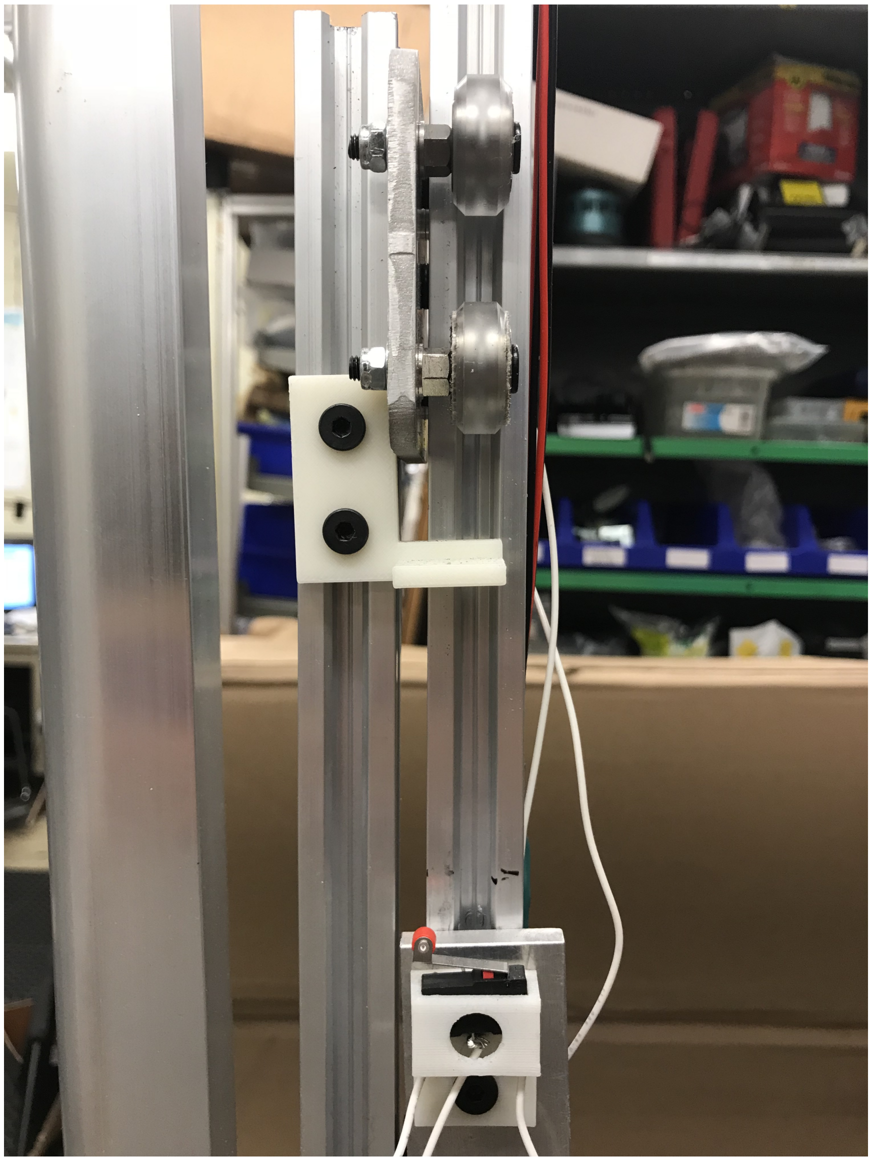


Figure 4.3: The limit switches are crucial to the safety of the gantry system. The height of the Y-axis limit switch (shown) is important to ensuring good contact between the magnetic plates and electromagnets.

but now the foam can deform in a way which prevents this force from damaging the gantry arm as well as ensuring that both electromagnets are fully making contact with the two magnetic plates. A similar compliance is also built into the snake by placing foam between the magnetic plates and the 3D printed support, which will be discussed in Section 7. Combining this compliance with the correct Y-axis limit switch height and proper calibration of the servo motor ensures that when the gantry receives accurate rigid body position data for the snake and the gantry, the gantry will be able to pick up the snake consistently.

4.2.2 Photoelectric Resistors

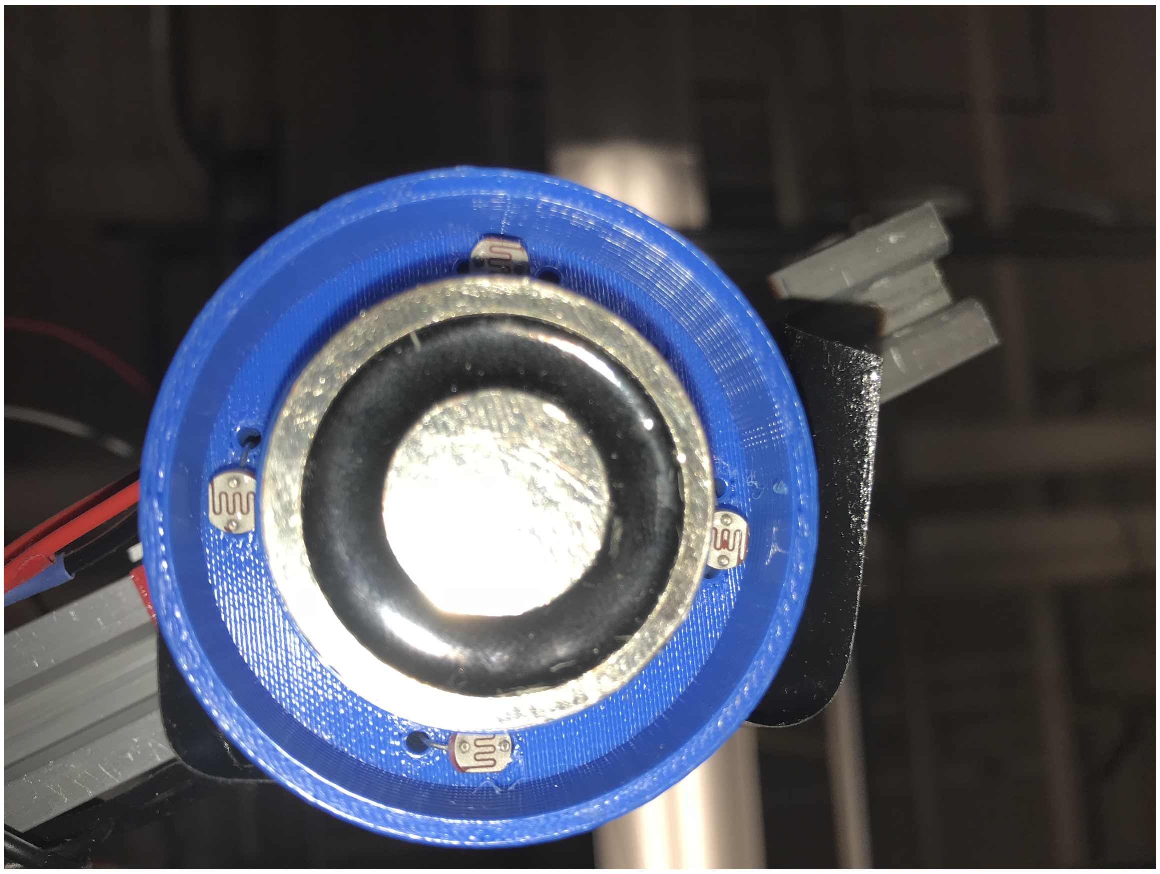


Figure 4.4: This simple photoresistor circuit determines the proximity of the electromagnet to the magnetic plate on the snake. A threshold value is determined for what constitutes successful contact, allowing the program to exit in the case of poor contact with the snake.

The photoelectric resistors (Fig. 4.4) are important for detecting if this contact is reliable. As mentioned in Section 2.1, the circuit acts as a simple proximity sensor so that we can detect if something is wrong. While debugging the Optitrack data, we observed that if the gantry moves to the wrong location, it will still attempt to pick up the snake and move it back to the start position which could cause the snake to be dragged by its wires, potentially damaging the snake or the posts. Therefore, the photoelectric resistors act as a simple safety switch to prevent this from happening. Within a 3D printed shell designed to fit around the exterior of the electromagnets, four connected photoresistors return a voltage corresponding to the light intensity detected since each photoresistor decreases its resistance with an increase of light intensity. This voltage signal is filtered by a capacitor then sent to two analog input pins on the Arduino Mega (one for each electromagnet). Through a trial and error process, we determined what the appropriate threshold should be for successful contact. Even with the capacitor, however, there are still occasional spikes in the analog signal so there is also some filtering done in software to make the best estimate for the state of the contact. This circuit will occasionally shut down the program in cases where there is only partial contact between the magnet and the plate, but this error was considered more acceptable than the alternative case where the magnets don't make contact and the system continues. Still, even with this occasional error this circuit is effective at preventing the program from continuing when there is clearly no contact.

4.3 Actuation

The information gathered from the sensing is used to inform the actions of the actuators. We saw previously how the camera data is used to dictate the motion of the stepper motors, how the Y-axis limit switch is important for the frigelli linear actuator, and the relationship between the photoresistor proximity sensors and the electromagnets. In this section, we will briefly discuss the hardware behind this actuation.

4.3.1 Stepper Motors

Two OpenBuilds Nema 23 stepper motors were selected to actuate the X and Z directions of the gantry. These motors are controlled via a WANTAI 2H microstep driver. Fig. 4.5 shows how the motor is wired to the driver and Arduino. Two GPIO output pins from the Arduino supply the Pulse and Direction signals to the driver. The motors are powered by a variable power supply at 25.7 V, and together draw about 1 Amp of current during operation. The driver converts the pulse and direction signal into the appropriate signal for

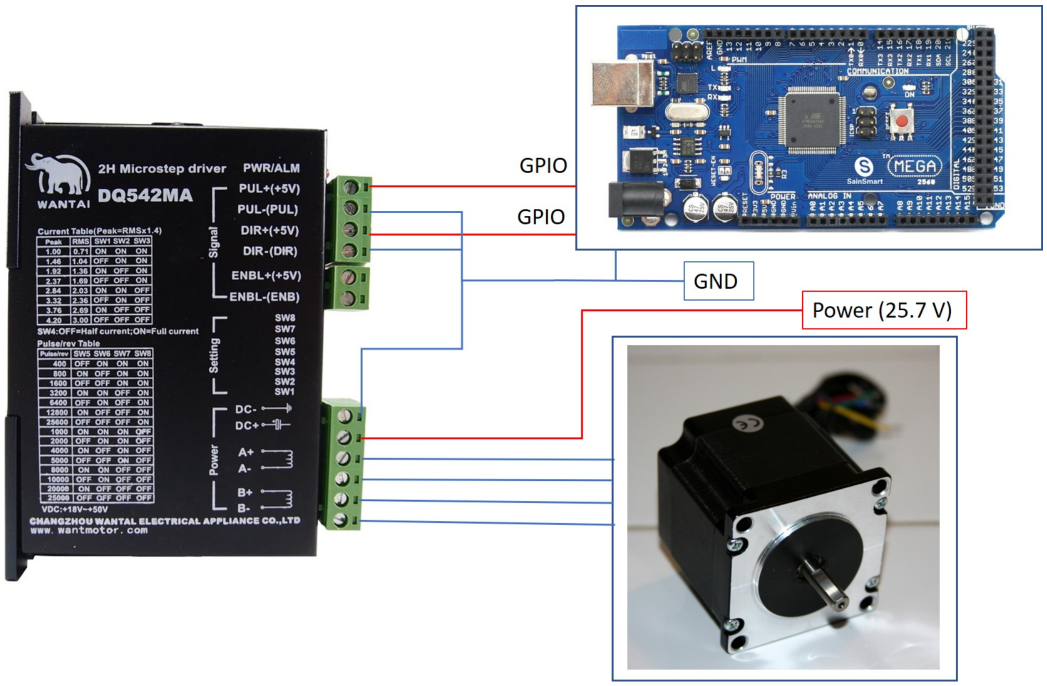


Figure 4.5: An overview of the stepper, motor driver, and Arduino circuit for X and Z axis actuation.

the two phases of the stepper. Finally, all of the Arduino, stepper motor, and the driver are connected to a common ground.

Fig. 4.6 shows how the steppers are implemented to control actuation in the gantry system. This photo shows a closeup of the Stepper motor on the X axis just above the extendable gantry arm. A gear is attached to the output shaft of the stepper motor which meshes with a timing belt which runs the entire length of the X axis. A machined steel plate with wheels is used to interface the gantry arm with the open build rails so that the entire arm can can roll left and right along these rails. A similar mechanism is used to translate the entire X-axis rail in the Z direction. As mentioned previously, the drag chains associated with these two axes were essential for preventing wires from getting caught and damaged during gantry locomotion.

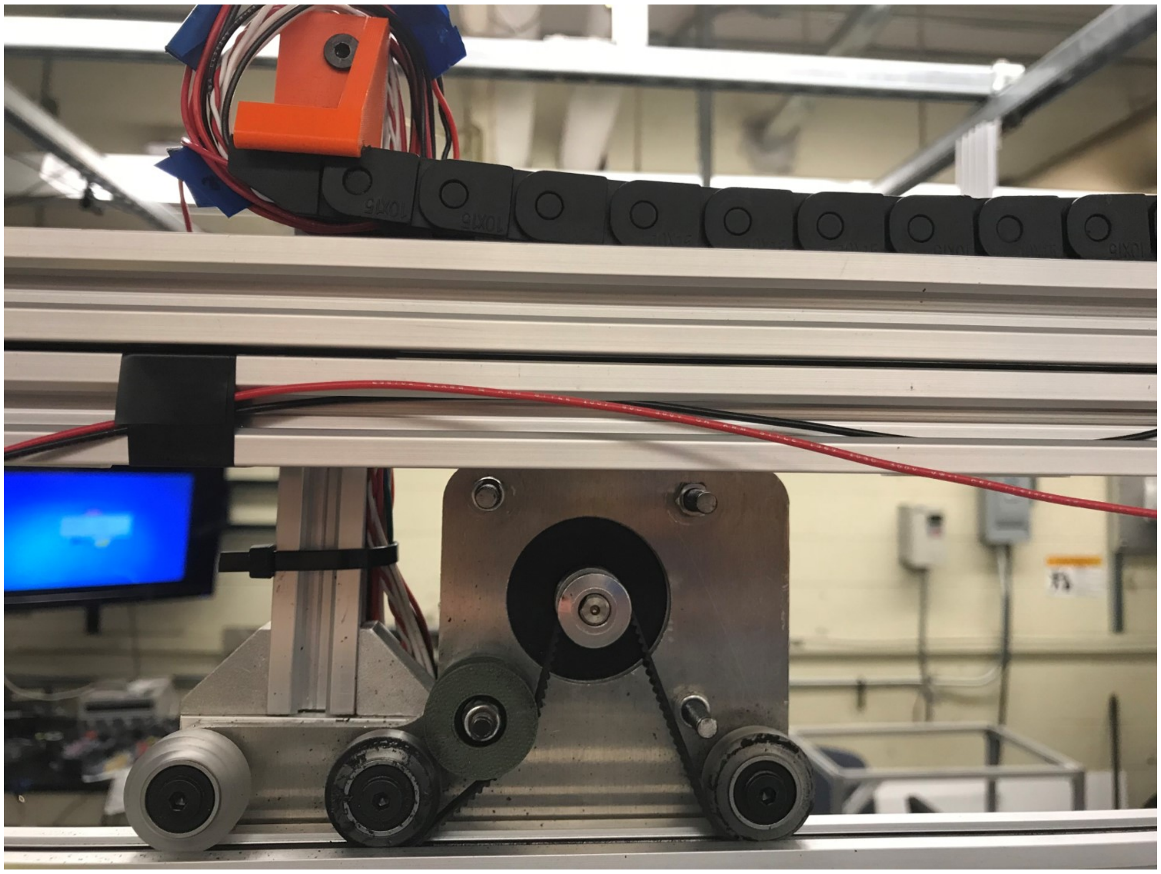


Figure 4.6: An example of how the Nema 23 is implemented in the gantry system to drive X axis locomotion.

4.3.2 Servo Motor and Electromagnets

Fig. 4.7 shows the electromagnets and the servo motor on the electromagnetic gripper. The magnets are turned on and off using a single GPIO output pin on the arduino with the help of a relay switch (shown on the red 3D printed mount just above the blue arrow). As discussed previously, the blue connection 3D printed segment allows for compliance for better connection between the magnets and the magnetic plate. The Hitec servo motor is located just below the relay switch mount and provides 180 degrees of rotation. In order to accurately command the gripper arm to a known angle, the input to this servo must be calibrated to an output angle in the real world. In order to do this, the servo was commanded to reach a world angle of 45 degrees (with 0 degrees corresponding to rotating

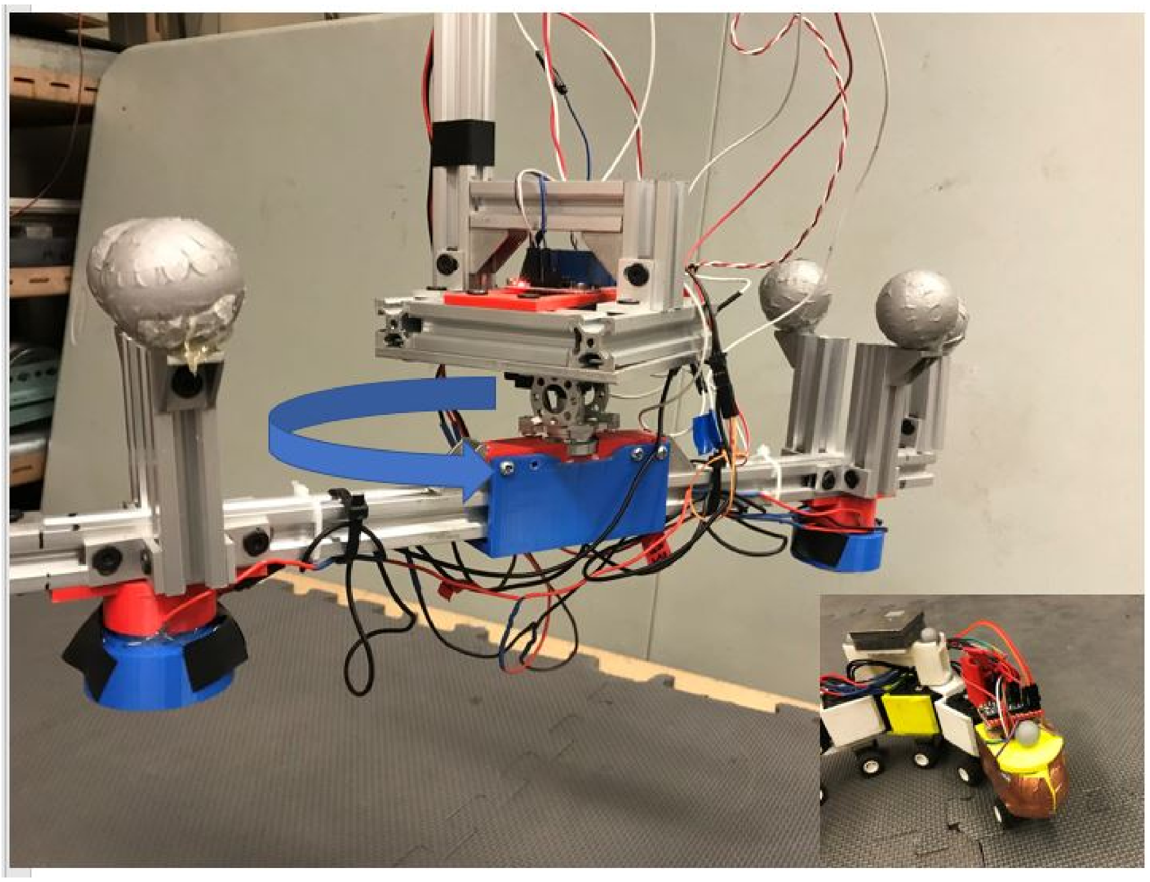


Figure 4.7: Electromagnetic gripper at the end of the extendable gantry arm. The electromagnets are encased in the two blue 3D printed cases and the servo motor rotates the entire gantry arm by the angle shown in blue. An inset photo of the snake is included to highlight the interface between the electromagnets of the gantry and the magnetic plates on the snake.

the servo clockwise until the gantry gripper is parallel to the line of posts) using a trial and error process. The input required to generate this angle is recorded, then the servo is commanded to an angle of 90 degrees (perpendicular to the posts) and this servo input is recorded. Using these two data points, we were able to come up with the equation of a line that relates the world frame angle of the gripper arm to the servo input required to move it there. This equation is used to determine what servo input is necessary to line up the gantry arm with the magnetic plates on the snake after calculating the angle the snake rigid bodies create using the Optitrack data. For most scattering angles of the snake, this calibration is extremely effective for aligning the snake and the gripper. However, near the two extremes of the servo's range (0 degrees and 180 degrees), this relationship no longer holds, and the alignment between the gripper and snake may be a bit inaccurate. Additionally, in some cases, the snake is completely unable to make it through the posts and actually is reoriented by over 90 degrees in either direction. In both of these cases, the photo resistor circuit is crucial to determining if the contact is good enough to effectively pick up the snake.

4.3.3 Frigelli Motor

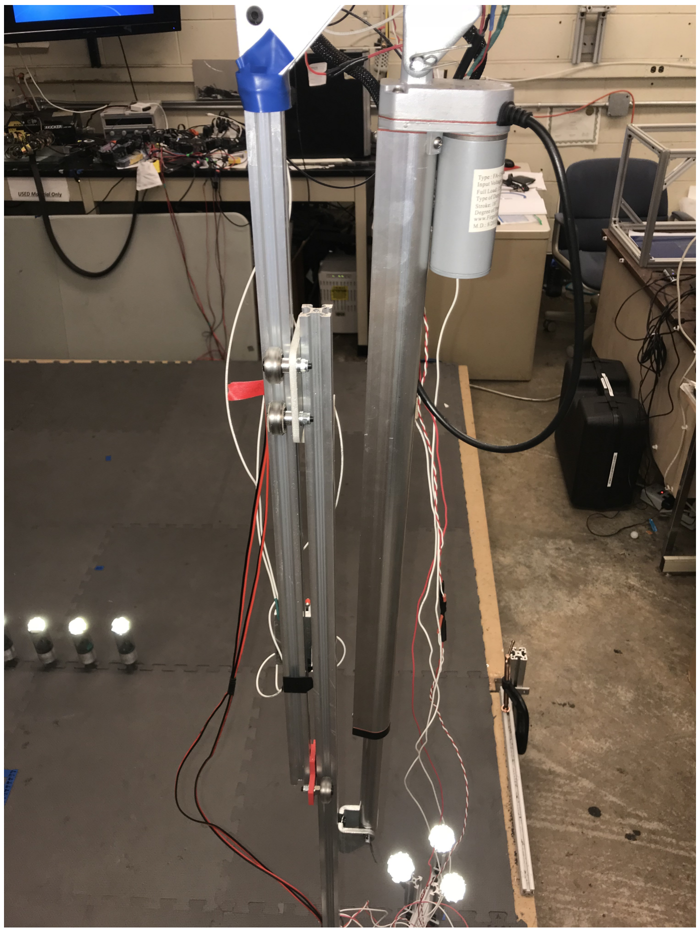


Figure 4.8: A Frigelli linear actuator is used to raise and lower the vertical gantry arm to pick up the snake robot.

Fig. 4.3.3 shows how the Frigelli FA-150-S-15-18 linear actuator raises and lowers the gantry vertical arm to pick up the snake robot. Two open builds linear v rails are installed in parallel. One of the rails is mounted directly to the machined plates overhead, and the other rail is mounted to the output shaft of the frigelli. A machined plate with wheels allows the Frigelli rail to extend and retract in parallel to the fixed rail. A limit switch attached near the bottom of the fixed rail defines the end of the fixed rail. As discussed previously, this limit switch is essential to setting the height of the gantry when picking up the snake. Similar to the electromagnets, the Frigelli is controlled using a combination of GPIO pins and a relay switch. The Frigelli can only be told to move positive, negative, or stop moving so having this Y-axis limit switch is crucial to automating the pickup of the snake.

CHAPTER 5
GANTRY RESULTS

A video demonstration of the complete gantry system functioning correctly (as well as the open loop and controlled snake described in part II) can be seen in the video at https://youtu.be/GO3HrDTGhZ0.

Note that this video is sped up roughly 100x to show the snake complete a full set of experiments. In this early demonstration of the gantry system, individual experiments took about 5 minutes on average. However, since the time this video was taken, considerable improvements to the code have allowed the system to complete a full experiment closer to 3 minutes. Note that individual experiment times may vary based on the location that the robot scatters. Over the course of the past year and a half, this gantry system has performed thousands of experiments for evaluating and improving its performance as well as developing and testing the snake robot. In the single post setup, there were very few issues which could stop the gantry from running. During these experiments, I could set up the gantry and leave the lab for hours at a time knowing that there were few possible scenarios where the gantry system would ”fail,” and that appropriate precautions were taken to mitigate the risk associated with these scenarios.

At its top performance, the gantry system was able to complete over 150 experiments consecutively with no human input on multiple occasions. However, as with any system that is used so heavily, the repeated use of the gantry caused some of the gantry components to wear out over time. In particular, the reflective tape used on the snake robot to identify the magnetic plates needed to be replaced frequently. In addition, other parts which were replaced due to consistent wear including the AX-12A motors on the snake, the stepper motor wires, and the snake robot wheels.

The multiple post setup, however, provided additional complexities which prevented the gantry from running unsupervised for these long periods of time. In particular, certain initial conditions caused the robot to get completely caught as it moved through the posts. For this reason, I preferred to always have someone in the room when running the system for the multipost experiments. If the snake ends an experiment still in the middle of the posts, lowering the gantry arm to the snake could damage the posts or the arm, and there was nothing in the software that could prevent that from happening. Additionally, potentially hazardous scenarios were simply more common with the multipost setup since nearly every experiment resulted in the robot hitting a post whereas with the single post, most experiments do not hit the post.

Even with these limitations, the gantry system was effective at collecting the necessary data for the multipost experiments. Furthermore, such a system can be easily generalized to pick up different types of robots, as well as to pick up and move obstacles in order to generate an arbitrary terrain for the robot to navigate. This natural extension of the gantry system can be used to explore really interesting problems in terms of path planning, navigation, and control of a biologically inspired robot through an arbitrary environment.

Part II

Snake Robot Anticipatory Controller

CHAPTER 6

CONTROLLER INTRODUCTION

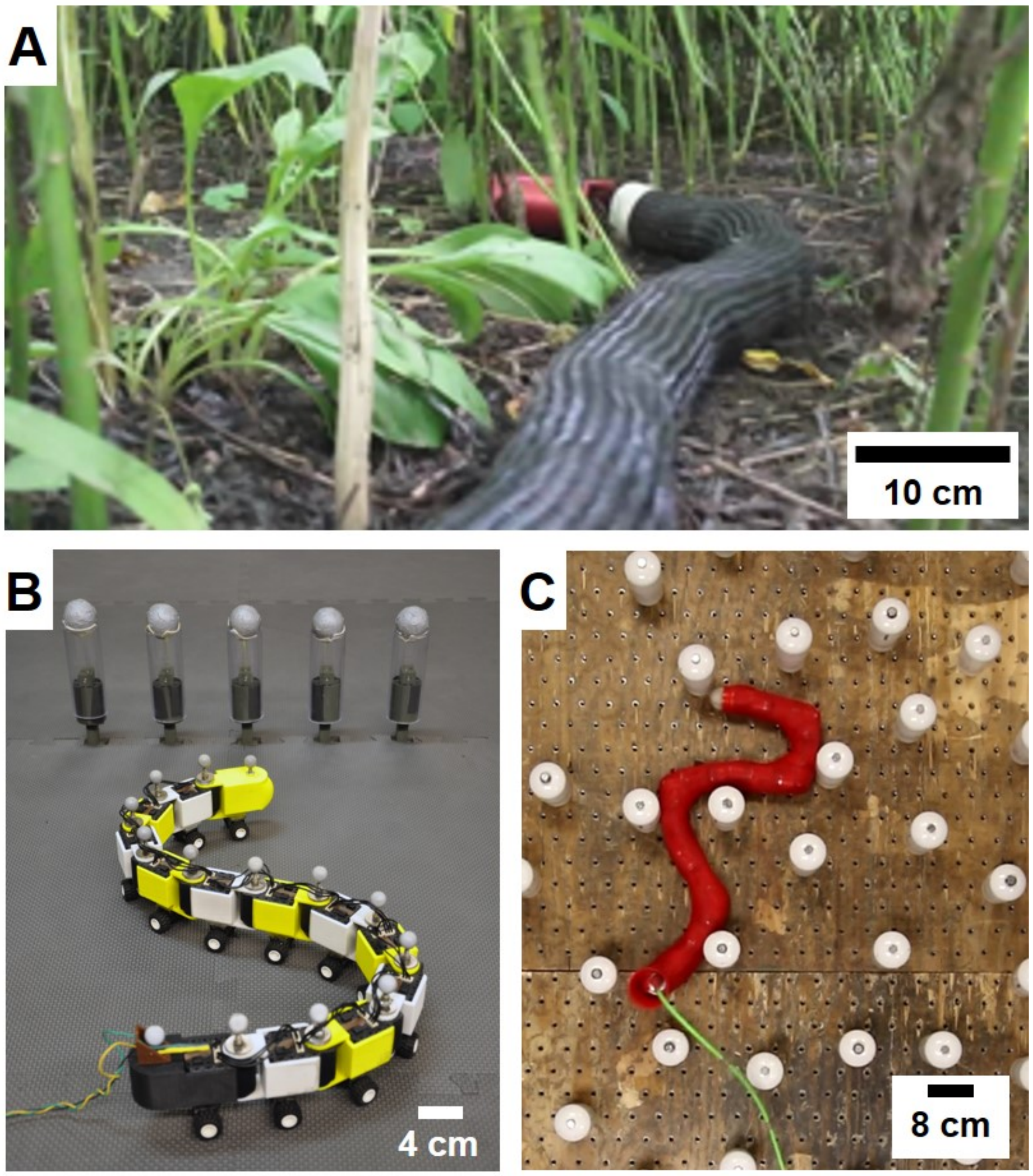


Figure 6.1: Snake robots in unstructured environments of varying levels of complexity. Images A and C courtesy of the Biorobotics lab at Carnegie Mellon University.

6.1 Motivation

For scenarios where a snake robot may be deployed in the field such as disaster response and surveillance, collisions with obstacles are not only highly likely, but often unavoidable. For example, Fig. 6.1-A illustrates the type of heterogeneous environment that a snake robot may encounter in field deployment. Understanding what happens during the interactions between robot and obstacles, and coming up with effective control strategies to locomote through these environments is essential for successful deployment. As mentioned previously, our approach to this challenge is to develop a controller that utilizes the physics of these collisions. While machine learning techniques may be used to train robots to navigate through challenging environments (such as in [8]), we don't really understand how the solution that the reinforcement learning generates works at a fundamental level. With that said, previous work in snake robot control and sensing, is still essential for developing and understanding our controller in the context of existing frameworks for controlling snake robots.

6.2 Previous Snake Robot Control Systems

Early work in the field [1] attempted to use contact sensing along the robot in order to avoid obstacles. More recent work, however, seeks not to avoid collisions with obstacles but to use them advantageously. In [9] interactions between snake robot and obstacles are modeled as a hybrid dynamical system which includes the continuous dynamics of the robot locomotion as well as discrete events which model the collisions with obstacles. This model is tested in simulation and experiment and used to develop a controller. This controller is composed of two main schemes: a leader follower scheme for free snake robot locomotion, and a jam resolution scheme to free the robot when it gets pinned on obstacles. Finally, a jam detection scheme determines which state the controller should be in. If a single joint deviates from its commanded angle by more than a given threshold, the joint is defined to be jammed. If more than two joints on the robot are jammed for greater than some threshold, the robot will enter the jam resolution scheme.

The main drawback to this scheme, however, is that there are only two distinct states of snake motion, and it only modifies its behavior once the snake is completely jammed. Conversely, shape based controllers address these limitations by continuously varying the parameters of the snake's waveform to modify its local curvature based on sensing of the robot's environment. The idea of shape based controllers, first presented in [10] and utilized in [11] and [12] provides a natural framework for thinking about snake locomotion. These works control the snake robot to follow a family of shapes which allow the robot to locomote through complex environments. While our system lacks the inertial sensing to fully utilize these schemes, we will show in the following section that we are able to implement a special case of this controller using only very simple onboard sensing to control the orientation of the robot. Our novel contribution to this field, therefore, is not the mechanism used to steer the snake, but the way that the controller uses the model of the interaction to inform the steering behavior.

6.3 Our Approach - Anticipatory Control

Previous work in a Chronos physics simulation shows a linear correlation between scattering duration and contact duration between the head of the robot and the post [2]. Inspired by the anticipatory controller developed in [13], we propose using this correlation to develop a controller for the snake robot. In [13], a bioinspired hexapod robot collides with a boulder in granular media. Contact sensing panels on the boulder determine where the robot makes contact with the obstacle. When the robot is given an open loop trajectory, the scattering angle between the robot and the boulder can be reasonably well predicted by looking at which panel of the boulder made contact with the robot. Once this relationship is developed experimentally, it is used to correct the robots trajectory with a simple anticipatory controller. Based on the location of contact between the robot and the boulder, the controller anticipates the scattering angle, and steers to correct for it by using a tail as a rudder.

The obvious drawback to this scheme, however, is that the robot is using sensing on the boulder to inform its controller which is not an accurate depiction of how living or robotic systems move through environments. Conversely, for our system, we rely only on onboard sensing to inform the controller. Therefore, in order to reliably measure contact between the head of the snake robot and the vertical, posts we developed a contact sensing circuit at the head of the robot. Inspired by this idea of using a interaction model to perform control, we propose using this contact duration to inform a control strategy. Using an amplitude modulation steering mechanism first developed by [14], we propose using the interaction model to predict the scattering angle resulting from the collision, and steer to correct it.

The remainder of this thesis is divided into the following chapters. Chapter 7 provides an overview of our robot system, as well as the contact sensing circuit. Chapter 8 develops the theory and experimental testing needed to develop the anticipatory controller. Chapter 9 discusses the results of testing this controller, and hypothesizes how the controller can be improved in the future. Finally, chapter 10 presents and tests possible modications to improve the performance of the controller.

CHAPTER 7
SNAKE ROBOT OVERVIEW

7.1 Hardware

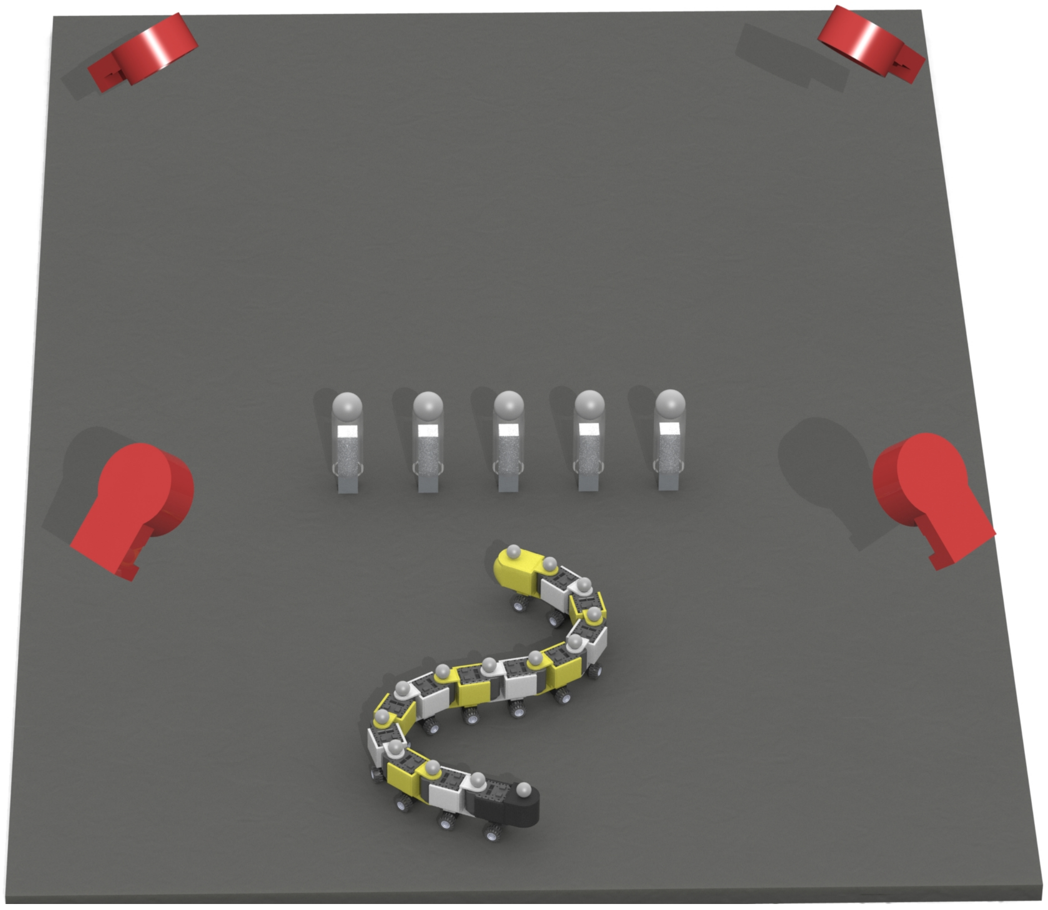


Figure 7.1: Early rendering of an overhead view of the snake robot setup in a typical multipost setup (courtesy of Zachary Goddard). All posts have a radius of 2.3 cm and the spacing between posts is 6.9 cm.

Our snake robot is composed of twelve AX-12A Dynamixel motors joined together using 3D printed brackets. While the body brackets are the same throughout the body of the snake robot, two specialized brackets were built for the head and tail of the robot. The head module is rounded so that there is a hemispherical surface which first contacts the post, and the back of the tail module is rounded in a similar fashion. Each bracket connects to passive lego wheels on the base of the brackets which provide the direction dependent friction necessary for the robot to locomote.

Ax-12A motors were selected due to their low price point and high usability. At just under 50 dollars per motor, these motors are near the lower end of the dynamixel line, but still provide plenty of torque for our application (1.5 N m stall torque at 12 V). Since this work focuses primarily on developing new software for an existing system, only a high level description of the mechanical design of the snake is provided as a reference to the reader to better understand the experiments. However, key modifications to the robot to improve integration with the gantry system and provide the sensing necessary for the controller will be provided in the subsequent subsections to show how these new components achieved these objectives.

7.1.1 Gantry Interface

The first set of modifications to the original snake robot sought to improve tracking of the the snake by the Optitrack camera system. As mentioned in Chapter 3, getting quality Optitrack data was one on the greatest challenges to developing a robust automated gantry system. We attempted to address this by adding red 3D printed columns to lift the markers off of the body of the snake and hopefully improve tracking of individual markers. The individual marker data, however, still was subject to markers getting swapped or dropped, so a more robust way of identifying the snake's final orientation was needed. The white 3D printed brackets were designed with this problem in mind. By applying reflective tape on top of the magnetic plate, we were able to create two geometrically distinct rigid bodies on the snake at the exact points where the robot will be picked up. Additionally, these brackets provide the largest possible contact surface between the magnetic plate and the electromagnet without protruding beyond the dimensions of the snake below. Finally, a

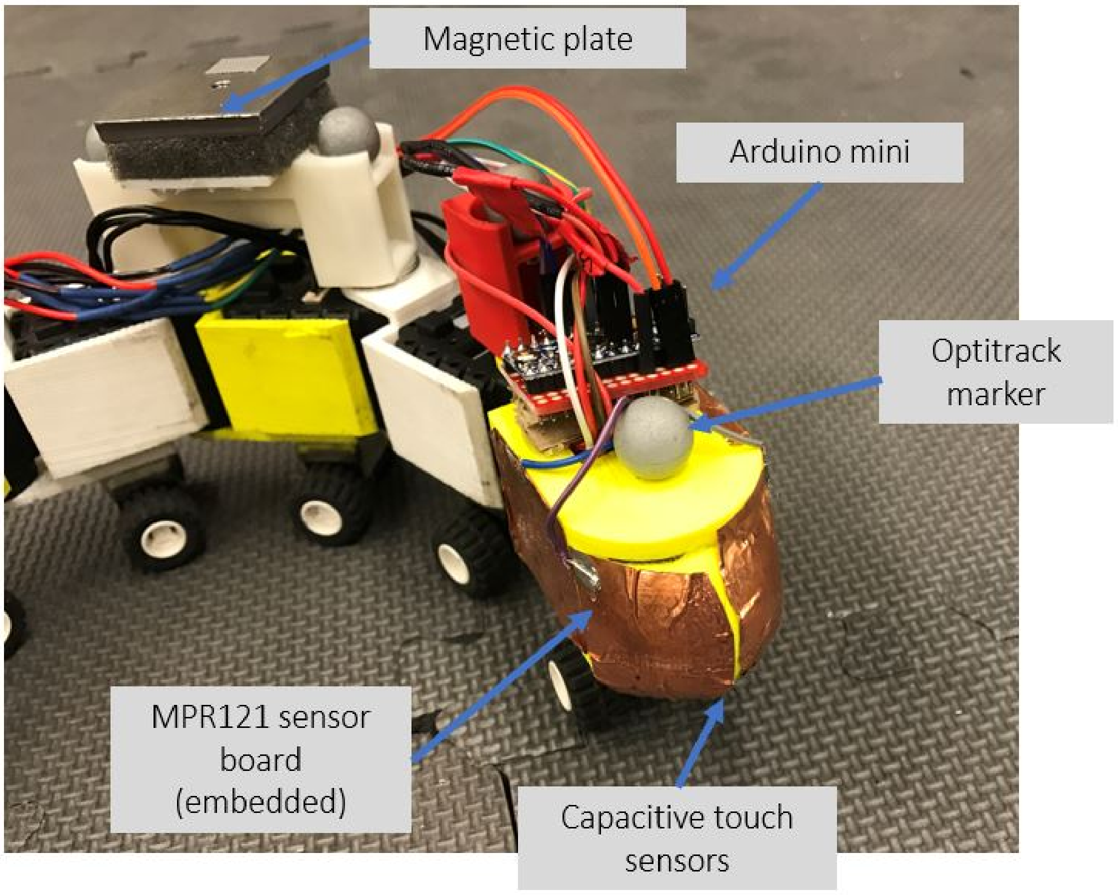


Figure 7.2: This figure highlights two crucial modifications to the snake robot: the magnetic plate bracket used to pick up the robot with the gantry and the contact sensing circuit used to measure contact location and duration between the robot and the posts.

foam piece is placed in between the bracket and the plate, which ensures the best possible contact between the electromagnets and the magnetic plates. Fig. 7.2 shows an assembled bracket on the snake. Note that this bracket uses a single piece of reflective tape to create the three reference points necessary to define a rigid body, whereas the second bracket (not shown) uses four references points to prevent confusion between the two brackets.

7.1.2 Tail Module

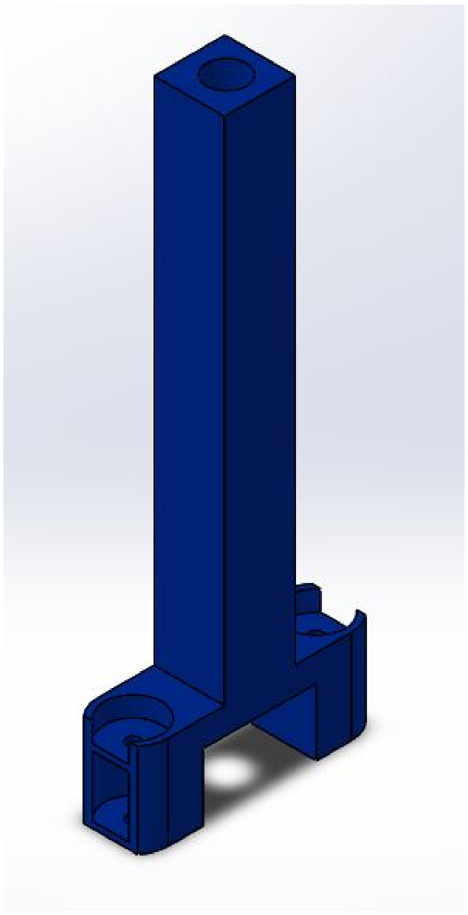


Figure 7.3: CAD design of the snake tail, used to prevent snake wires from dragging behind the snake.

The tail module shown in Fig. 7.3 was designed in an iterative process with wire management in mind. During hand placed experiments, wires would frequently get caught on the posts during experiments, so it was necessary to have someone around to prevent this from happening while the snake was moving. Additionally, we were extremely cognizant of the fact that the trailing wires of the snake could potentially effect locomotion of the snake, or get tangled with the gantry arm when the snake was being picked up. To address these concerns, the tail was designed to keep the snake wires from dragging behind the snake during locomotion. The tail extends directly above the last module of the snake to prevent contact with the posts during experiments, and the wires are threaded through a hole in the center of the piece. These wires hang down directly from the gantry system which moves alongside the snake so that the snake wires are always hanging directly above the robot to prevent them from being caught. Note that the wires are not directly fixed to the gantry arm above, but have some slack and room to translate so that the snake wires are never pulling on the robot from above. Ensuring that the wires don't pull on the snake was an important step to prevent damaging the robot and to produce dependable data in our experiments.

7.1.3 Contact Sensing Circuit

In order to collect experimental data on the interaction between the head of the robot and the environment, developing a contact sensing circuit at the head of the robot was crucial. Previous work in simulation [2] has shown that the majority of the scattering behavior observed in the system can be understood by looking at the interaction between the head of the robot and the post. Therefore, this contact sensing circuit will be important for understanding the behavior of the snake robot at contact. Furthermore, this contact sensing circuit is also an essential part of the controller by allowing us to determine the location and duration of contact between the head of the robot and the posts. The wiring for the contact sensing circuit is shown in Fig. 7.4, and the actual implementation on the robot can be seen in Fig. 7.2. Four capacitive touch sensing panels are connected to an MPR121 contact sensing circuit board from Adafruit using I2C serial communication. This contact sensing board connects to an Arduino mini where a simple code compares the capacitance value for each panel to a threshold to determine if contact has been made. We decided to use the mini because then both the MPR121 and the Arduino mini could be mounted on the head of the robot. We found that having long cables running between the contact panels and the MPR121 was not effective because they affected the capacitance measurements. To implement these boards on the robot, a new hollow head module was designed so that the

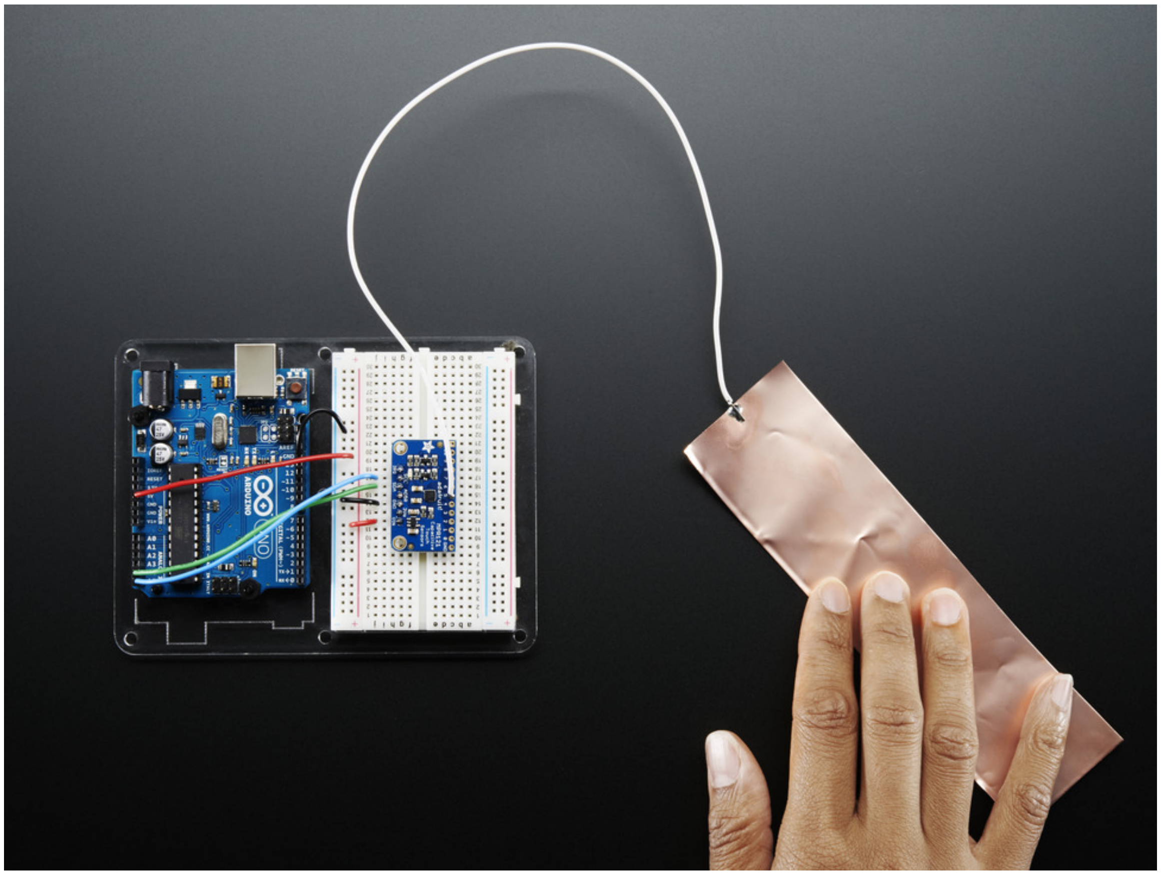


Figure 7.4: The MPR121 from Adafruit provides an easy way to integrate capacitive sensing onto the head of the robot. This figure from the Adafruit website illustrates how the board is interfaced with an Arduino Uno and a capacitive touch sensor to show how the board should be wired, and the actual implementation is shown in Fig. 7.2

MPR121 could be embedded within the head module of the robot, and the Arduino mini could be mounted directly on top of the head of the robot.

7.2 Software

The control of the snake is easily integrated with the rest of the gantry code as shown in Fig. 3.1. The snake code can easily be divided into two main components: sensing via the contact sensing circuit and actuation via the AX-12A motors.

7.2.1 Actuation

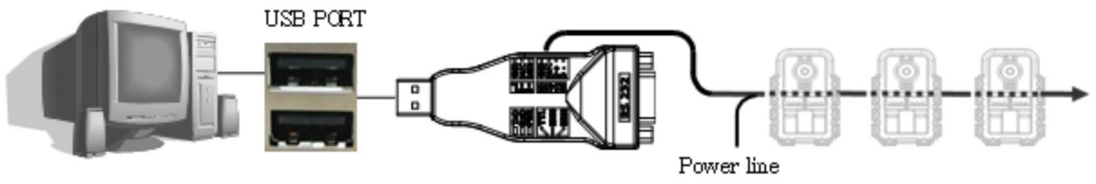


Figure 7.5: A USB2Dynamixel by Robotis was used to command the AX-12A motors from a desktop computer. Image from the Robotis documentation.

The Robotis SDK software was used to control the Ax-12A motors on the snake. This software package allows the motors to be controlled using robust, well documented libraries on the Dynamixel SDK Github page, so it is used frequently to control many robots in the lab. We use the C++ package because it allows for easy integration with the other components of the gantry system as mentioned in Chapter 3. The commands are sent from the computer to the robot with the USB2Dynamixel, which plugs in to a USB port in the desktop computer. The USB2Dynamixel outputs to three wires (power, ground, signal) which connect to the motors using transistor to transistor level serial communication. The power line is directly connected to a 12 V power supply, and the motors draw less than an amp of current during normal operation. The motors are daisy chained together using short three-pronged connector wires, and assigned their own ID. Motor registers such as motor ID, torque, position, and more can be directly and easily modified with the Dynamixel wizard, or in the code using the SDK.

7.2.2 Sensing

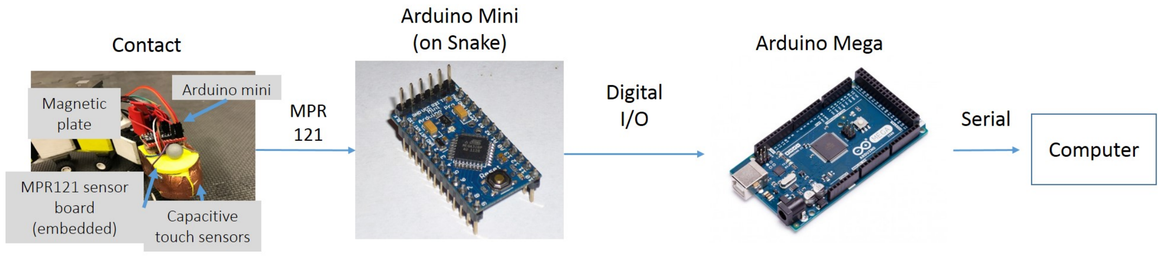


Figure 7.6: This figure shows how data is transferred from the contact sensing circuit on the head of the robot to the Arduino Mini, to the Arduino Mega, and finally to the computer.

In order to implement the controller, the actuation of the motors must be combined with sensing data from the contact sensing circuit. Fig. 7.6 shows how data is transferred from the contact sensing panels (7.4) to the main program. Starting with the Arduino mini, a simple sketch compares the analog value for the capacitance of each panel with a threshold to determine if contact has been made. Then, for digital output pins, one corresponding to each of the panels, are set to either HIGH or LOW depending on if contact is detected. This signal is sent to the Arduino Mega over four wires, and input as an GPIO digital input. This data can be requested in the Arduino Mega sketch, again relying on the serial interrupts. Writing the command 'data' to the serial monitor tells the the Mega to read these four pins and determine the contact state. In the main snake loop, this information is requested at every time step, and is written to a CSV file along with the Optitrack data. In addition, this information can be used in the loop to determine when contact has been made between the snake and the post and trigger the controller. In order to prevent a random spike from activating the controller, a minimum of three contact points is necessary for the controller to activate. We have seen in post-processing the contact sensor data that this a good threshold for determining what is a real contact and what is just a spike in the readings.

7.3 Serpenoid Curve

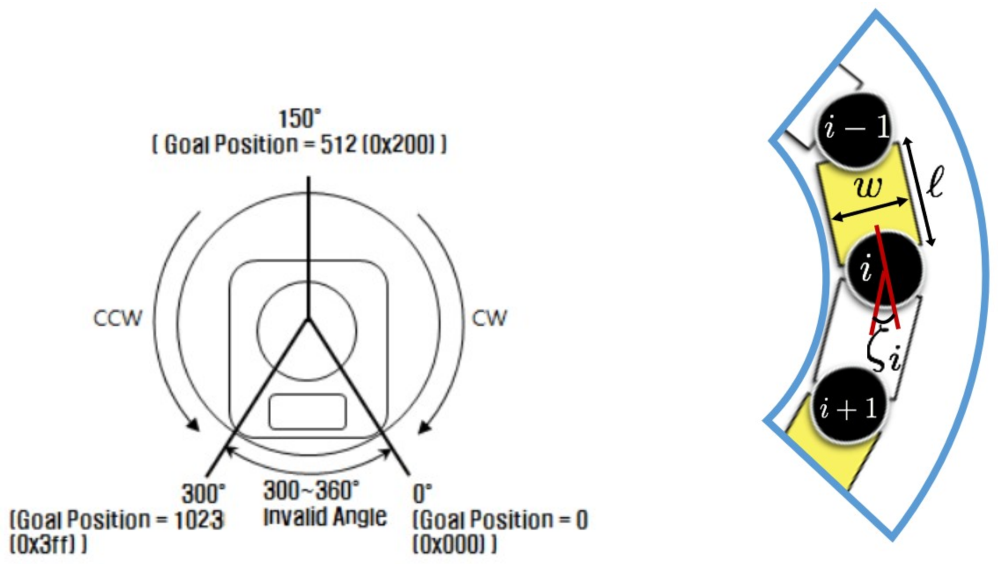

150°
[Goal Position = 512 (0x200)]
CCW
CW
300°
[Goal Position = 1023 (0x3ff)]
300~360°
Invalid Angle
0°
[Goal Position = 0 (0x000)]
$i-1$
w
ℓ
i
ζ_i
$i+1$

Figure 7.7: The angle of each segment, $\zeta_N(t))$, is defined according to the serpenoid equation, where the angle is defined from the center position. The lefthand figure (from Robotis documentation) shows the relationship between the physical angle and the analog input value for the AX-12A motors, and the righthand figure (from [2]) shows these angles defined along the body of the snake robot.

The theory for how to generate the open loop trajectory is well established in the literature. We consider an N-jointed snake robot, where we specify the joint angles $\zeta(t) = (\zeta_1(t), ..., \zeta_N(t)) \in \mathbb{R}^N$ for $1 \leq i \leq N$ at some time t. Hirose et al. [1] described a family of curves, the serpenoid curves, that can parameterize the shape of a snake robot during locomotion under a variety of gaits. In this work, we focus on two dimensional locomotion for which the corresponding open-loop serpenoid curve is given by [1, 11, 15]:

$$\zeta_i(t) = \zeta_0 + A \sin(\omega_S s_i - \omega_T t), \tag{7.1}$$

where ζ_0 is a constant curvature offset, s_i is the distance from the head of the snake to the joint i along the snake's body, A, ω_S and ω_T respectively are the amplitude, the spatial and temporal frequencies of the serpenoid curve. For our robot, we implement the Hirose equation to find that the position of each motor is given by:

$$\zeta_i(t) = A\sin\left(\frac{2\pi i}{N} - 2\pi f t\right), \tag{7.2}$$

where A is 40 degrees, i is the motor number, N is the total number of motors on the snake (12), and f is 0.22 Hz. As shown in Fig. 7.7, the angle ζ is defined from the center of the motor so that if all motors were commanded to have an angle of 0, the robot would be in a straight line. Therefore, this commanded angle needs to be shifted and scaled to match the command input range of the AX-12As where a commanded value of 0 to 1023 maps to an output angle of 0 to 300 as defined in Fig. 7.7.

7.4 Modeling Robot Obstacle Interactions

7.4.1 Straight Line Testing

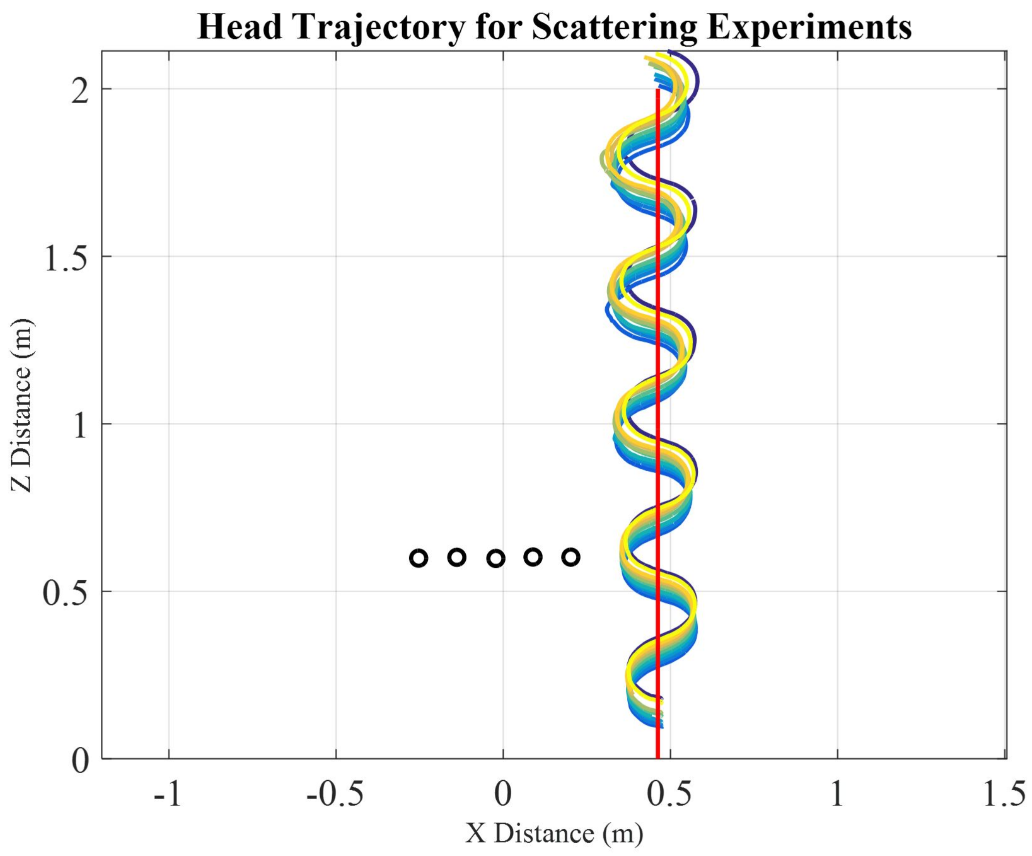


Figure 7.8: Straight line experiments were run off to the side of the posts to verify that the open loop controller resulted in a straight line trajectory of the center of mass of the robot.

In order to validate that programming these angles results in the desired open loop behavior, we tested the robot off to the side of the posts. Fig. 7.8 shows an example of a successful data set for this kind of straight line callibration. For the ten experiments shown, the mean angle is -0.5 degrees with a standard deviation of 1.5 degrees. Archiving this level of reproduciblity was a fairly challenging problem. Issues with the snake and servo motor

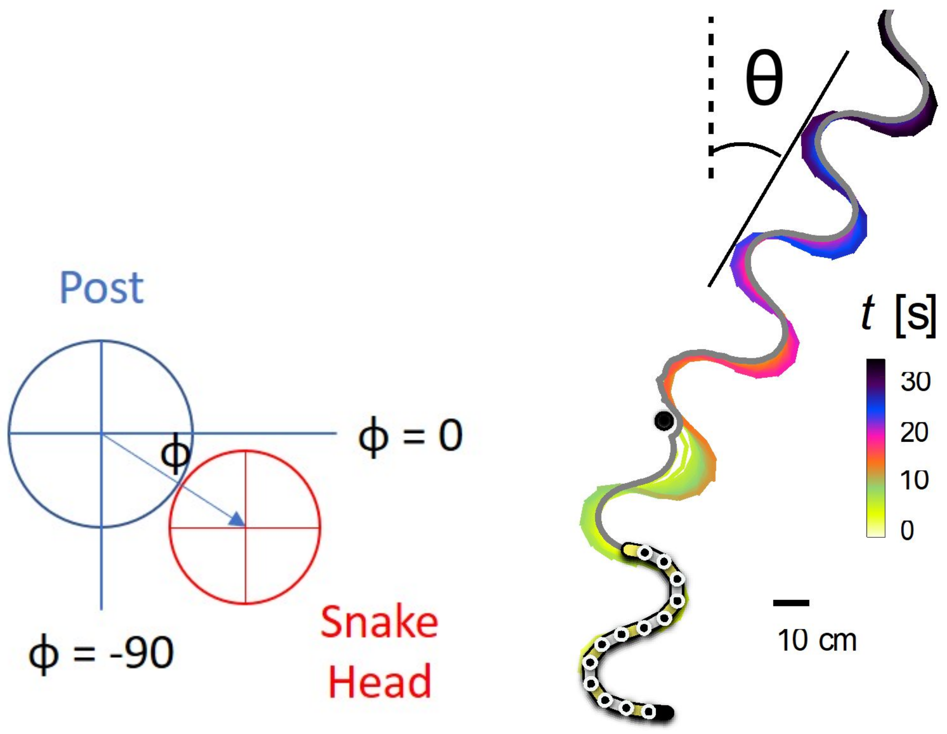


Figure 7.9: Definition of the important parameters to our model of the robot-obstacle interaction. The lefthand figure shows the head contact angle ϕ between the post and the head of the robot, and righthand figure (courtesy of Jennifer Rieser) shows the scattering angle θ determined from the head trajectory of the robot.

caused the trajectories to either be at an angle or curved. In particular, we observed that the straight line angle changed over time after many experiments. This was caused by the fact that the screw in the servo motor controlling the gantry orientation loosened over many experiments. Adding loctite to the screw fixed this issue, but we also noticed some curvature in the trajectory of the snake. This curvature was caused by wear on the wheels, which changed the nature of the direction dependent friction for a few segments of the body. Replacing all of the wheels on the snake resulted in the straight line behavior seen in the figure.

7.4.2 Sampling Initial Conditions

Finally, we needed a way to model and understand the collisions between the snake robot and the posts. We observed that collisions between the snake and posts caused the robot to change its orientation and follow a new straight line trajectory. We call this angle between the original heading of the robot and the new trajectory the scattering angle, which we determine from the Optitrack data in post processing. Fig. 7.9-B shows the sign convention used for the scattering angle measurement. Once the robot has passed a threshold distance past the posts in the Z direction, we fit two lines the maximum and minimum values of the waveform, then average the slope of these two lines. From the slope of the line, we can then determine the angle from the global reference frame.

We model this interaction between the posts and the robot by measuring scattering angle as a function of two contact states: head contact angle ϕ and the phase η. The head contact angle is defined according to the sign convention presented in Fig. 7.9-A, and tells us the location on the post where the snake first makes contact. The phase of the snake is defined by equation 7.3 and tells us what part of the cycle the snake is in when it makes contact.

$$\eta = \text{atan}\left(\frac{\text{dx}}{wx}\right) \tag{7.3}$$

In order to systematically sample all geometrically allowable contact states, we systematically sample a box of head position initial conditions. For a single post, the size of this box is dictated by the dimensions of the robot. This centered 20 by 40 cm box is randomly sampled by the gantry system for around 500 experiments. In the multiple post setup, the dimension of this box is dictated by the length of the robot and the center to center distance between the posts. This sampling method allows us to make a fair comparison between open loop and controller, as well as to fully understand the different contact conditions that arise in the experiments. Chapter 8 will further explain how this control scheme was developed, then the results from these tests will be presented and discussed in chapter 9.

CHAPTER 8
CONTROLLER DEVELOPMENT

8.1 Controller Overview

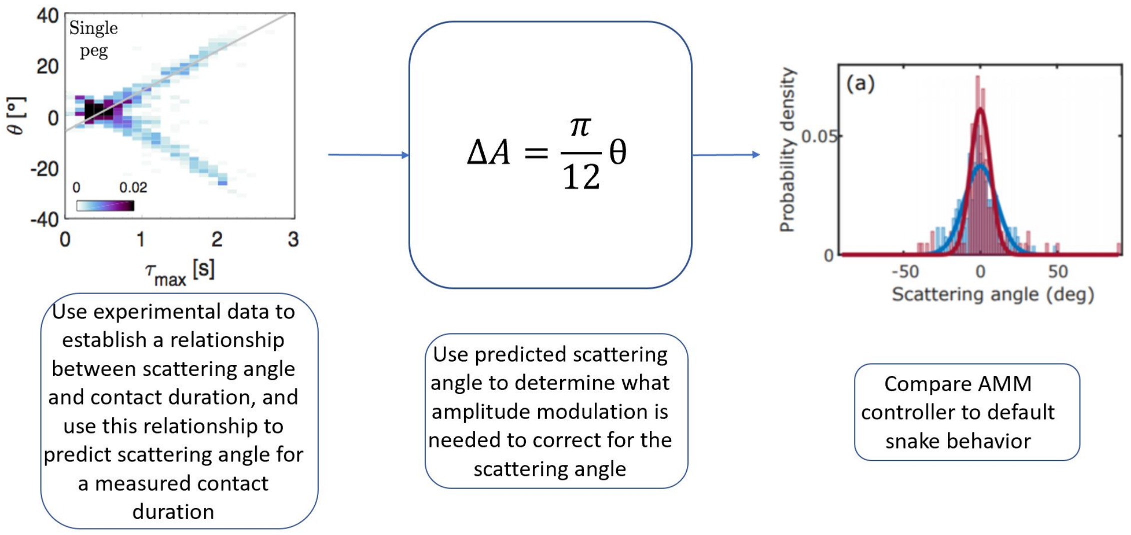

Single peg
40
20
0
-20
-40
θ [°]
0 0.02
0 1 2 3
τ_{max} [s]
Use experimental data to establish a relationship between scattering angle and contact duration, and use this relationship to predict scattering angle for a measured contact duration
$\Delta A = \frac{\pi}{12}\theta$
Use predicted scattering angle to determine what amplitude modulation is needed to correct for the scattering angle
(a)
Probability density
0.05
0
-50 0 50
Scattering angle (deg)
Compare AMM controller to default snake behavior

Figure 8.1: Overview of the anticipatory controller. Interactions between the robot and the post are studied in simulation and experiment. A linear correlation between scattering angle and head contact duration is used to determine the amplitude change required to correct for the scattering. The robot uses AMM steering to correct for scattering after contact, and the resulting distributions are compared for the open loop snake and controller.

Previous work in the Chrono simulation environment [2], [16] demonstrates the existence of a linear relationship between the scattering angle and the duration of contact between the head and the post shown in the first panel of Fig 8.1. Given this relationship, this section describes how an anticipatory control scheme was developed by building on the Amplitude Modulation Method (AMM) [14]. At a high level, the controller uses this relationship to anticipate the scattering angle based on the recorded contact duration, and implements the amplitude modulation method to correct for it. The resulting distributions for the open loop trajectories and the controller are compared for the single and multiple post setups in section 9.

8.2 Controller Implementation

This section addresses the details as to how the controller is implemented on the robot. Section 8.2.1 explains the method we use to predict the magnitude of the scattering angle by summarizing a result from [2]. Section 8.2.2 describes the theory behind how an amplitude change on the snake results in a change in heading. Section 8.2.3 takes this theory and validates it with systematic testing on the robot. Finally, section 8.2.4 discusses a method to predict the scattering direction of the robot using only the available onboard sensing.

8.2.1 Predicting Steering Magnitude

We hypothesize that the relationship between scattering angle and contact duration (Fig. 8.2) can be used to control the robot to reduce the scattering behavior for larger scattering angles. Note that data points with a contact duration of about .5 seconds and less, this linear relationship no longer holds. In this region, there is little the controller can do to correct for the steering behavior so scattering within plus or minus five degrees in this region is expected with or without the controller. For longer durations, however, we should be able to use a linear regression of this data in order to determine how much steering is necessary to correct the orientation of the robot. This relationship is verified experimentally in 9, and implemented in the controller with the help of the Amplitude Method for steering.

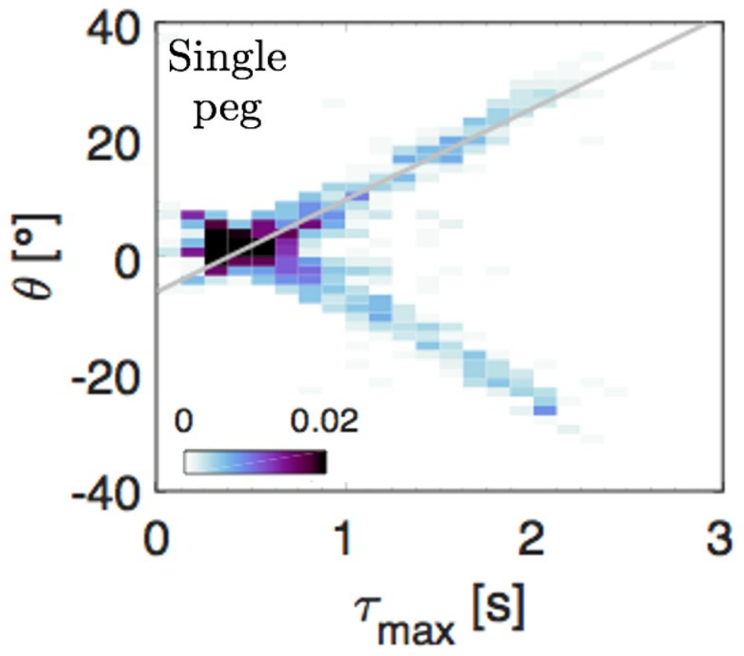


Figure 8.2: Simulation plot of scattering angle vs contact duration taken from [2].

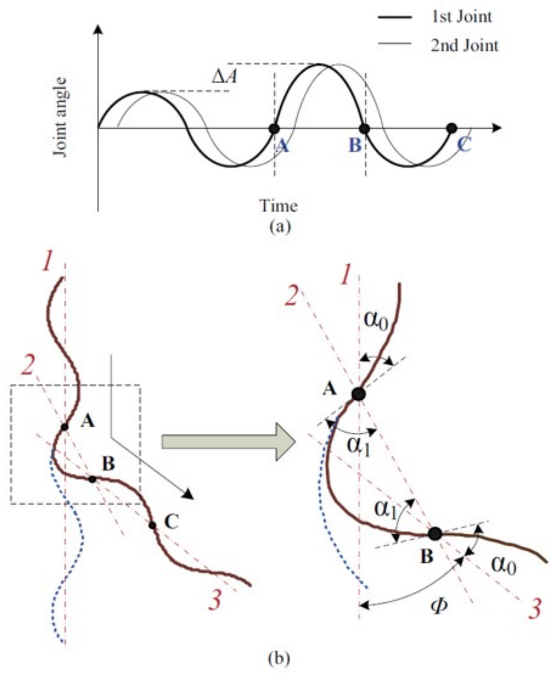


Figure 8.3: Wu et al provides a framework for steering a snake like robot - this figure is first presented in [14].

8.2.2 Steering Implementation: Amplitude Modulation Method

According to [14], a snake robot can be steered by increasing or decreasing the amplitude of the sine wave in equation 7.2 at what is known as a point of zero curvature along the snakes backbone, i.e. where the sine wave and the resulting joint angle are equivalently zero. Wu and Ma outline the following steps to achieve this steering. First the robot moves along straight line (labeled 1 in Fig. 8.3-B) until it reaches point A. At point A, the amplitude of the first motor is increased, so that the snake follows trajectory 2. In the joint angle vs time plot, (Fig. 8.3-A) point A is what is known as a point of zero curvature, and is important because it is the beginning of a new undulation so the snake joint angle is zero. The amplitude change from the first motor is passed down the body of the snake following the offset in the snake equation, as shown in 8.3-A. Once the snake completes half a cycle, the amplitude of A is changed back to normal, and this amplitude change is passed down the body once again. The result of the controller is that the snake now moves along path 3, and the resulting steering angle Φ is directly related to the amplitude change ΔA as shown in Equation 8.1.

$$\Phi = \frac{2N}{\omega_S}\Delta A \tag{8.1}$$

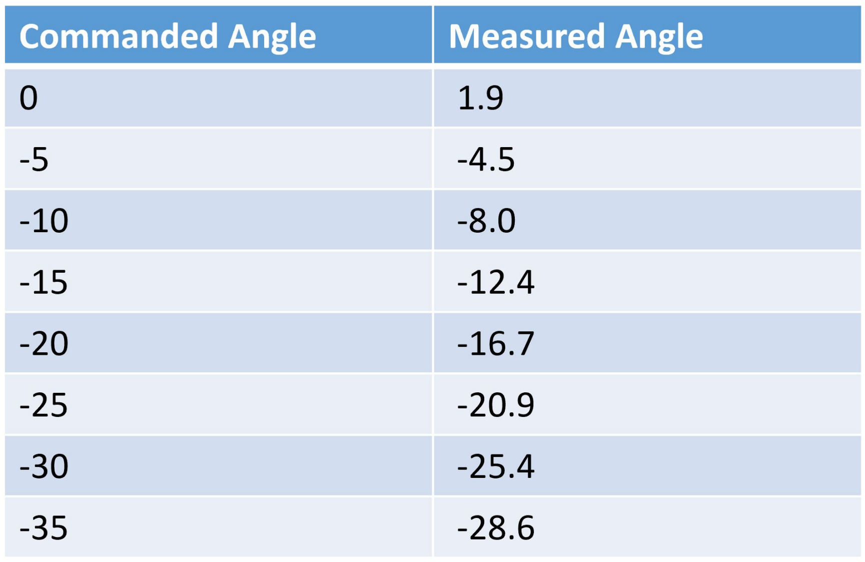
where ω_S is the spatial frequency of the serpenoid curve, and N the number of joints on the robot. This equation provides the theory behind how we can steer to correct for scattering, but we still need to validate that this approach will actually work on our robot.

8.2.3 Steering Implementation: AMM steering testing

To test this steering scheme on the robot, first the joint angle vs time plot from Fig. 8.2.2-A for each joint was reproduced in a Matlab script to verify that the method was implemented correctly. In this testing script, an array of angles is given as an input, then equation 8.1 is used to calculate the appropriate amplitude change required to steer at the given angle.

This amplitude change is then implemented on the head joint after the first full undulation, and passed down the body by making the amplitude change at the correct time. At the next point of zero curvature, the amplitude is changed back to the normal value.

Once we could reproduce Fig. 8.2.2-A and generate Fig. 8.11, we were confident that the code was working correctly so it was implemented on the robot. Using a similar approach, the commanded angles in column 1 of table 8.4 are given as an input for the controller on the robot running off to the side of any obstacles. After implementing the AMM steering (again at the first undulation), the resulting angle caused by the steering is measured by the Optitrack cameras overhead from the head trajectories shown in Fig. 8.5.



Commanded Angle	Measured Angle
0	1.9
-5	-4.5
-10	-8.0
-15	-12.4
-20	-16.7
-25	-20.9
-30	-25.4
-35	-28.6

Figure 8.4: Angles commanded to the robot and measured via the Optitrack system (in degrees). Note that error increases at higher angles.

These measured angles show a systematic error between the measured angle and the actual angle. This is not surprising since nothing in the Wu paper addresses our particular robot. Therefore, equation 8.1 is modified by adding a calibrated constant, α, to correct for

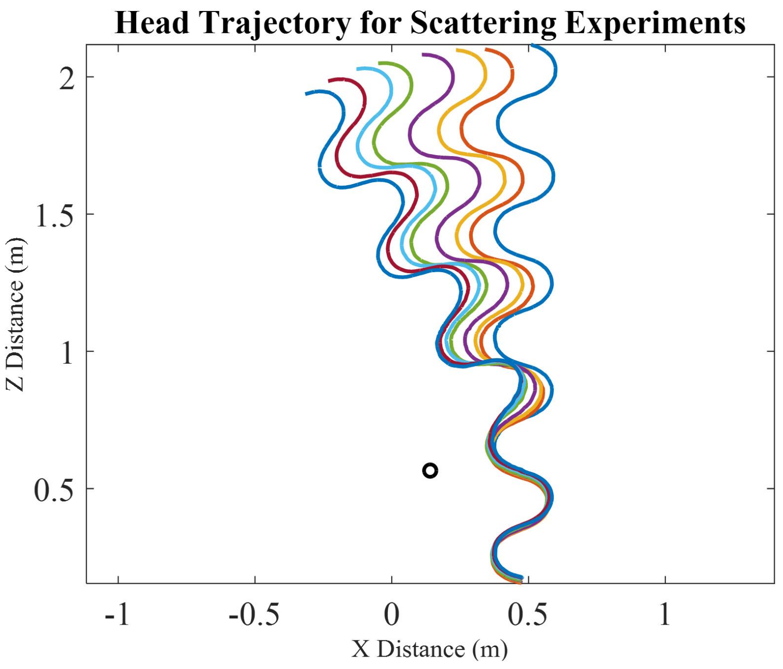


Figure 8.5: The resulting head trajectories of implementing the AMM on the robot to steer to the angles in table 8.4.

this systematic error. The modified steering equations now reads:

$$\Phi = \frac{2N\alpha}{\omega_S}\Delta A \tag{8.2}$$

where the calibrated constant α is specific to our experimental setup. Fig. 8.6 shows how adding this constant improves the tracking of commanded angles by plotting measured angle vs commanded angle. For perfect tracking, we would expect the points to fall along the line y = x, shown in black. The red X markers represent the experimental data points before implementing the calibration constant, and the blue X markers represent the experimental data points with the correction. While there is still some small error between the commanded value and measured value for these points, it is random. This figure shows that we can accurately predict the correct angle to steer, the amplitude modulation method gives us the ability to achieve that angle on the robot accurately.

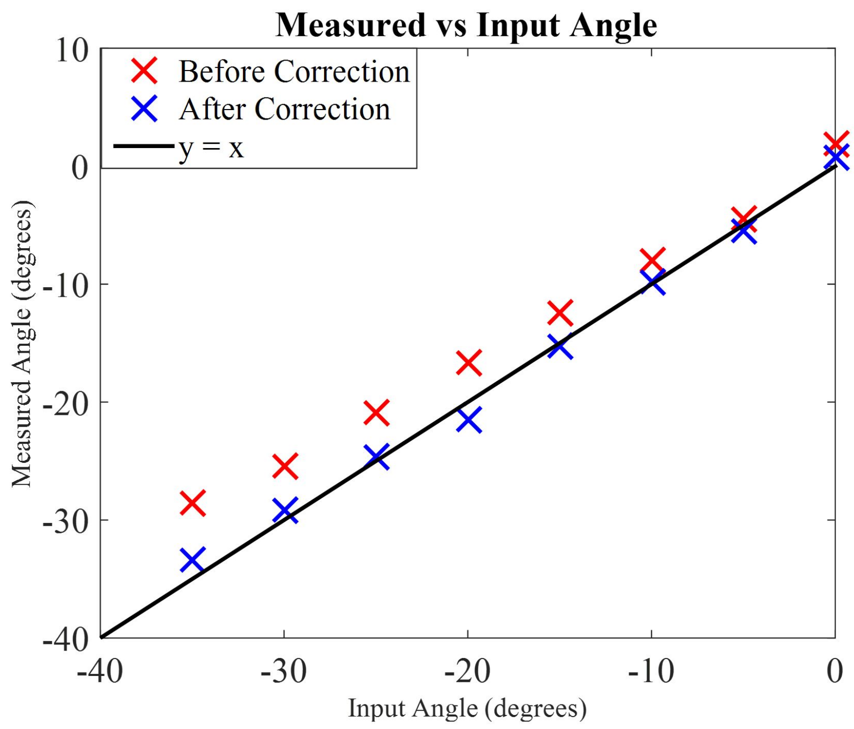


Figure 8.6: Using the experimentally determined correction constant results in better steering performance for large angles.

8.2.4 Predicting Scattering Direction

Knowing the magnitude of the scattering angle, however, is not sufficient for correcting scattering. Depending on the state of contact between the head of the robot and the post, the robot may scatter to the left or the right. Predicting the scattering direction based on the available on board sensing, however, is not a trivial problem. The work in simulation [2] again provides us with a useful starting point. Fig. 8.7 shows that contact on the right side of the post generally scatters in the positive direction and contact on the left side of the post generally scatters in the negative direction. Therefore, we hypothesize that contact with the right front contact panel on the head of the snake 9.3 will correlate strongly with contact on the left side of the post. From 8.7, we know that contact on the left side of the post will most likely will result in scattering in the positive direction. Therefore, we command

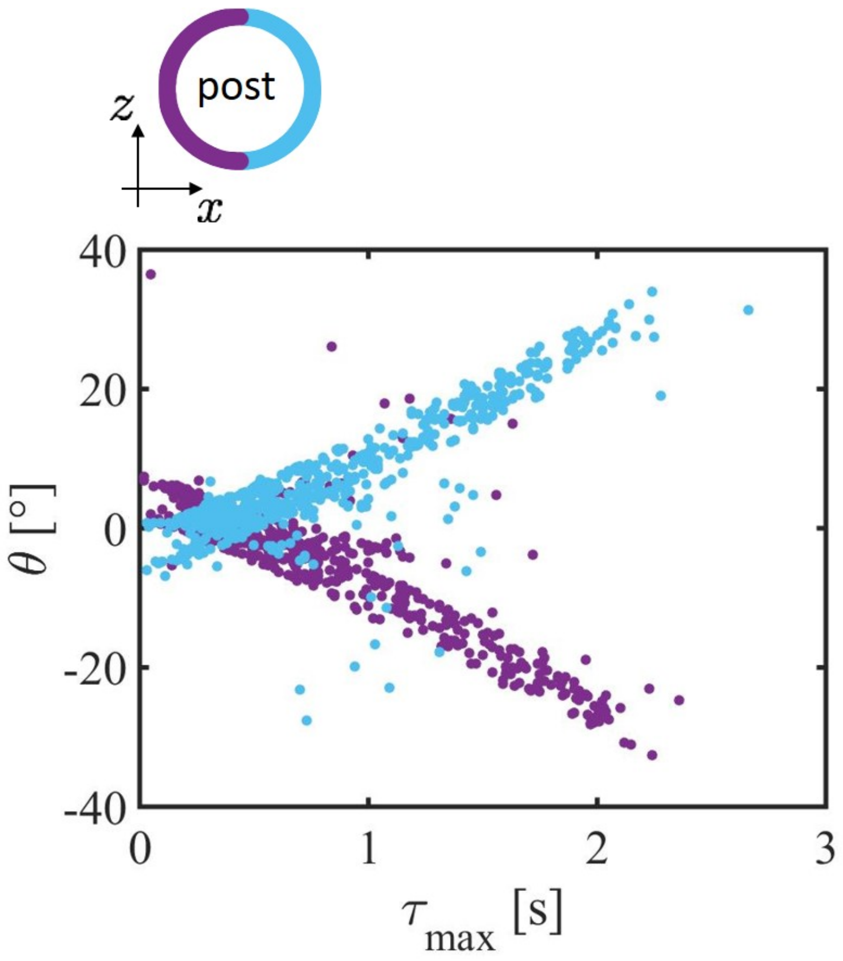


Figure 8.7: Contact location on the post is a fairly reliable indicator of the direction that the robot will scatter. Reproduced from [2].

the snake to steer in the negative direction to counteract this predicted scattering. Still, the correlation in 8.7 is not true for all of the simulation data. This is an important observation because predicting the wrong scattering direction will result in the controller performing worse than the open loop case. Based on the simulation results, we hypothesized that looking at which contact panels make contact with the post can be a good predictor of the direction of the resulting scatter. In this first iteration of the controller, scattering direction is predicted based on the first state of contact between the robot and the obstacle. However, the analysis in Chapter 9 will show that the last contact state gives a better prediction of scattering direction.

8.2.5 Steering Direction Implementation

One interesting observation from these early experiments is what the amplitude modulation looks like on the body. When we were running these tests to develop the accuracy of the angle tracking, the amplitude modulation was always implemented after the first full undulation of the snake. We observed that, for this particular point of zero curvature, in order to steer in the positive direction a negative amplitude change is implemented. This causes a local decrease in curvature, and the snake lengthens in the direction of locomotion which we call expanding (shown in Fig. 8.8-A). Conversely, to steer in the negative direction, a positive amplitude change is implemented. This causes a local increase of curvature, and the snake shortens in the direction of locomotion which we call contraction (shown in Fig. 8.8-B).

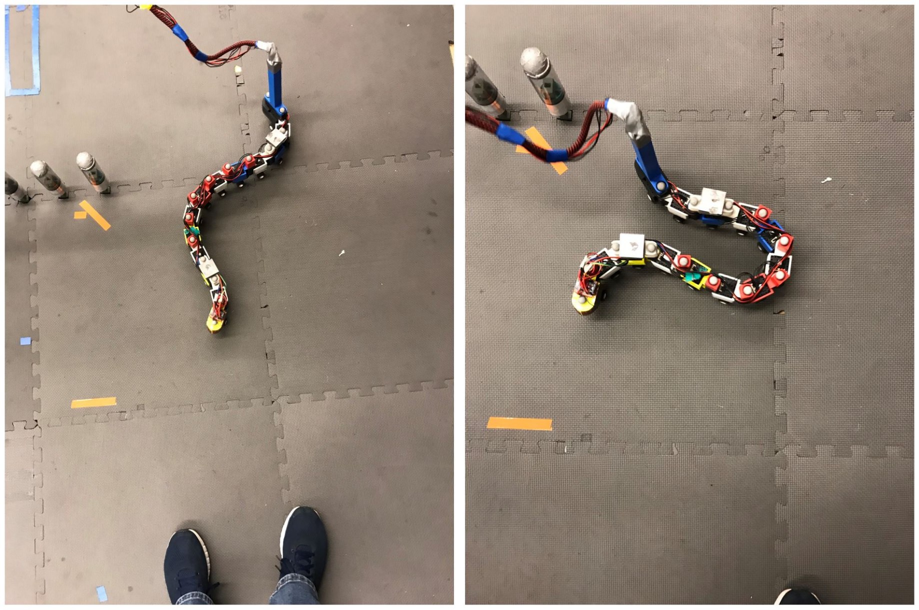


Figure 8.8: The amplitude change causes the snake to expand or contract when viewed from above. Depending on which point of zero curvature the steering activates, either shape can be used to steer in the positive or negative direction.

However, we can see from 8.3 that there are also points of zero curvature at half undulations. In order to take advantage of additional points of zero curvature, we modified the controller so that it can steer at these half cycle as well as full cycle points of zero curvature. Another interesting observation is that the steering behavior at the half cycle is actually reversed. This behavior is likely due to the fact that at the half points of zero curvature, the global angle of the head joint is pointing in the opposite direction as when then robot is at full cycle points of zero curvature. Fig. 8.9 shows how if you want to steer in the positive direction and you make begin implementation at a full cycle point of zero curvature, you should expand the snake. Conversely, if you wanted to steer at the half cycle point, then you would want to contract the snake. Therefore, we can achieve steering at next immediate point of zero curvature by changing the sign of the amplitude change for the half cycle points of zero curvature, which is what is implemented in the first iteration of the controller. Variations on this steering decision are considered in chapter 9.

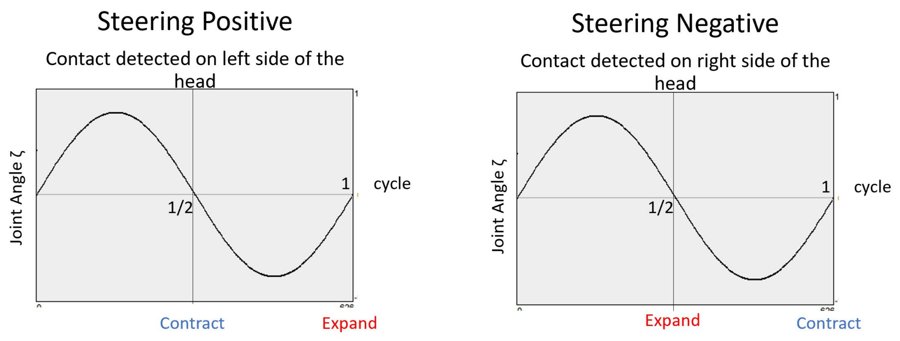


Figure 8.9: The sign of the amplitude change required to steer in a desired direction depends on which point of zero curvature the steering method is activated.

8.3 Controller Summary

The following steps summarize the anticipatory controller:

1. Detect contact and record contact location and duration

2. Predict scattering direction based on location of contact on the head of the robot

3. Predict scattering angle based on the linear correlation between scattering angle and head contact duration

4. Determine the appropriate amplitude change in order to correct for the predicted equation using equation 8.2.

5. Implement amplitude modulation method with the correct sign at the first point of zero curvature after contact

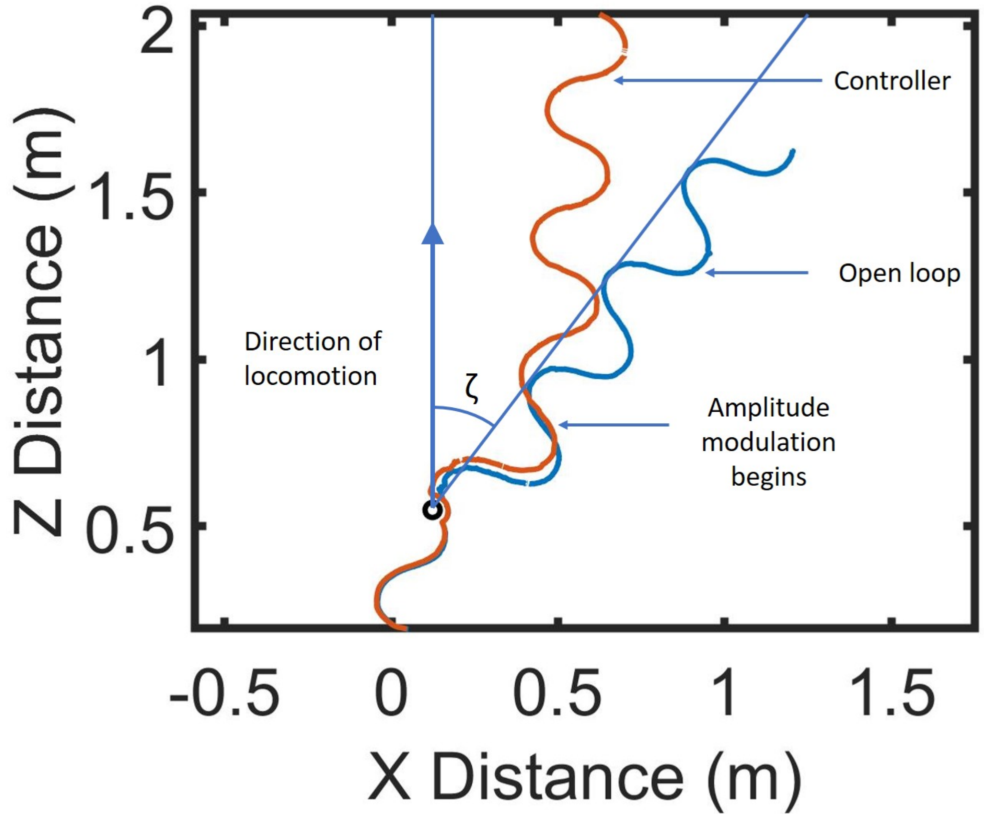


Figure 8.10: Two experiments with the same initial conditions are compared, one with the anticipatory controller and one open loop.

Note that in step three, only the front two panels are used to measure contact duration in order to be consistent with the implementation in simulation. Figure 8.10 illustrates the controller in action for a typical single peg experiment. The open loop head trajectory (blue) results in a fairly large scattering angle based on the collision with the post. The controller (red), however, is able to correct for a majority of this scattering by implementing this amplitude change at the point of zero curvature. It is worth noting that while the trajectory has translated in the X direction due to contact with the post, this kind of trajectory is typical of the controller. Since we are trying to control the snake's orientation not position, this translation is not an indicator of poor controller performance. This translation, however, is an interesting feature of the controller and worth considering, especially in terms of the modified controllers in Chapter 10. We would expect the size of this translation to be closely tied to how long the controller needs to wait to implement steering between the end of contact and the appropriate point of zero curvature. For this reason, the first iteration of the controller steers immediately at the first available point of zero curvature.

8.4 Anticipatory Controller within the Shape-based framework

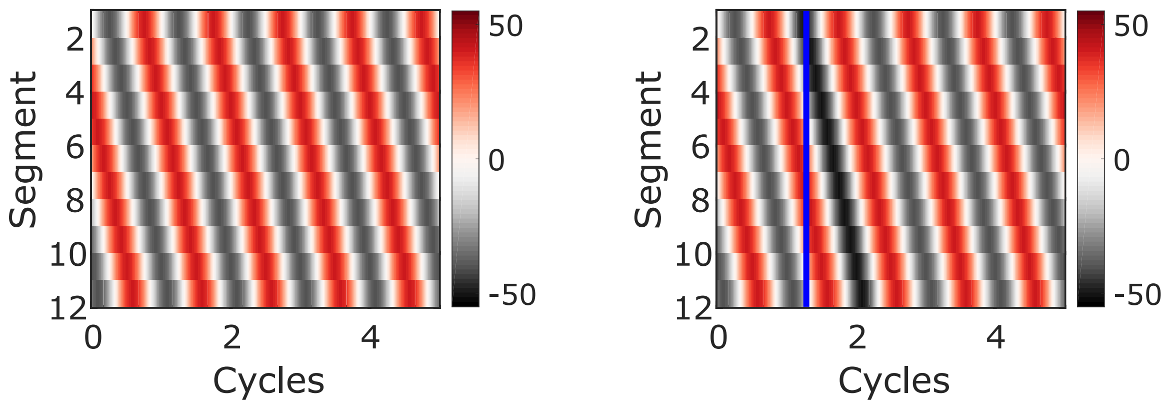


Figure 8.11: Two space-time plots are generated in Matlab, one for the open loop case and one for the anticipatory controller.

In this section, we briefly explain how this controller fits within the shape-based control framework first mentioned in 8.1. Shape-based controllers use a family of shape functions to reduce the dimensionality in systems with such high degrees-of-freedom like a snake robot [17]. The goal of such a shape function $h : \Sigma \rightarrow \mathbb{R}^N$ is to determine $\zeta(t)$ as a function of a small number of control parameters. Those parameters lie in the shape space Σ, which is usually of a lower dimension than $\mathbb{R}^N$. In [17], shape functions $h(A, \omega_S) = \zeta(t)$ are defined in the form of Eq.(7.2) where the amplitude and spatial frequency are allowed to vary, and the other parameters are held constant:

$$\begin{aligned} h : \qquad & \Sigma = \mathbb{R}^2 \mapsto \mathbb{R}^N \\ h_i(A(t), \omega_S(t)) \quad = \quad & \theta_0 + A(t)\sin\left(\omega_S(t)\, s_i - \omega_T\, t\right). \end{aligned} \tag{8.3}$$

Shape based controllers define groups of joints along the snake as windows, which are specified by joints between points of zero curvature along the body. These windows allow for the robot to make local changes in curvature based on local sensing at the head of the robot, and then pass theses changes down the body [11]. The first window encompasses all joints from the robot's head to the first point of zero-curvature along its body. The following windows are defined between two successive points of zero-curvature along the robot's body, and the last window between the last point of zero curvature and the tail. The windows naturally move down the body at the same rate as the serpenoid curve (i.e., $2\pi/\omega_T$). This process enables us to naturally pass information along the robot's body, as the snake locomotes through its environment.

We can see now that the Amplitude Modulation Method presented by [14] is equivalent to discretely changing the curvature of the head window, at the first point in time when the head module is at a zero-curvature point after a collision. By changing the amplitude at the head and passing it down the body, we are effectively modifying the amplitude of the new head window, which begins when the head joint reaches its point of zero curvature and ends when the head joint reaches its next point of zero curvature. At that second time, a new head window is initiated at the head joint, and the amplitude of this new window is reset to its normal value.

This is illustrated nicely by the two spacetime plots in Fig. 8.11 which were generated in Matlab with the script described earlier. The controller plot (right) is clearly distinguishable from the open loop plot (Left) by the dark band that is passed down the body at around two cycles. This dark band represents an amplitude change which is initialized at the end of the first cycle at the head, and then passed down the body.

CHAPTER 9
RESULTS

In order to verify some of the simulation results presented in Chapter 8 as well as evaluate the performance of the controller, we conduct experiments with a single post or an array of evenly spaced vertical posts. Section 9.1 presents the results from the single post experiments. Section 9.2 shows the results for the multiple post setup. Finally, Section 9.3 looks closer at the results to understand what factors impacted the performance of these controllers. In addition to potential shortcomings of the controllers, we will also mention methods which could improve the performance of the controller in these experiments. These insights are used in Chapter 10 to develop and test modifications to the controller with the goal of further reducing the scattering distributions for both the single post and the multiple post setup.

9.1 Single Post Results

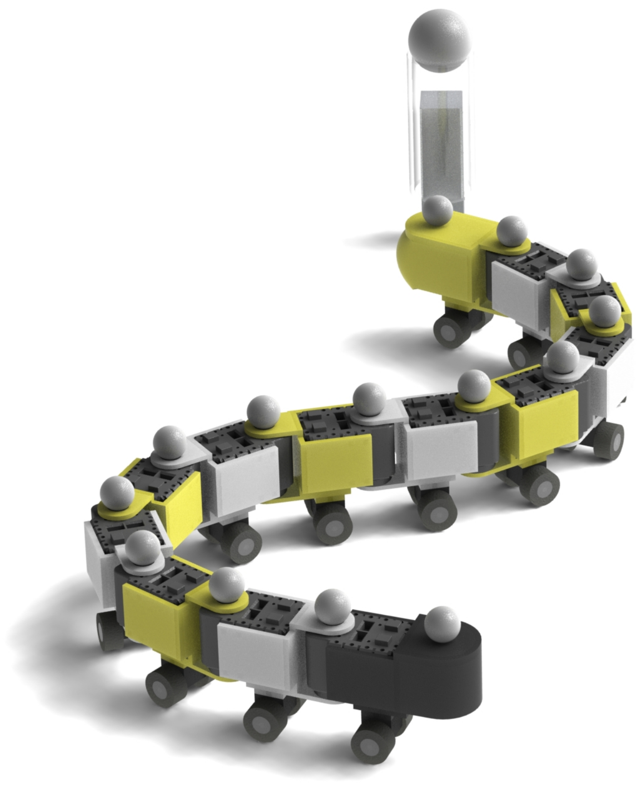


Figure 9.1: An early rendering of the snake robot in a typical single post experiment (courtesy of Zachary Goddard).

The single post experiments were crucial both for controller development as well as testing the controller. In the single post setup, a box of initial conditions is sampled randomly. This box represents the position of the head joint of the snake on the mat. The box is symmetric about the post, which means that if the head position of the robot falls along the axis of symmetry of this box, the trajectory of the center of the mass of the robot passes directly through the center of the post. The size of this initial condition box is dictated by the size of the robot, and is 28 cm wide by 40 cm tall. Once the system has been calibrated so that the robot is traveling straight as shown in Fig. 7.8, and since the locomotion of the snake is cyclic, sampling this box of initial conditions gives a thorough sampling of the allowable contact states (contact angle and phase defined in Chapter 7) between the robot and the post. This is verified in Fig. 9.2, where the scattering angle is plotted as a function of the contact states. Note that for the single post, a majority of the experiments do not actually hit the post. Experiments where no contact was made don't offer much insight into the interactions between the robot and the post, and are generally omitted from the analysis.

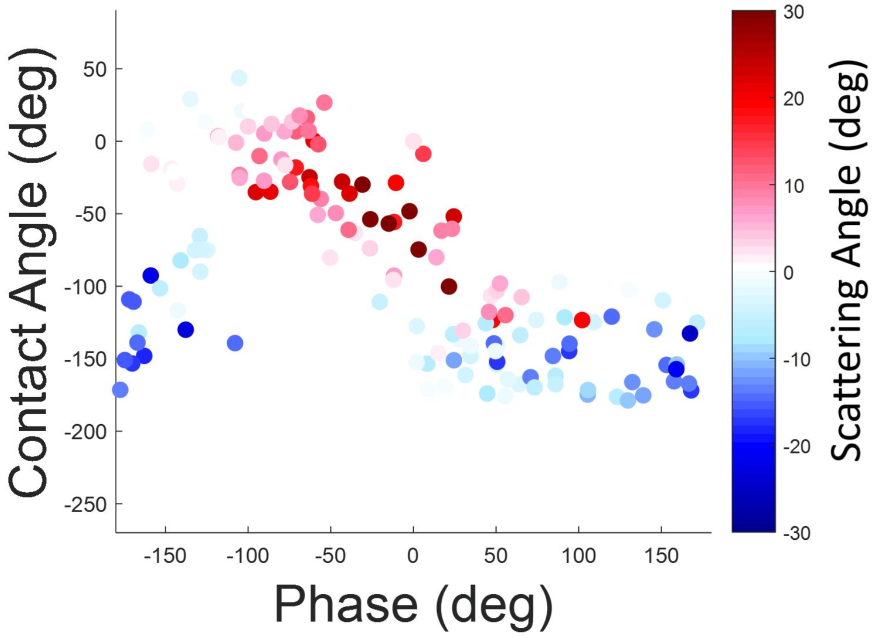
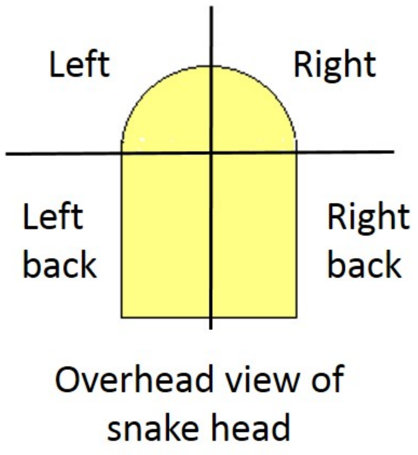


Figure 9.2: The scattering angle is calculated as a function of the contact states contact angle, ϕ, and phase, η. Comparison with the simulation results in [2] shows that our sampling of robot head initial conditions provides a good representation of the allowable contact states between the robot and the post.

9.1.1 Open Loop Results: Predicting Scattering Direction



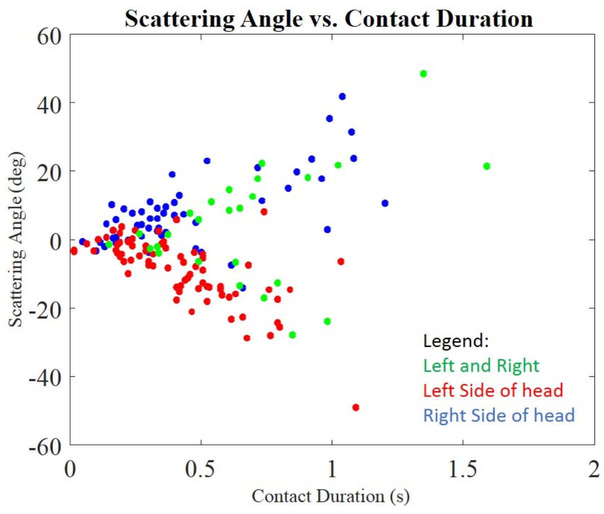


Figure 9.3: Contact between the head of the robot and the post is used to predict scattering direction. The left-hand figure gives an overview of the possible contact states on the head of the robot, and right-hand figure plots scattering angle vs contact duration, colored by contact state.

In Chapter 8, we predicted that the contact location on the head of the robot would be a good indication of the direction the robot should scatter. One of the important early results from the open loop data was to verify this prediction experimentally. The plot in Fig. 9.3 shows the contact state of the robot at the moment of first contact as a function of the scattering angle and contact duration. Note that the left back and right back sensors are not used in order to stay consistent with the simulation work presented in [2]. The points in blue represent the experiments where the robot registered contact on the right front panel of the robot, and generally scattered in the positive direction. The points in red represent the experiments where the robot detected contact on the left front panel of the robot, and these points generally scatter in the negative direction. This shows that the initial contact detected by the robot at first contact is generally a good predictor of the direction that the robot will scatter, and this is what is implemented in the first iteration of the controller in order to determine which direction to steer. The points in green, however, make contact with both the left and right panel at the first point of contact and can scatter in either direction. The analysis is Section 9.3 provides some insight into a better way to predict scattering direction by addressing these points, and is used to inform a better anticipatory controller in Chapter 10.

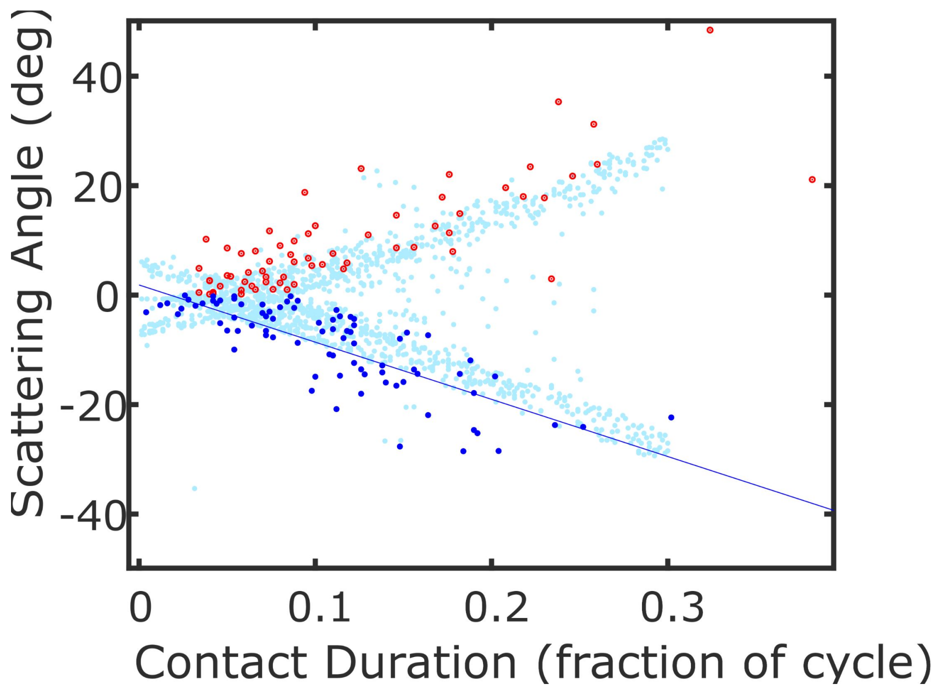


Figure 9.4: Scattering angle vs contact duration for simulation (light blue) and experiment (positive angles in red and negative angles in blue).

In addition to the steering direction, we wanted to verify the relationship shown in Fig. 8.2 experimentally in order to have a method to determine the magnitude which the robot should steer. The light blue points represent the simulation data, and the points in red and blue are the positive and negative experimental data respectively. Note that the durations for both the simulation and experiment are scaled by their respective frequencies in order to make a fair comparison, which is why the duration units are in fractions of a cycle. This figure not only shows a strong agreement between simulation and experiment, but also provides us with the equation we need for the controller: the linear regression of the negative data points shown in dark blue. For simplicity, we used this same equation for both the positive and negative scattering angles based on the symmetry of Fig. 9.4. However, we will see for the multipost experiments, the resulting open loop scattering distributions are not always perfectly symmetric, and this assumption may impact controller performance.

9.1.2 Controller Results

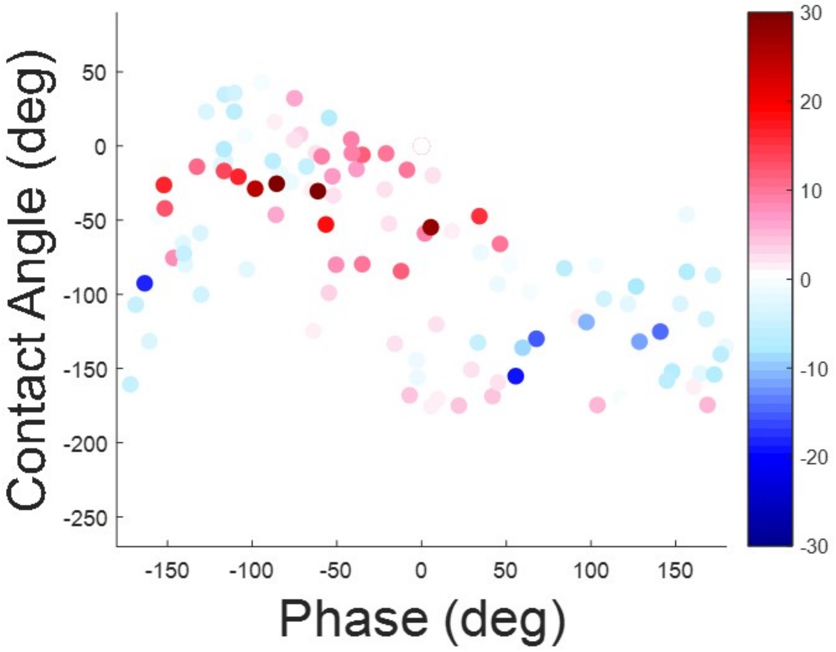


Figure 9.5: Scattering angle is plotted as a function of the two contact states for the controller.

Using these open loop experiments, we were able to develop the controller, then test it on the single post setup. The same sampling procedure is used to acquire the controller data, and roughly 500 total experiments are collected for both the open loop and controller so that we can determine the effect of the controller on the scattering distribution by comparing the controller data against the open loop data. Fig. 9.5 plots the controller scattering angle as a function of the two contact states. Note that there are fewer dark red and blue points in this figure than in 9.2. However, to make a more direct comparison, it is easier to compare the histograms of the scattering angle in Fig. 9.6. These histograms of the scattering angle of experiments where the robot made contact with the post show a reduction in scattering with the implementation of the controller. Note that these two probability density plots can be roughly approximated with a Gaussian function. In order to quantify this reduction in scattering, we fit the data with these functions. For the open loop experiments, $\sigma = 10.7 \pm 0.9$ and for the controller $\sigma = 6.5 \pm 0.6$. This shows a noticable reduction in scattering, but as we will see in Section 10, the single post controller can be improved considerably with some clever modifications.

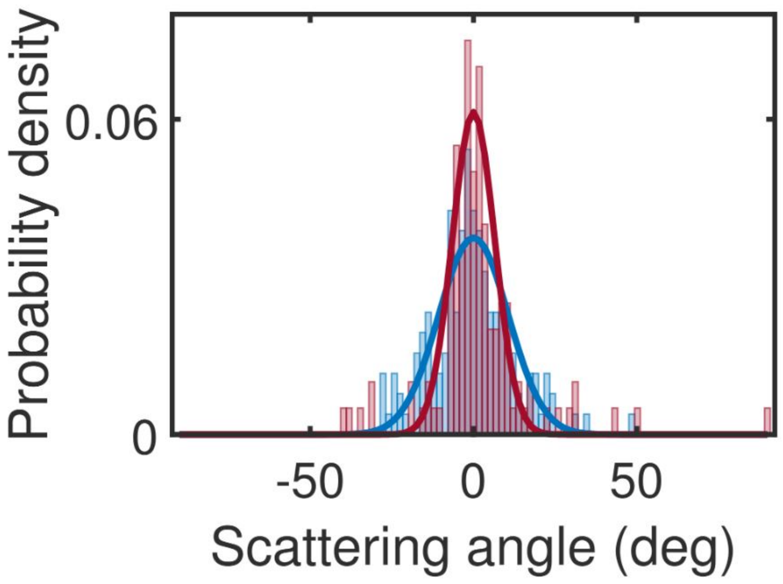


Figure 9.6: The histograms for the open loop controllers are fitted with normalized gaussian functions in order to get an estimate of the spread of the data.

9.2 Multiple Post Results

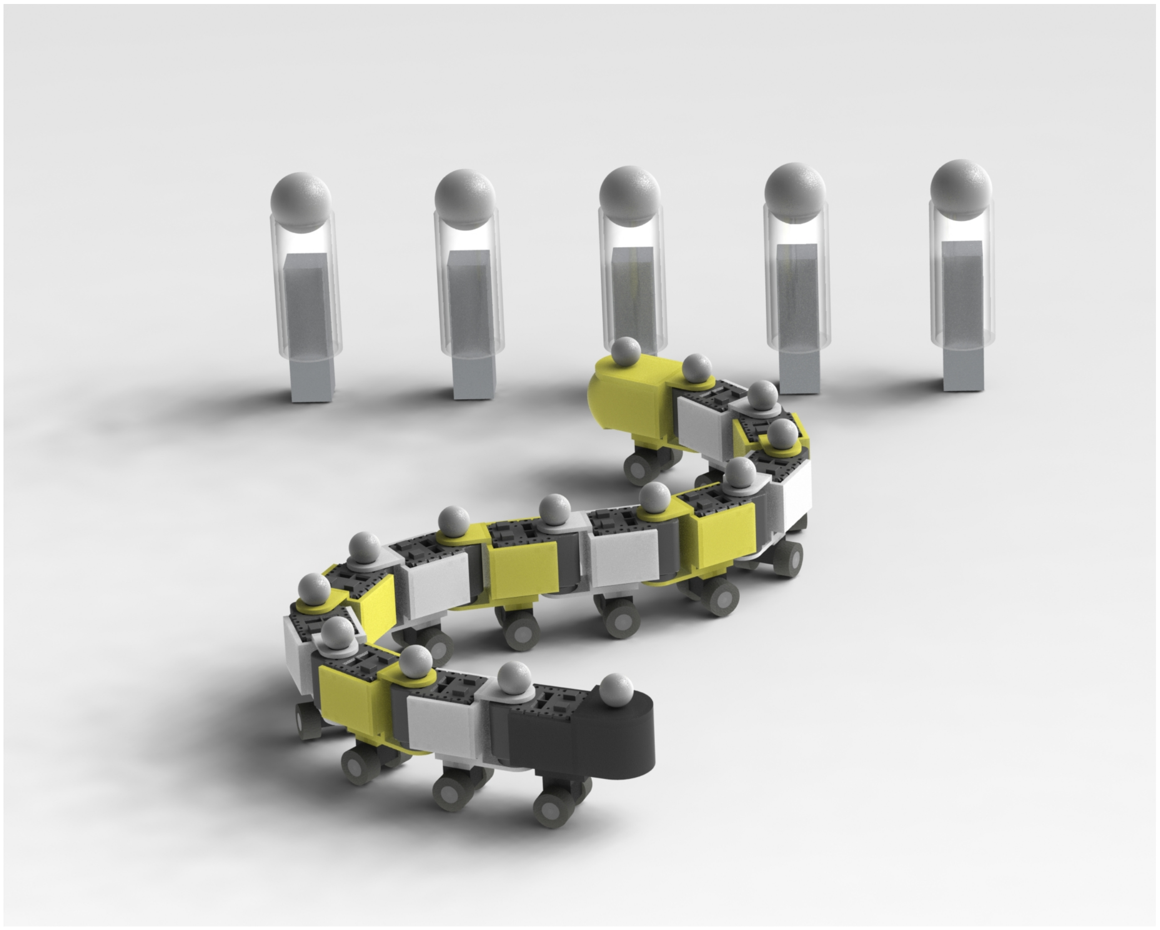


Figure 9.7: An early rendering of the snake robot in the multipost experimental setup prior to making the modifications in Chapter 7 (courtesy of Zachary Goddard). All posts have a radius of 2.3 cm and the spacing between posts is 6.9 cm.

With the controller functioning reasonably well for the single post experiments we were interested to see how this controller could handle an array of five evenly spaced posts. We selected a distance of 6.9 cm between the posts because we observed quantitatively that at smaller spacings, the robot often failed to locomote through the posts, and at larger spacings, the robot effectively acts the same as in the single post case. Again, we systematically sample a box of initial conditions symmetric about the center post. Now, however, the box of initial conditions is dictated by the post spacing and the robot length. The results for the multiple post controller are shown in Fig. 9.8. In the open loop case, the two open loop peaks at ± 20 degrees are destroyed by the controller. However, this means that we cannot use Gaussian functions to estimate the spread of the distribution. Instead, we calculate the 15th and 85th quantiles to use as a measure of the spread, since this tells us that 70 percent of the data falls between these two values. The quantiles for the open loop and controller are (-19.9,15.3) and (-9.5,18) respectively. This shows that the controller has a noticeable reduction in the scattering behavior, but could definitely be improved.

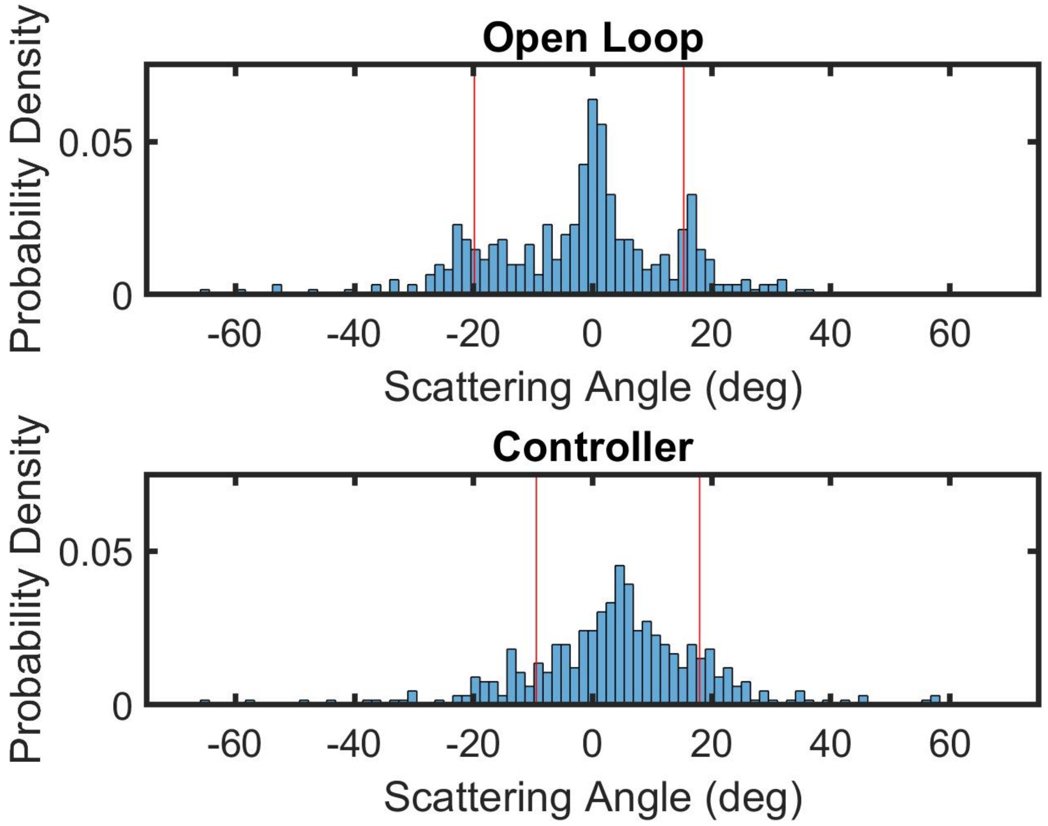


Figure 9.8: Histograms for the multipost open loop and controller, with the 15th and 85th quantile plotted as red vertical lines.

9.3 Discussion and Analysis

In order to understand the when and where the controller is effective, a close analysis of the controller performance is required. Analyzing the shortcomings of the controller in these experiments will allow us to develop alternative approaches that should improve performance. Furthermore, some of the limitations are inherent to our system, and these limitations are also important to be aware of in order to contextualize the results from the experiments.

9.3.1 Single Post Controller Limitations

There are two clear sources of error in the controller in the single post controller. First, there is some error in predicting the magnitude of the scattering. Since the amount the controller steers is determined by a linear regression of experimental data, there is definitely uncertainty associated with this fit. For the points in Fig. 9.4 that do not fall near these two curves, the controller would not be effective at correcting the orientation of the robot.

A much more likely source of error, however, are mistakes in determining the steering direction. We saw in Chapter 8 that the first point of contact between the robot and the posts was a fairly good indicator of the direction that the robot will scatter. We hypothesized that this prediction could be further improved by instead looking at the last contact between the robot and the post. Fig 9.9 shows that last contact is in fact a much better indicator of scattering direction than first contact. With this in mind, the controller is modified in Chapter 9.9 to choose the steering direction based on the last panel in contact with the post.

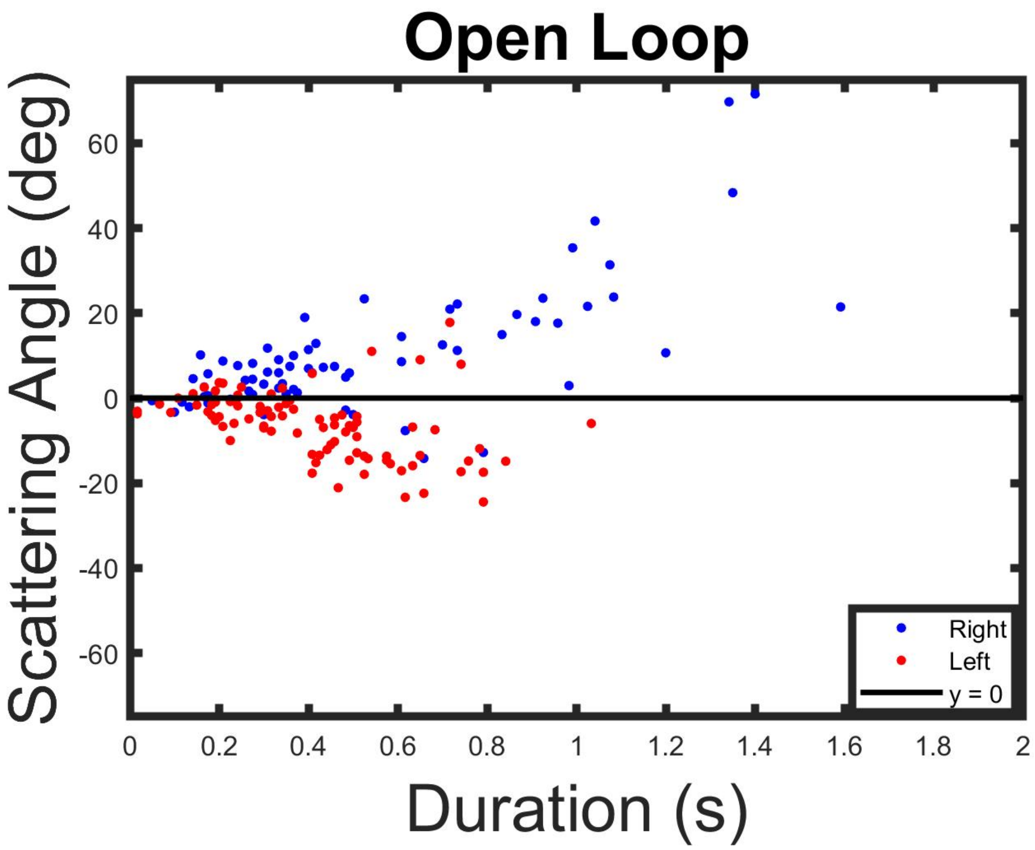


Figure 9.9: Scattering angle vs duration colored by contact sensing panel in contact with post at the last moment of contact. Last contact serves as a better indicator of scattering direction than first contact.

9.3.2 Multiple Post Controller Limitation

One clear difference between the multipost setup and the single post experiments is that the robot must locomote between the posts after the contact between the head and the posts is broken. This certainly has the potential to impact the trajectory of the robot, especially if the controller is attempting to change the shape of the snake while it is fitting through this gap in the posts. Therefore, these re-orientations caused only by the body cannot be corrected by the controller since we only have local sensing on the head of the robot. In order to minimize the effects of these body contacts, we propose several variations on the controller based on the idea that steering at the right time or in the right way can assist the snake to locomote through the posts. These approaches are developed in detail in Chapter 10, and the results for each of the modified controllers are presented.

Another complication present in the multiple post setup is that the robot can actually make contact with multiple posts. In this case, it is unclear which contact duration will dictate the scattering behavior of the robot. Again, we turn to the simulations in [2], which say that the longest contact duration will be the best predictor of the scattering behavior as shown in Fig. 9.10. Therefore, we would expect that the somewhat naive open loop controller would guess the scattering direction incorrectly. Imagine the robot making a glancing contact with a peg on the left of the head, followed by a longer contact on the peg on its right. We would expect that the second contact will dictate the direction of the scattering. However, the controller will actually steer the robot based on the first contact, and will actually make the controller perform worse than the open loop in this example. Therefore, we propose modifying the open loop controller so that the scattering direction is predicted based on the longest duration. Implementing this controller, however, is not trivial. Chapter 10 discusses one possible method to accomplish this as well as the potential drawbacks of this method.

9.3.3 Robot Limitations

The physical limitations of the robot also certainly had an impact on the controller performance in some experiments. While the simulation assumes that individual robot segments are perfectly rigid, this is certainly not true with 3D printed brackets. While these brackets generally hold the motors in place, in some instances when they are experiencing high torque (like when the robot is ”caught” in between posts in the multipost setup), these brackets are certainly capable of bending. In particular, it seemed like the brackets became a little bit more flexible over time. Many of the brackets have been on the robot for well over a year, experiencing thousands of experiments. In order to make future versions of the robot more robust, these brackets can be reinforced by making them a little bit thicker or using a higher fill on the 3D printer.

In addition, the lego wheels were also subject to wear over the course of many experiments. This was a particularly problematic because once the inside of the wheel hub has been worn out by friction between the axle and the hub, the wheels lose the direction dependent friction that is needed to locomote. As this friction property changes at different rates on different wheel modules along the robot's body, the robot will slowly lose its ability to locomote straight. Even with the gantry servo motor calibrated correctly, worn wheels will cause the robot to follow a curved trajectory instead of a straight line, which has the potential to impact the quality of our results.

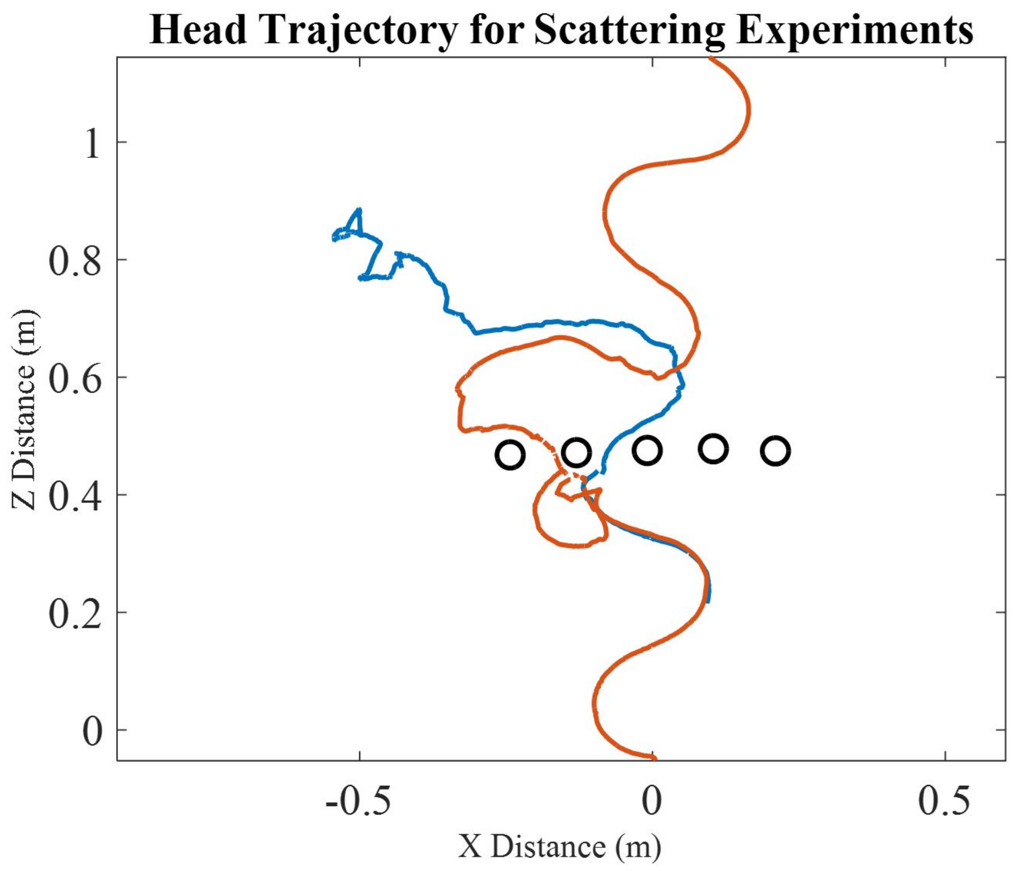
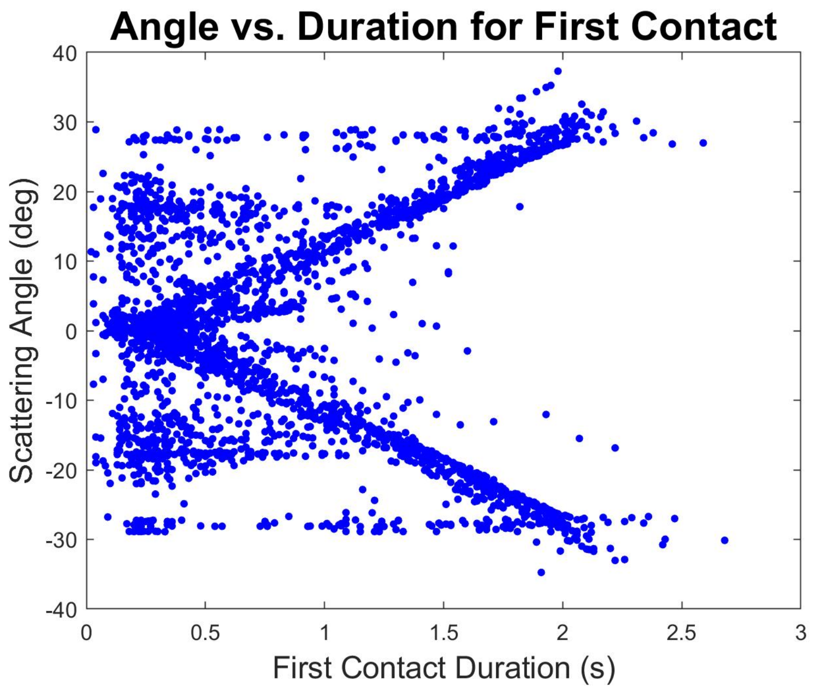


Figure 9.11: Unusual deformations sometimes occur for both the open loop (blue) and controller (red) in the multipost setup.

Finally, the robot is limited by the torque output of the robot. In general, deformations to the shape of the robot tend to be relatively small, but sometimes the robot hits a post with the right combination of contact angle and phase which causes the head to get pinned on the post. These contact cases are very rare, but the controller is unable to correct for the large scattering which can be generated. The resulting trajectories are disjointed and unpredictable as shown in Fig. 9.11, and controlling the robot in these uncommon cases is outside of the scope of what we hope to accomplish with the controller.



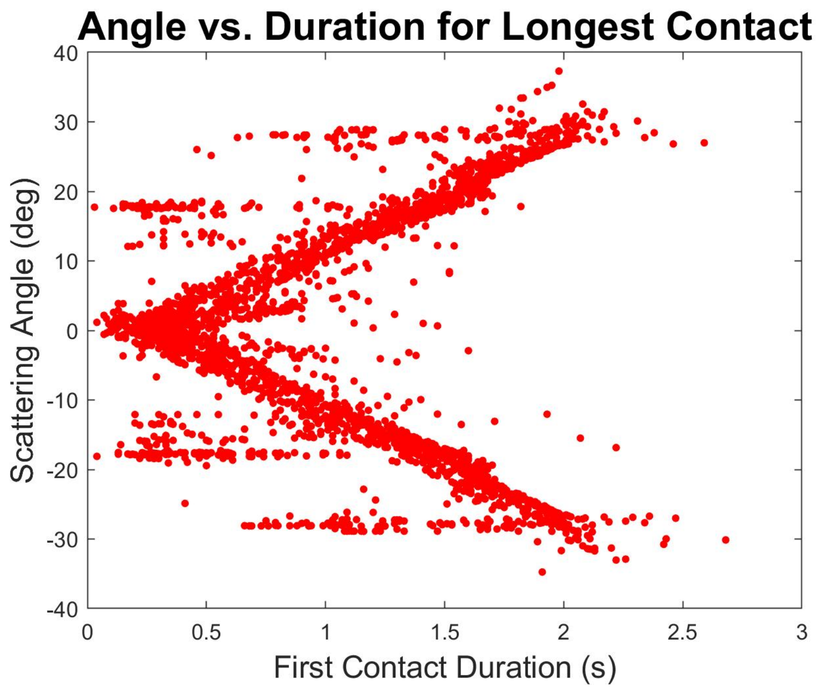


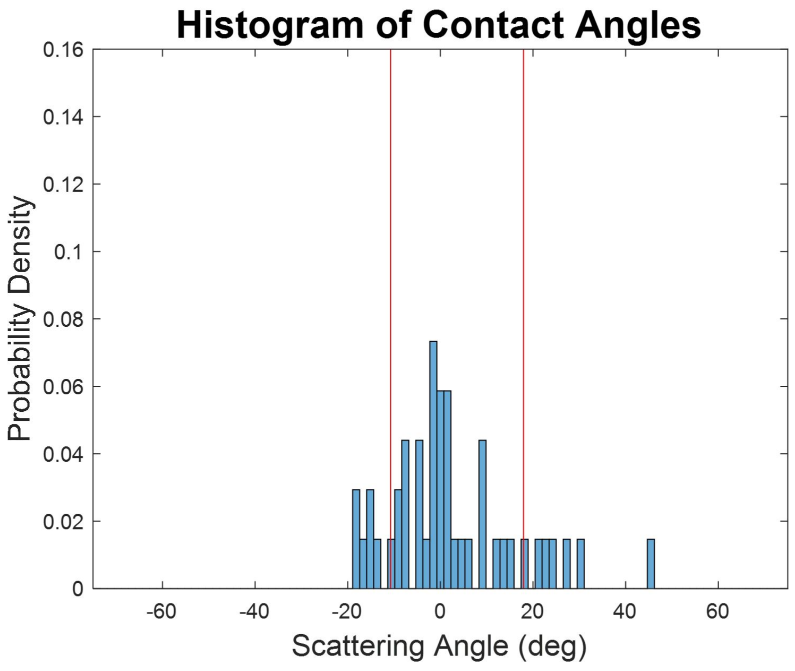
Figure 9.10: In the multipost experiments, having more than one contact between the robot head and the posts is possible. Results from simulations in [2] suggest that the longest contact duration is a better indicator of the resulting scattering duration than the first contact. The top figure shows the angle vs duration relationship for the first contact between the robot and the posts and the bottom figure shows the longest contact between the robot and the posts.

CHAPTER 10
IMPROVING THE CONTROLLER

In this chapter, ideas for improving the performance of the controller are quickly developed and tested. For this reason, each iteration of the controller has significantly less experimental data than in the previous section. These controller modifications are inspired by and seek to address some of the limitations and challenges of the controller discovered by the analysis in Section 9.3. First, in 10.1 the single post controller is modified to decide scattering direction based on the last contact panel in contact with the post instead of the first contact panel in contact with the post. Then, in 10.2, several modifications are tested in order to address the limitations of the controller in the multiple posts configuration. In particular, we will attempt to reduce the body effects of the snake robot as it locomotes through gaps in the posts and improve the steering direction prediction when the snake contacts more than one post. We hope that these modifications to the controller offer the road map to how to create a more effective controller for navigating the single and multipost setups, as well as offering further insights into the collision between the robot and posts.

10.1 Modified Single Post Controller

While the single post controller results presented in Chapter 9 certainly show a reduction in scattering, we believe that this can be improved further by using the last contact to determine scattering direction as opposed to first contact as mentioned in Section 9.3. We hypothesize that making this change will reduce the overall scattering pattern for the single post by greatly reducing the point where the controller chooses to steer in the wrong direction. To test this hypothesis, the new controller and open loop were tested for 100 initial conditions each. These initial conditions span a 8 by 40 cm box, with experiments evenly spaced 2 cm apart.



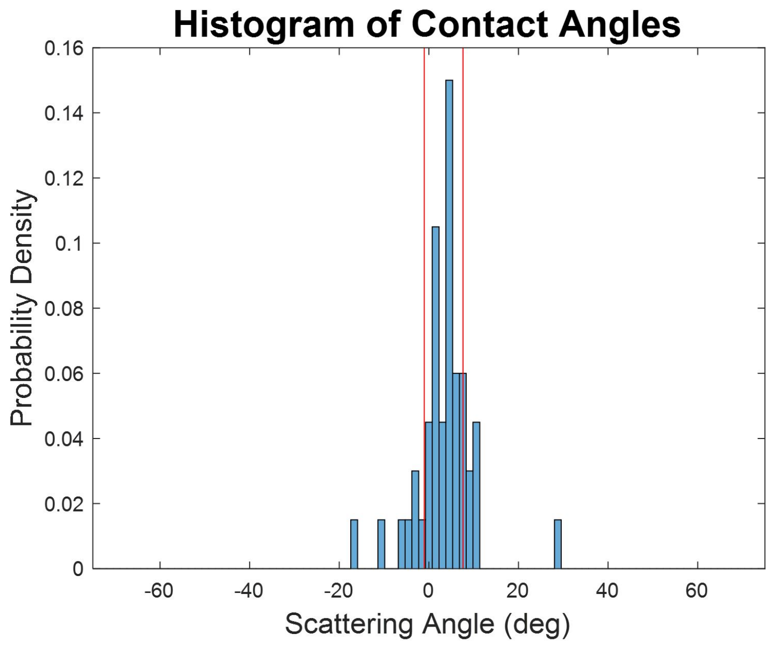


Figure 10.1: Histograms of the open loop (top) and new controller (bottom), with quantiles shown in red.

We found that the new controller was able significantly reduce the scattering distribution. Fig. 9.8 shows the histograms for the open loop and controller for these initial conditions. We can see that the quantiles for the open loop (-10.7,17.9) span a much larger range of angles than those for the controller (-1.0,7.6).

10.2 Modified Multiple Post Controller

Now that we have dramatically improved the controller performance for the single post case, we can consider improvements specific to the multipost setup. As mentioned in Section 9.3, the two biggest issues impacting the performance of the controller in the multipost setup are the re-orientations caused by the body moving through the posts and multiple head contacts. This section proposes and tests modifications to the controller in order to help address this concerns.

10.2.1 Reducing robot body effects

The first group of controller modifications seeks to reduce the body re-orientations by changing how and when the steering mechanism is implemented. Recall from Chapter 8 that the robot can be steered in either direction by just contracting or expanding its shape if the steering is initialized at the appropriate point of zero curvature. Based on this observation, four variations of the controller are compared against the open loop controller and each other. For each modification, 60 initial conditions centered around the middle post and evenly spaced 2 cm apart are tested. The controller modifications are described as follows:

1. Improved single post controller: This is the controller just presented in the previous section. The last contact state of the first detected contact is used to determine which direction to steer the robot. Then the controller will initialize steering at the next point of zero curvature. The pseudo-code below summarizes the controller and provides a reference for comparing the proposed modified multiple post controllers.

```
if End of first contact is detected then
```

```
    Record last contact panel touched
    Record contact duration
    Choose steering direction based on last contact
    Determine amplitude change $\Delta$A from eq. 8.2
    Begin Amplitude Modulation Method steering at next point of zero curvature
end if
```

2. Expanding controller: This controller again picks the direction to steer based on the last contact state of the first detected contact. Then, the controller steers using only the expanding shape. If the expanding shape results in the desired steering direction at the next point of zero curvature, it steers as soon as possible. Otherwise, the snake waits another half undulation to steer with the expanding shape. Implementation of this scheme is summarized by the pseudo-code below.

```
if End of first contact is detected then
    Record last contact panel touched
    Record contact duration
    Choose steering direction based on last contact
    Determine amplitude change $\Delta$A from eq. 8.2
end if
if Expanding Shape ($\Delta A < 0$) then
    Begin steering at next point of zero curvature
else
    Wait another half cycle to steer
    Flip the sign of $\Delta$A
end if
```

3. Contracting controller: This controller again picks the direction to steer based on the

last contact state of the first detected contact. Then, the controller steers using only the contracting shape. If the contracting shape results in the desired steering direction at the next point of zero curvature, it steers as soon as possible. Otherwise, the snake waits another half undulation to steer with the contracting shape.Implementation of this scheme is summarized by the pseudo-code below.

```
if End of first contact is detected then
    Record last contact panel touched
    Record contact duration
    Choose steering direction based on last contact
    Determine amplitude change $\Delta A$ from eq. 8.2
end if
if Contracting Shape ($\Delta A > 0$) then
    Begin steering at next point of zero curvature
else
    Wait another half cycle to steer
    Flip the sign of $\Delta A$
end if
```

4. Delayed Controller: This controller again picks the direction to steer based on the last contact state of the first detected contact. This controller steers using either the contracting or expanding shape, but waits an extra half undulation before steering. The idea behind this controller is to just give the robot more time to clear the posts before sending an amplitude modulation down the body.Implementation of this scheme is summarized by the pseudo-code below.

```
if End of first contact is detected then
    Record last contact panel touched
```

```
        Record contact duration
        Choose steering direction based on last contact
        Determine amplitude change $\Delta$A from eq. 8.2
        Wait an extra half cycle then begin AMM steering
    end if
```

The results of these four controllers were compared, and the expanding controller was shown to be the most effective at reducing the scattering behavior. The raw data for the expanding controller and open loop are presented in Fig. 10.2 to show visually how the controller is again most effective for durations longer than 0.5 seconds. Furthermore, in order to estimate the spread of the data, the quantiles are plotted in the histograms of Fig. 10.3. The quantiles for the expanding controller (-2.0,14.2) show a considerable reduction over the open loop quantiles (-8.9,23.3) as well as the other controller variations. If we define a metric δ for describing the spread of these distributions as the difference between the 85th quantile and the 15th quantile, we can easily rank these modified controller in terms of performance. These controllers, in order of best to worst performance, are the expanding controller (δ = 16.2), the improved single post controller (δ = 23.1), the contracting controller (δ = 24), the open loop controller (δ = 32.2), and the delayed controller (δ = 41.3). It is worth noting that while both the expanding and contracting controller were on par with or better than the improved single post controller, the delayed controller performed quite poorly. This suggests that there may be a cost associated with waiting too late to steer. The histograms for these other controllers are included in the appendix.

10.2.2 Addressing Multiple Collision Experiments

As mentioned earlier, multiple head collisions is a problem that is not currently addressed by the controller. Based on the simulation results in Section 9.3, we believe modifying the controller to choose the steering direction based on the longest contact duration will help to mitigate this problem. Taking the expanding controller that performed the best in

Section 10.2.1, we build on this controller to adapt it to base its decision on the longest of the first two contacts between the robot and the post. One of the complications of this controller is that we need to estimate how long to wait for a potential second contact.

This is addressed by adding a waiting state in between the end of the first contact and the start of the steering implementation. The longest duration controller is summarized below:

While the snake is moving, different conditions are checked during each loop iteration to determine when to activate the controller. First, the code checks to see if contact is detected between the head of the snake and the posts. If this contact is detected the contact start time is recorded, and the code checks each iteration to see if the contact has been broken. Once this contact is broken, the contact end time and contact panel are recorded. The end of contact also causes the code to enter a wait state, which was selected to be three seconds. During this wait state, again we check for a second contact initialization and contact end. If a second contact is not detected during the wait state, the snake determines the correct magnitude and direction to steer based on the contact duration and final contact panel, as well as the appropriate point of zero curvature to begin the steering. These parameters are fed as inputs into a subfunction that executes the amplitude modulation method to steer the snake. However, if a second contact is detected during this wait state, the duration and contact panel for this contact are recorded, and the wait state is terminated immediately once this second contact is broken. If the second contact is longer than the first contact, the appropriate magnitude and direction to steer is determined based on this second contact. In order to implement steering, we use only the expanding shape like in the expanding controller discussed in the previous section, by checking the sign of the amplitude change. If the amplitude change is negative, then the snake steers at the next point of zero curvature, but if the sign is positive then the sign of the amplitude is reversed and the steering begins a half undulation later.

In this manner, we can chose the correct direction to steer by using the longest of the first two contacts. This controller was tested for 60 experiments, with the head initial conditions centered on the middle post, and spaced two cm apart. The results from this controller, which are shown in 10.4, demonstrate that the performance of this control scheme is fairly comparable with that of the expanding controller without looking at the longest duration. In particular, the quantiles (-1.2,15.1) and $\delta = 16.3$ for this controller are nearly identical to those of the first contact, expanding controller. While implementing longest contact did not produce the further reduction of scattering we were expecting, this result at least shows consistency performance from the expanding controller.

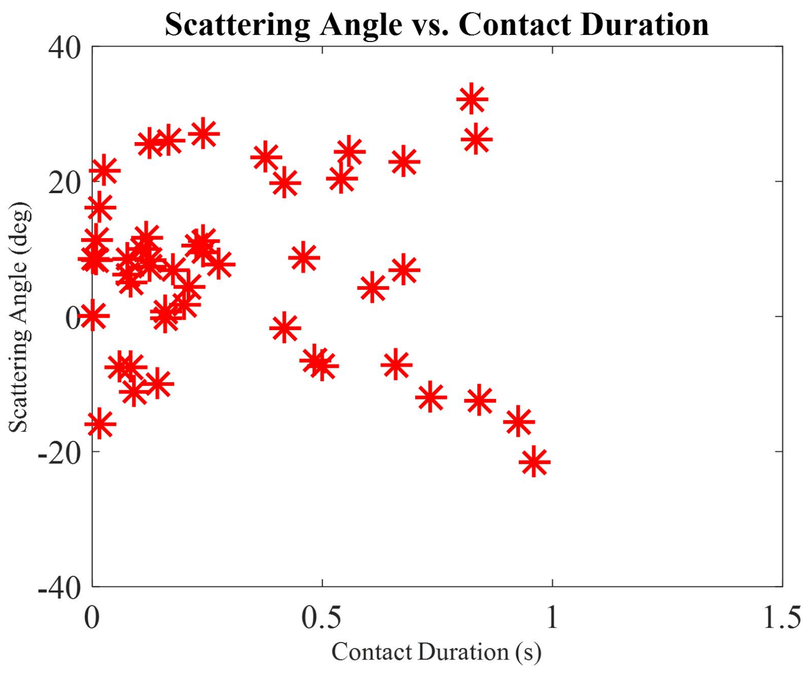
While the longest duration/expanding controller did not offer an additional reduction in scattering over the first duration/expanding controller, we believe that this controller can be further improved with a few simple modifications. In particular, the duration of the wait state is an easily adjustable parameter of the controller which could have an impact on its performance. The wait state was conservatively selected to be three seconds since durations longer than one second are rarely observed, and we estimated that these durations would never be more than half a second apart. The downside to this conservative wait state, however, is that the steering method is often implemented very late in the robot's trajectory. We believe that this wait state is much longer than it needs to be, and reducing the length of this state could improve the performance of the controller.

10.3 Conclusions and Future work

In conclusion, this thesis outlined the iterative approach used to design an anticipatory controller for a snake robot based on a model for the interaction between the robot and its environment. We saw how the development of an automated gantry system assisted this process by allowing data to be collected in large quantities. With the aid of this gantry system, we studied the open loop behavior of the snake for a single vertical post and an array of five evenly spaced posts to verify previous experimental results and determine the model. Furthermore, we used this model along with an established steering method to design an anticipatory controller to correct for the scattering caused by interactions between the robots and the posts. We carefully studied the results of implementing this control scheme for both experimental setups, and identified potential short comings of the controller based on quantative observations and data analysis. Finally, we used these shortcomings to rapidly iterate on variations on the control scheme to address these limitations and improve performance. We found that this iterative process resulted in two variations of the controller that significantly reduce the scattering of the robot in both sets of experiments.

In particular, we saw from our comparison of the controllers in 10.2.1 that for the multipost setup, the ability to steer is not enough, how you implement the steering on the robot matters. We believe that the expanding controller is particularly well suited for a horizontal line of posts because it modifies the shape of the robot in a way that assists the robot fit between the posts. This minimizes the effects of robot body collisions which our model does not take into account. The expanding controller, however, may not be the best method of steering in every environment. For example, in a row of vertical posts you might want to use the contracting controller so that you can achieve your desired steering while traveling less distance in the direction of locomotion. Future work along these lines could examine which method of steering is most advantageous for a particular set of obstacles. In a way, telling the robot to expand its shape because we know it needs to fit through a gap in posts is cheating in a robotics sense, since the controller should rely on and make decisions only based on the available onboard sensing. However, if the robot can learn which steering method is best for a particular environment, then we can add additional sensing on the robot (like a vision system) to allow the robot to estimate the set of obstacles the robot is approaching, and use a higher level controller to tell the robot which variation of the anticipatory controller should be applied for this particular scenario.

Another interesting area for future work would be to study the effect of compliance on the performance of the robot in these unstructured environments. Previous work with the biologically inspired Rhex robot has shown how the passive compliance in the legs of the robot can assist the robot to move through challenging terrain [18]. Therefore, cleverly adjusting the compliance of the robot at contact with obstacles may lead to some interesting new controllers, especially since recent work with biological snakes [19] has shown how the passive dynamics are also important for snake locomotion. Therefore, we envision that the work presented in this thesis along with other works in the field can help lead to the development of improved snake robot control schemes. These novel control schemes will allow snake robots to move gracefully through any arbitrary arrangement of obstacles so that these robots can be deployed successfully in real world applications.



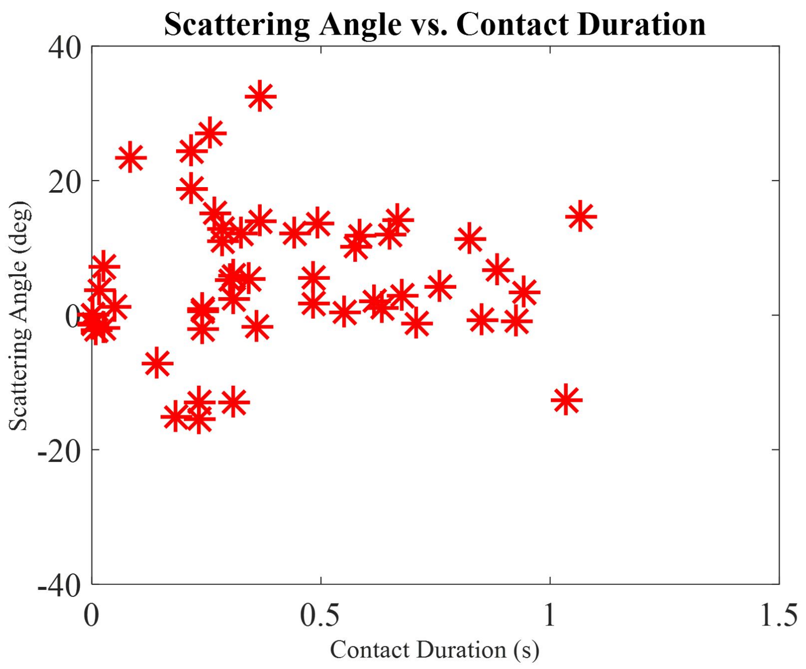
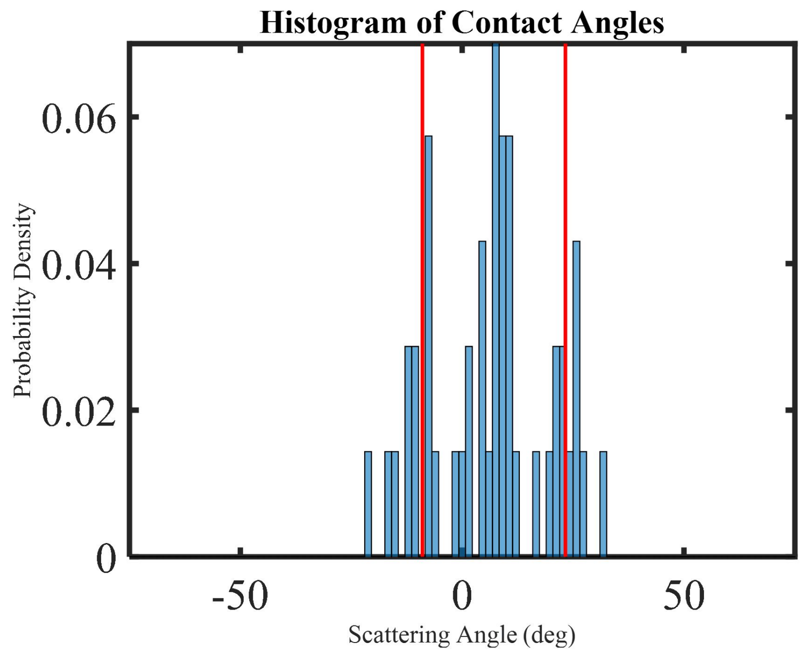


Figure 10.2: The raw data for the expanding controller (bottom) shows a significant decrease in scattering over the open loop experiments (top), especially for contact durations greater than 0.5 seconds.



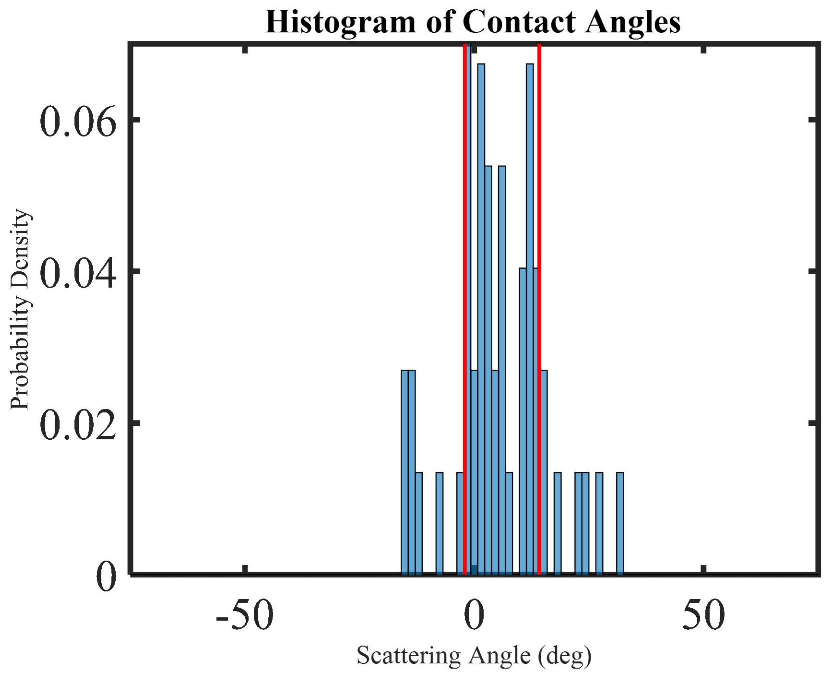
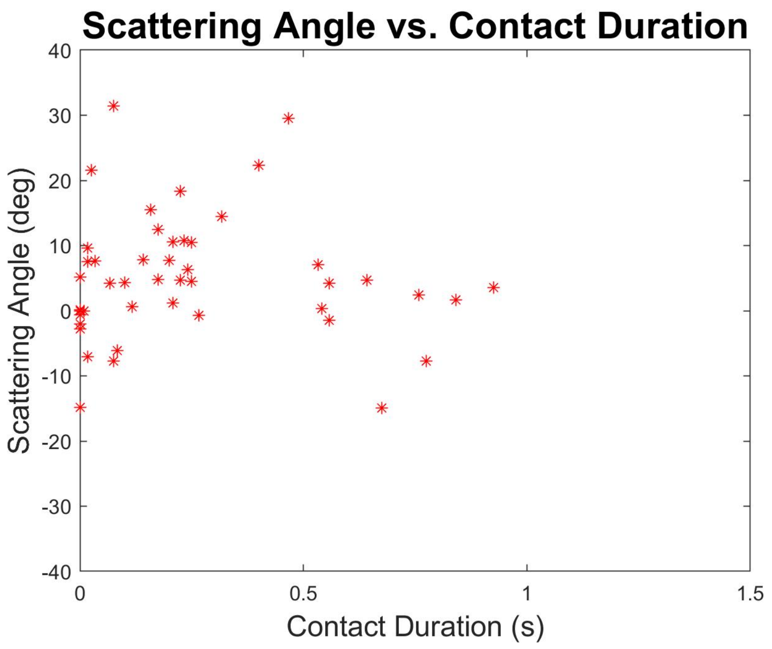


Figure 10.3: Histograms of the contact angles for the open loop (top) and expanding controller (bottom). The expanding controller shows an improved reduction in scattering over the original controller.



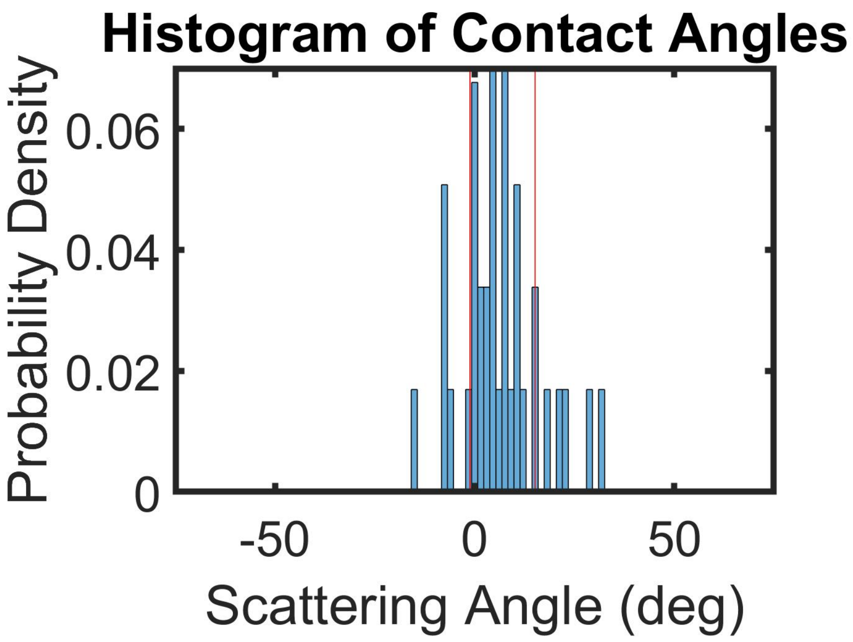
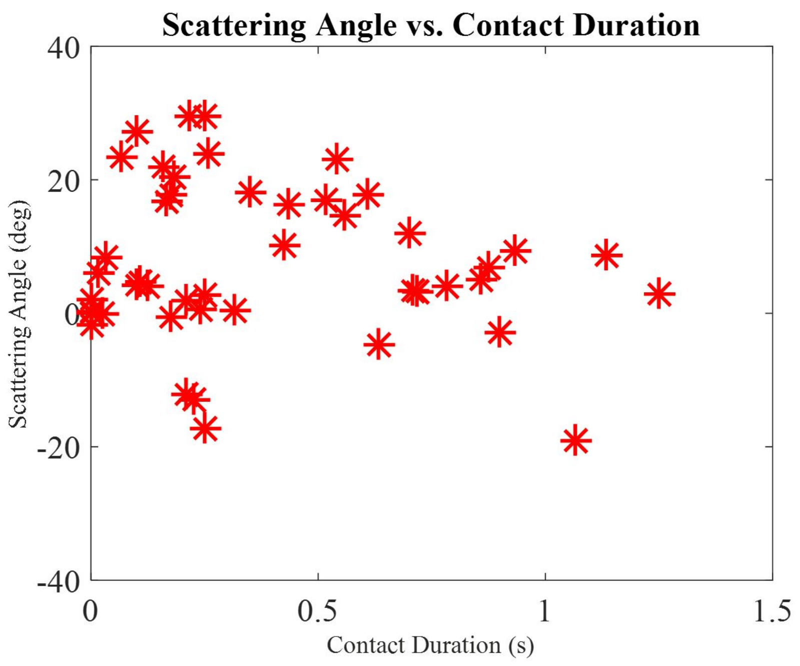


Figure 10.4: Raw data (top) and histogram for the longest duration controller. The longest duration controller shows a similar reduction in scattering with the expanding controller.

Appendices

APPENDIX A

SUPPLEMENTAL FIGURES



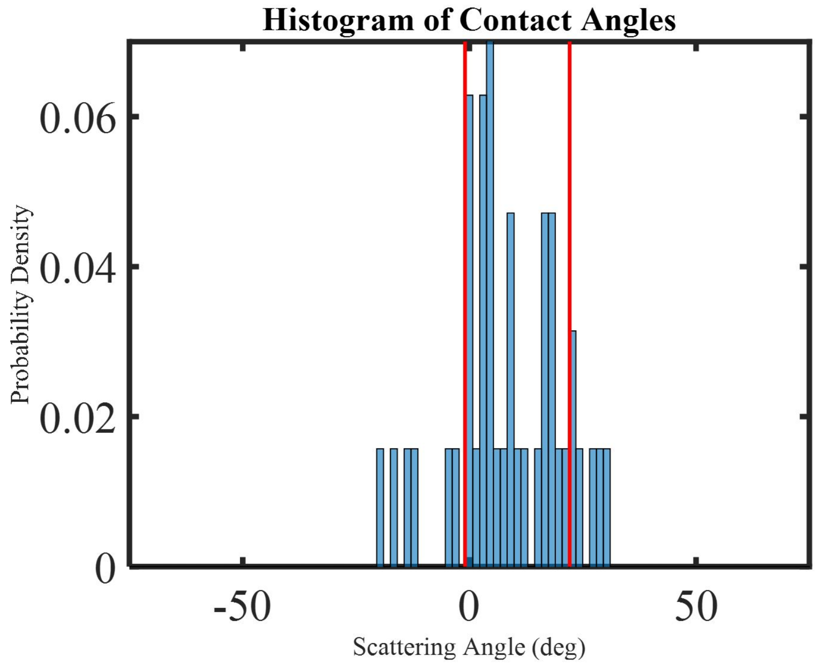
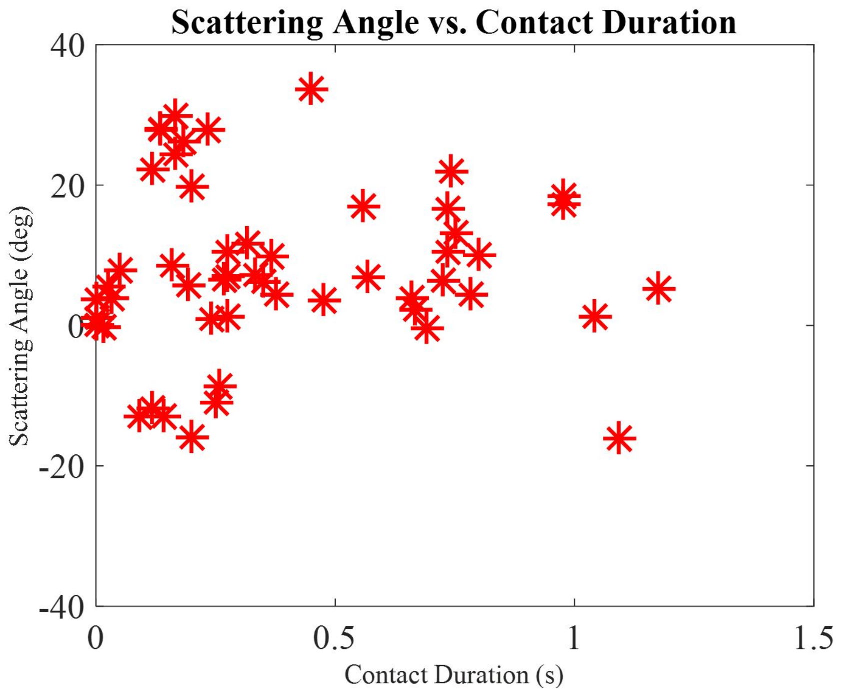


Figure A.1: The raw data for the improved single post controller from the multipost experiments (top), and the associated histogram (bottom) with the 15th and 85th quantiles shown in red. Note that only experiments where head contact was detected are included.



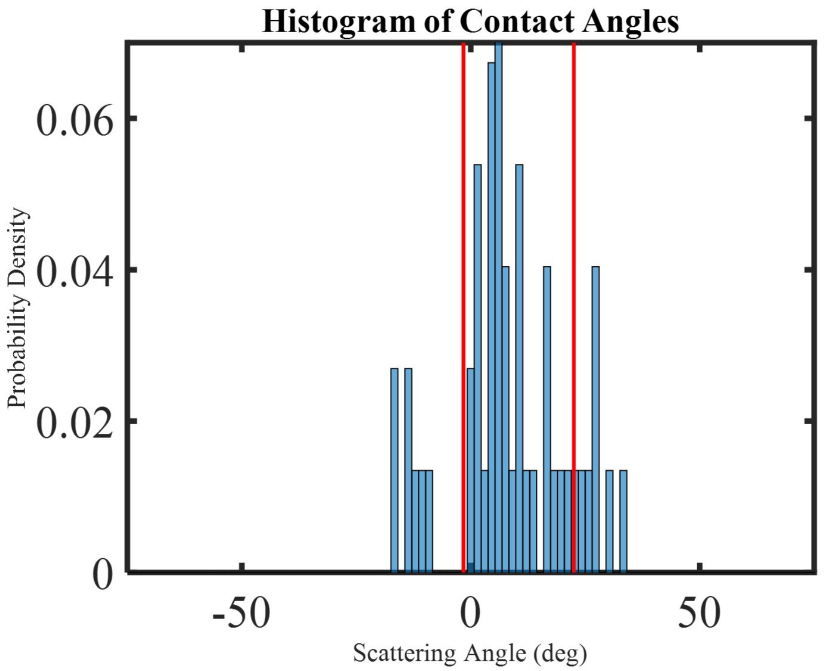
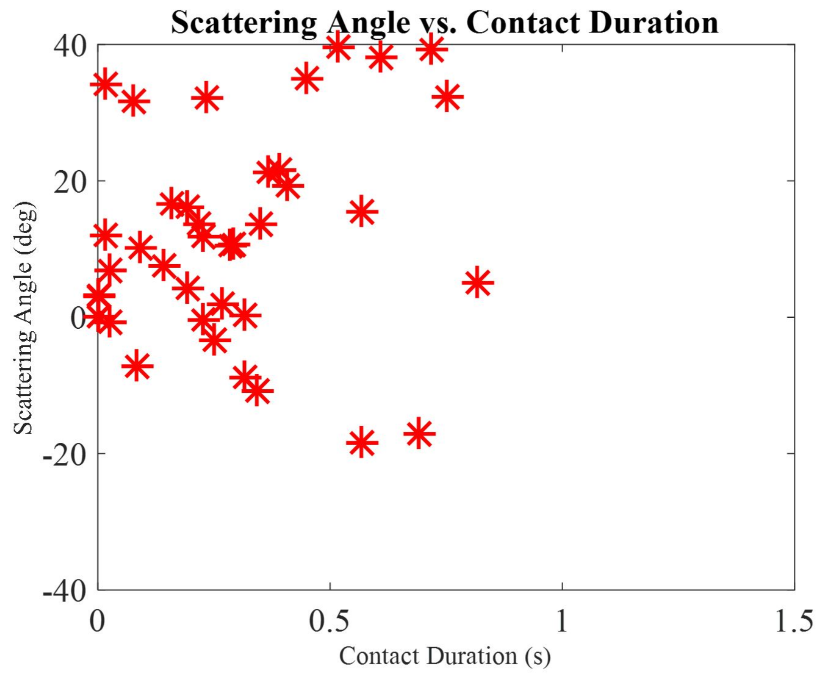


Figure A.2: The raw data for the contracting controller from the multipost experiments (top), and the associated histogram (bottom) with the 15th and 85th quantiles shown in red. Note that only experiments where head contact was detected are included.



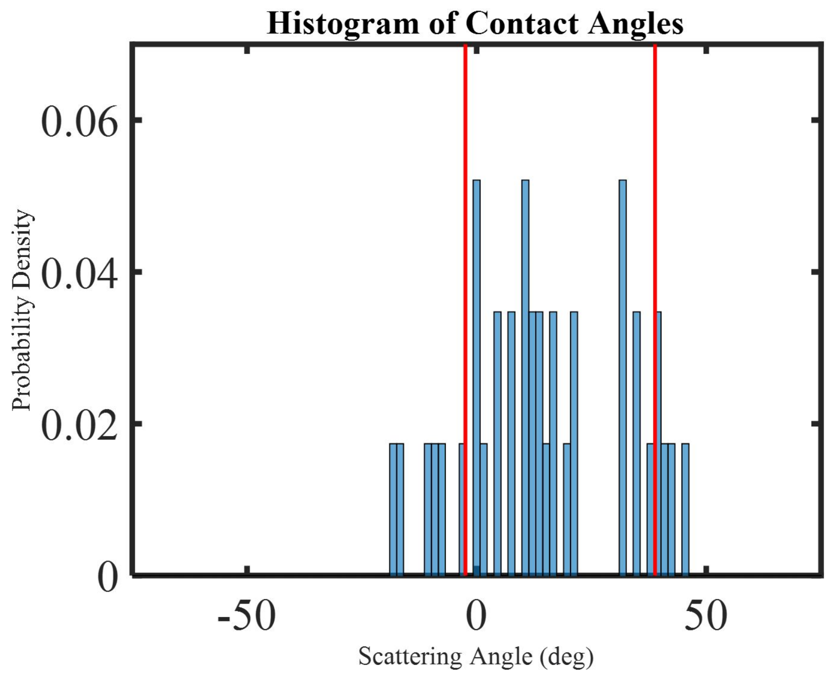


Figure A.3: The raw data for the delayed controller from the multipost experiments (top), and the associated histogram (bottom) with the 15th and 85th quantiles shown in red. Note that only experiments where head contact was detected are included.

REFERENCES

[1] S. Hirose, *Biologically Inspired Robots: Serpentile Locomotors and Manipulators*. Oxford University Press, 1993, ISBN: 0198562616.

[2] J. M. Rieser, P. E. Schiebel, A. Pazouki, F. Qian, Z. Goddard, A. Zangwill, D. Negrut, and D. I. Goldman, "The dynamics of scattering in undulatory active collisions," *ArXiv e-prints*, Nov. 2017. arXiv: `1712.00136 [physics.class-ph]`.

[3] P. E. Schiebel, J. M. Rieser, A. M. Hubbard, L. Chen, and D. I. Goldman, "Collisional diffraction emerges from simple control of limbless locomotion," in *Conference on Biomimetic and Biohybrid Systems*, Springer, 2017, pp. 611–618.

[4] J. Aguilar, T. Zhang, F. Qian, M. Kingsbury, B. McInroe, N. Mazouchova, C. Li, R. Maladen, C. Gong, M. Travers, R. L. Hatton, H. Choset, P. B. Umbanhowar, and D. I. Goldman, *A review on locomotion robophysics: The study of movement at the intersection of robotics, soft matter and dynamical systems*, 2016. arXiv: `1602.04712 [cs.RO]`.

[5] J. Gosyne, C. Hubicki, X. Xiong, A. Ames, and D. Goldman, "Bipedal locomotion up sandy slopes: Systematic experiments using zero moment point methods," *Humanoid RObots (Humanoids), 2018 18th IEEE-RAS International Conference on*, 2018.

[6] F. Qian, K. Daffon, T. Zhang, and D. I. Goldman, "An automated system for systematic testing of locomotion on heterogeneous granular media," Aug. 2013, pp. 547–554, ISBN: 978-981-4525-52-7.

[7] *Motive api: Function reference*.

[8] G. Sartoretti, Y. Shi, W. Paivine, M. Travers, K. Sun, and H. Choset, "Distributed learning for the decentralized control of articulated mobile robots.," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[9] P. Liljebäck, K. Pettersen, Ø. Stavdahl, and J. Gravdahl, *Snake Robots: Modelling, Mechatronics, and Control*. Jan. 2013, ISBN: 978-1-4471-2995-0.

[10] M. Travers, J. Whitman, P. Schiebel, D. Goldman, and H. Choset, "Shape-based compliance in locomotion," in *Proceedings of Robotics: Science and Systems*, AnnArbor, Michigan, Jun. 2016.

[11] J. Whitman, F. Ruscelli, M. Travers, and H. Choset, “Shape-based compliant control with variable coordination centralization on a snake robot,” in *CDC 2016*, Dec. 2016.

[12] M. Travers, A. Ansari, and H. Choset, “A dynamical systems approach to obstacle navigation for a series-elastic hexapod robot,” in *2016 IEEE Conference on Decision and Control*, Dec. 2016.

[13] F. Qian and D. Goldman, “Anticipatory control using substrate manipulation enables trajectory control of legged locomotion on heterogeneous granular media,” vol. 9467, 2015, pp. 9467 - 9467 - 12.

[14] X. Wu and S. Ma, “Neurally controlled steering for collision-free behavior of a snake robot,” *IEEE Transactions on Control Systems Technology*, vol. 21, no. 6, pp. 2443–2449, Nov. 2013.

[15] M. Tesch, K. Lipkin, I. Brown, R. Hatton, A. Peck, J. Rembisz, and H. Choset, “Parameterized and scripted gaits for modular snake robots,” *Advanced Robotics*, vol. 23, no. 9, pp. 1131–1158, 2009.

[16] A. Tasora, R. Serban, H. Mazhar, A. Pazouki, D. Melanz, J. Fleischmann, M. Taylor, H. Sugiyama, and D. Negrut, “Chrono: An open source multi-physics dynamics engine,” in, T. Kozubek, Ed., Springer, 2016, pp. 19–49.

[17] M. Travers, J. Whitman, and H. Choset, “Shape-based coordination in locomotion control,” *The International Journal of Robotics Research*, p. 0 278 364 918 761 569, 2018.

[18] J. C. Spagna, D. I. Goldman, P.-C. Lin, D. E. Koditschek, and R. J. Full, “Distributed mechanical feedback in arthropods and robots simplifies control of rapid running on challenging terrain,” *Bioinspiration Biomimetics*, vol. 2, no. 1, p. 9, 2007.

[19] P. Schiebel, J. Rieser, A. M. Hubbard, L. Chen, D. Z. Rocklin, and D. Goldman, *Mechanical diffraction reveals the role of passive dynamics in a slithering snake*, 2018.