Project Number: ***MQP SAK-PJG1***

MQP-SAK-JCP1

MQP-SAK-DDP1

ROBOT RESEARCH PLATFORM FOR LOCOMOTION THROUGH GRANULAR MEDIA

A Major Qualifying Project Report

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Bachelor of Science

in Mechanical Engineering

Robotics Engineering

Physics

by

Patrick-Joseph Guill

JC Payan

Douglas Perkins

_____ _____ _____

Date: 4/28/2011

Approved:

_____

Prof. S. A. Koehler, Major Advisor

_____

Prof. K. A. Stafford, Co-Advisor

Keywords:

1. Robotic Snake
2. Sand-Swimming
3. Granular Media

Acknowledgements

We would like to thank the following people for their contribution to this project.

Professor Ken Stafford, for his support and guidance. His unending dedication, supply of ideas, and attention to both the big picture and small details greatly improved this project

Professor Stephan Koehler, for his interest in sand-swimming that led to the creation of the project. His assistance in the development of the guiding model and specification parameters. As well as his guidance and eye for detail that contributed to a more attentively defined project.

Brian Benson, for his experience and guidance in ordering and mass producing components.

J. Talley Guill for his assistance in charting the fault tree and hardening the electronic systems. As well as for his unending support as a sounding board for so many design ideas.

Professor Eduardo Torres-Jara, for his experience and guidance in designing drive systems based upon the working environment.

Abstract

The primary objective of this project is to further the study of motion through granular media. Being a joint Physics-Engineering project, the objective was tackled in a two pronged approach. The physics team was tasked with determining the variability of drag forces based on changes in the depth and surface area of a body. Meanwhile the Engineering team was tasked with the design and construction of a robotic test platform to assist in the study of wave motion within granular media. To this extent, a scalable biomimetic robotic snake was designed with the capability to move below the surface of granular media. Instilling the robot with the capability to follow arbitrary traveling waveforms while monitoring a sensor suite and collecting data for analysis is the first phase in a greater scheme to understand the physics associated with granular motion and propulsion systems.

Contents

List of Tables

List of Figures

Authorship

ACKNOWLEDGEMENTS	PG
ABSTRACT	PG,JP
INTRODUCTION	0
Motivation	PG,JP
Objective	PG,JP
BACKGROUND	
Inspiration From Nature	PG
Physics	JP
Previous Robots	PG,DP
METHODOLOGY	
Project Scope	
Design Specifications	PG,DP
Modification Of Theory	JP
Testing Methods	JP,DP
Design Specifications Discussion	PG
General Design Decisions	PG,DP
Control Methods	PG
Component Selection And Design	PG,DP
Motor	PG,DP
Pneumatic/Hydraulic Joints	DP
Gear train system	DP,JC
Bearings, shafts, bolts, and Gears	DP
Circuit Board	PG
Control System	PG
RESULTS AND ANALYSIS	PG,JP,DP
Physics	JP
Discussion	JP
Engineering Results	
New Joint Design	JP,DP
Final Design Of The Segment	DP
Electronics	PG
SOCIAL IMPLICATIONS	PG
RECOMMENDATIONS	PG,JP,DP
CONCLUSION	PG
APPENDICES	
APPENDIX A: CAD DRAWINGS	DP
APPENDIX B: ELECTRICAL DIAGRAM	PG
APPENDIX C: BILL OF MATERIALS	PG,DP
APPENDIX D: ROBOT ONBOARD SOFTWARE	PG
APPENDIX E: COMPUTER END CODE	PG
APPENDIX F: MATLAB CODE	JP

Introduction

Motivation

Snakes are highly versatile creatures, capable of inhabiting a wide variety of environments from swamps to deserts. They are highly mobile, and able to propel themselves through various granular media, such as sand, with ease. Creating an artificial organism that could mimic this movement would greatly improve humanity's understanding of these versatile creatures. Two years ago, another team made a robot snake with the same goals, but it turned out to have various defects. In particular, the driver system was not possessing sufficient power to maneuver while submerged. Due to this unforeseen design flaw, it was unable to move through the medium as desired and at times malfunctioned. Thus, this project was aimed at creating a new design, one that learns from the failing of the previous generation and strives to improve its capabilities. This was a joint robotics-physics department project, and it is hoped that the new design will better equip researchers with a toolset designed to study locomotion through granular media.

Objective

The goal of this project has been to develop a robotic snake platform capable of moving while submerged in granular media. The snake would then be used as a test platform to analyze the effectiveness of different waveforms to generate lateral motion. The snake is intended to be a tool to assist the primary advisor, Professor Stephan Koehler of WPI, in his study of dense granular flow. This goal was accomplished by designing and prototyping a robotic snake capable of following the shape of an arbitrary traveling wave while fully submerged within the media.

Background

Inspiration from Nature

Among all the creatures that walk, swim, and fly on this planet, none have a more unique and perplexing method of locomotion than snakes. Even within the snake family, there exist multiple forms of locomotion. The most efficient and iconic of these is the sinusoidal wave motion. Sinusoidal motion follows the premise of snake body undulation. During undulation, waves propagate down the length of the snake starting at the head and finishing at the tail. In certain cases, the amplitude of the sinusoidal wave increases towards the back of the snake. Lateral undulation requires a minimum of three contact points to result in forward movement; two to generate a force and a third to balance the forces and move in the proper direction (Dowling, 15). Under ideal conditions each point on the snake follows the point before it, so a single path is used. Most animals choose their gaits based on the speed at which they want to go. However, snakes choose their gait based on what environment they travel through. Therefore, it can be concluded that because snakes use lateral undulating motion for movement through sand, and it is the most suitable method of locomotion (Dowling, 20). This conclusion has prompted the scientific community to investigate how snakes move, and how this motion is harnessed to propel the creatures even while submerged within a granular media.

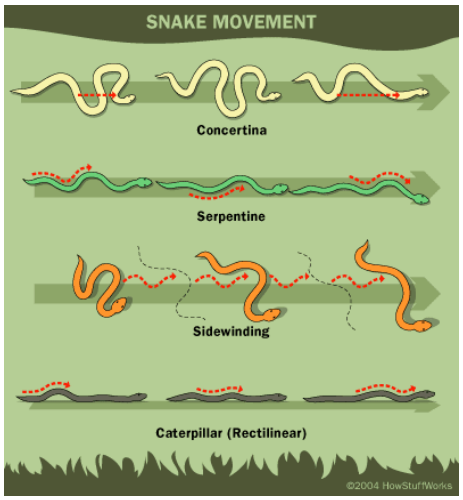


Figure 1: Snake Movement

Physics

Movement through granular media has long been a research interest among the fluid dynamics community, but so far little is known about it. Unlike movement through a fluid, such as water, there are additional variables to consider. In particular, it is dependent on the shape of the grains that make up the medium, and on the material that the grains are composed of. The laws and equations that govern motion through other types of media may not be applicable, since they are based on different assumptions (i.e. that the particles are small enough so that the medium acts like a fluid). Research in this field is starting to become more active, in particular among the biophysics community, since they are trying to seek answers to the questions on how organisms propel themselves through various media.

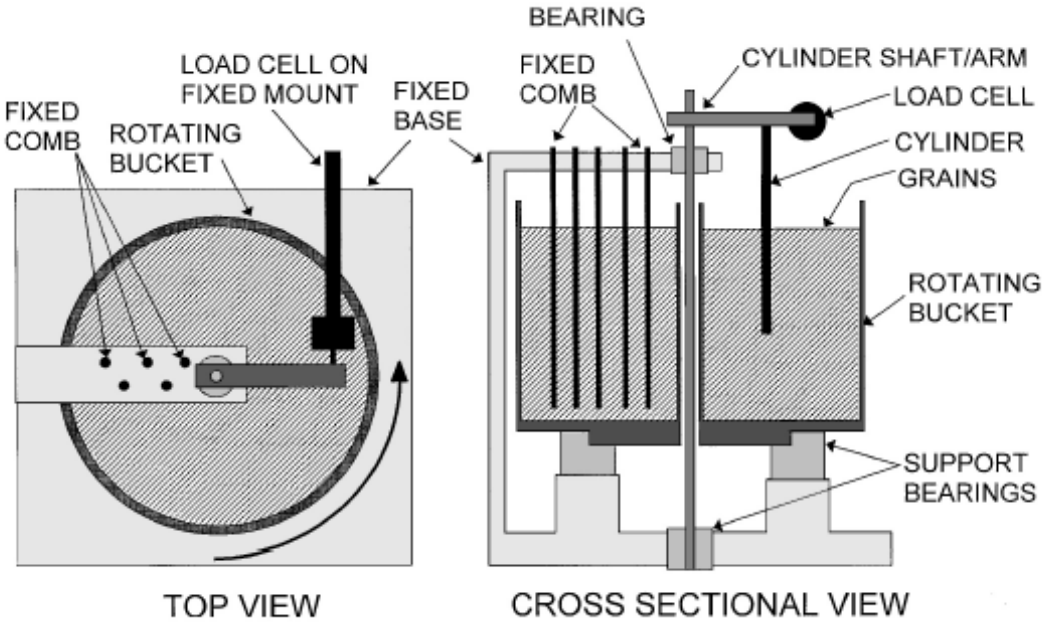


Figure 2: Figure from Schiffer's paper (Schiffer, 206) showing the experimental setup. The bucket is filled with glass beads with diameter d_g, and the cylinders are mounted downwards into the rotating bucket. The load cell would take force readings.

The purpose of the physics portion of the MQP is to experimentally verify the laws of motion through granular media, in particular through sand. This is important because it will be needed in order to properly determine the force, torque, and power requirements that the robot snake will need as it moves through the medium. It will be a useful model for how organisms, in particular snakes, move through sand in general.

There has been some research done on movement through granular media, most notably by P. Schiffer. Over a decade ago, Schiffer and his team conducted various experiments in order to determine the drag forces F_d that are exerted as objects are pushed through a granular medium. In the first experiment, his team studied the drag forces exerted on objects as it moved through the material at slow velocity (Schiffer, 206). The methodology consisted of taking vertical cylinders of various diameters d_c, and then extending it into a rotating bucket containing granular particles a certain depth H.

The granular particles were glass spheres with various diameters d_g, some of which are on the order of 3.0 mm (Schiffer, 206). The largest particles were nicely polished, while the smaller ones were not, and the velocity was limited to:

$$v < \sqrt{2gd_g}/10$$

where g is the acceleration due to gravity (Schiffer, 206). What they found was that the drag force increased with the cylinder diameter and immersion depth) increased exponentially with depth, as shown in Figure 2 (Schiffer, 206). They also found that it takes a minimum force F_T before the cylinder can move, as the grains locking up motion need to be reorganized to allow movement (Schiffer, 207). Schiffer reported the values of the prefactors as η = 4.32 (△), 2.83 (◆), and 2.43 (■, □, and ●), as shown in figure 3.

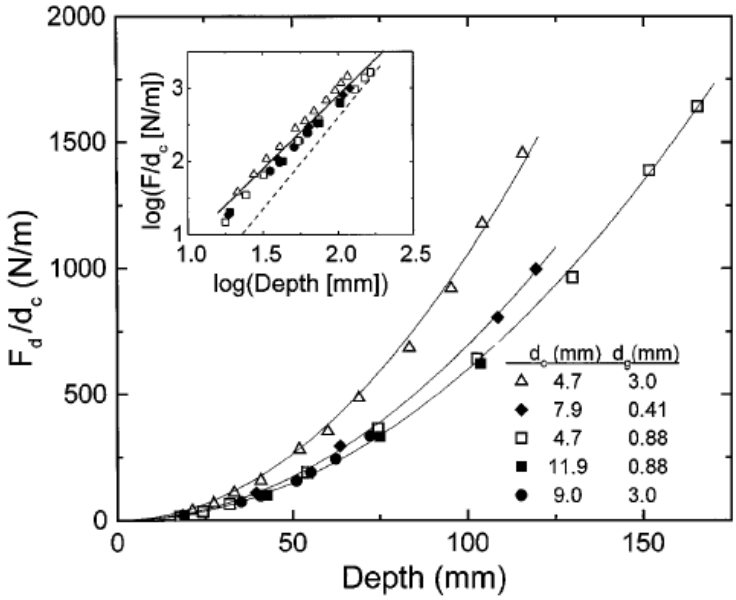


Figure 3: Figure from Schiffer's paper (Schiffer, 207), showing the results of his experiment and displaying the depth dependence of drag force.

In the second experiment, Schiffer and his colleagues studied how different geometries moved through the glass bead media. For this, they used five different geometries, consisting of a

disk, sphere, teardrop, cone, and a hemisphere (Schiffer, 1). All geometries were 25 mm in diameter (Schiffer, 1). In the case of the hemisphere, it also had a minor axis, which was about 15 mm long (Schiffer, 1). The speed of the movement was kept at a constant low velocity, at about 0.2 mm/s, since very low speeds do not affect the drag force (Schiffer, 1). They were attached to rods of varying diameters, so that they could be slowly dragged through the medium.

In the case of a vertical extended object (like a cylinder), the predicted drag force was:

$$F= \eta\rho g d_c H\text{^}2$$

where η is the prefactor, ρ is the packing density of the glass beads, g the gravitational acceleration, and d_c and H were the diameters of the geometries and the immersion depth, respectively (Schiffer 2).

For objects with a circular cross section, the force could be described as

$$F= \beta\rho g d^2_{obj} H$$

where β is the same as η, and d_{obj} is the diameter of the geometry (Schiffer, 2). In this experiment, He did, however, plot the depth dependence of the drag force, as seen in Figure 4.

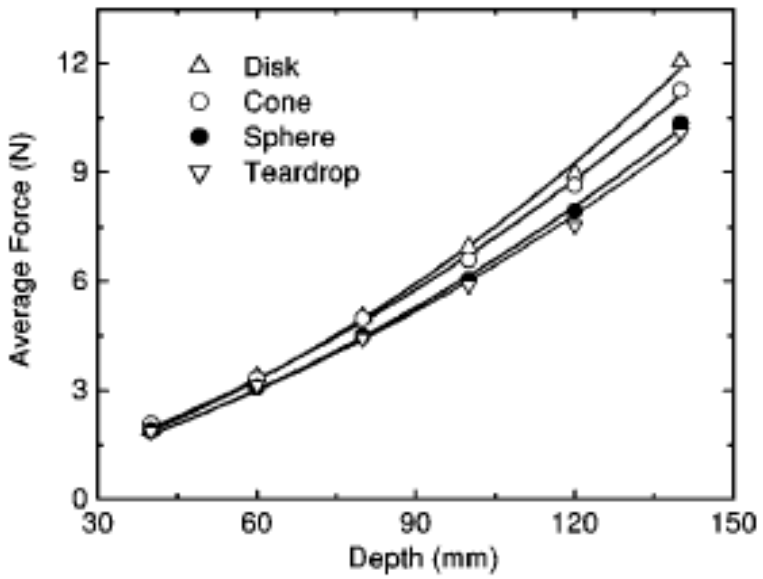


Figure 4: Figure from Schiffer's paper (Schiffer, 3), showing the dependence of immersion depth of the drag force.

In summary, Schiffer's experiments showed that the drag force depends on the frontal surface area, the area that is projected along the direction of motion. He also showed that the drag force is also dependent on the immersion depth, and the density of the granular media. Schiffer experimentally concluded that the effects due to the surface friction are negligible, and so it didn't have to be taken into account.

Previous Robots

Georgia Tech Sand Swimmer

In 2009, researchers at the Georgia Institute of Technology "CRAB lab" began investigating the motion patterns of desert reptiles and bugs. Of particular interest to them was the *Scincus scincus*, otherwise known as the sand fish, a lizard like organism that buries itself in the sandy surface of its desert habitat. In the process of burrowing, the sand fish adopts a sinusoidal motion and swims to its final resting location. In an effort to better understand this motion, the research team developed a mathematical model for mimicking the sand fish's

motion. Encouraged by the success of their model, the team built a 35-centimeter-long sand swimming robot, made from seven aluminum segments linked by six motors, and it was clothed in spandex to prevent the motors from becoming jammed ("Sandfish", 2). The snake robot was later submerged in 6mm diameter plastic beads. Utilizing their mathematical model it was able to traverse a distance of 0.3 body lengths per wave cycle. This proved that their concept was valid, and has led the way for further investigation into subsurface wave motion.

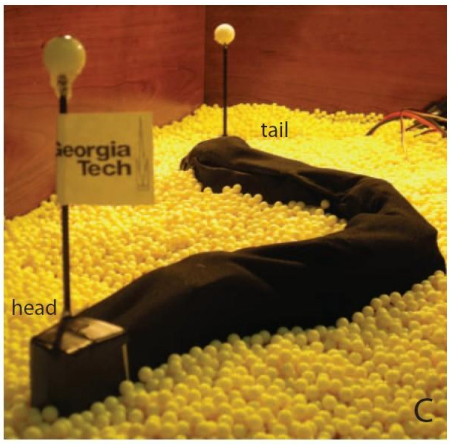


Figure 5: Georgia Tech's Sand Swimming Robot

WPI Sand-Snake

In 2009, Worcester Polytechnic Institutes' robotics and physics departments began a joint effort to investigate how locomotion through granular media could be achieved with various wave motion patterns. As part of this joint effort, the team designed and constructed a biomimetic snake robot to act as a testing platform for further research into the topic area. Upon testing the fabricated mechanism, several design flaws were brought to light that made the snake difficult if not impractical to use for the intended testing procedures (Humphrey, 20-60).

Despite an intensive and thoroughly conceived design process, the robotic mechanism constructed had several design flaws that prevented it from being used. The first flaw was that the motor torque was insufficient to overcome the static forces on the snake chassis while

submerged in the granular bed. The next issue was that the internal electronics boards within each of the robot's 11 powered sections were not sized to provide proper current pathways, and thus the copper traces tended to overheat and break down. Additionally, despite the chassis being thoroughly designed to resist fracture from motor stress, the reality was not so forgiving. The shear properties of the material varied depending on the direction of shear, and motor stresses still occasionally created fractures. This project concluded with the successful creation of a testing apparatus and provided a wealth of information design aspects that had proven successful and lacking for future designs to take into consideration (Humphrey, 50-75).

Figure 6: WPI Snake Robot (Generation 1)

Previous System Analysis

Servo motors

Servo motors that were used in the old snake robot's section joints provided precise control and position feedback. Unfortunately, the servos utilized did not possess the power to

successfully overcome the frictional drag forces and move the snake through the media. As a result, they regularly suffered being locked in a stall and overheated frequently. One of the current project's goals was to find a better way to move the snake since the previous selection of components was inadequate. Servos were still considered for use in the snake. One improvement over the old design would be to simply have better servo motors. Some of the servo motors reviewed had issues that prevented them from being used, such as being too costly or too large.

Among all the servo motors that were reviewed, one brand stood out from all the others. The Dynamixel servo motors were considerably better than most of the other ones available. They provided much more torque, and they had much wider range of rotation. The Dynamixel motors were expensive, but they came in a wide range of prices and performance. Ultimately, what made the Dynamixel servos so desirable was the fact that all of them were shaped the same. This would make the snake extremely upgradeable. The cheapest of the Dynamixel motors could be purchased and put into the snake. Tests can be read to see how well these servos worked. Then, if need be, the more expensive Dynamixel servos could be purchased to upgrade and improve the performance of the snake. This would also work if the budget fell short of purchasing more expensive Dynamixel servos. The project may end with a snake that does not meet all the performance requirements but further projects with the newer budgets would be able to. They can take the existing snake and focus all the purchases on the better Dynamixel servos. The better, more powerful Dynamixel servos would be completely compatible with an already fully functional snake.

The Dynamixel motors were not used for several reasons. Although the cheapest were $50 each, we would need at least 11 of them. This would equate to $550 total. Another issue

was that torque output. Although the Dynamixel motors had similar torque ratings as some of the DC Motors that were reviewed it still needed a significantly large gear reduction to meet the 20 Nm torque requirement. With a gear reduction of this amount applied to the servos the range that the joint could bend would be significantly too small for the design parameters.

The last issue was the voltage input required by the motors that would be used in the snake is required to be able to handle 24 volts. Being a typical servo the Dynamixel motors could handle only 12 volts. The best way to handle this would be to have the servo motors battery powered, thus reducing the voltage requirements.

These limitations meant that motors other than servo motors had to be used. DC motors can continuously run allowing unlimited gearing. There are also alternative ways for position control and feedback that servos provide.

Methodology

Project Scope

Background research and study of the previous project provided the required information to properly set project goals. A review of previous generations of snake robots provided insight into the unique set of challenges necessary to overcome when designing them. It also provided some insight into how to surpass these challenges, and pursue the development of more sophisticated designs. Acquiring a level of competence in understanding the physics of granular media allowed for an estimation of the physical properties needed in a sand-swimming robotic snake. A broad review of actuation technologies gave a variety of options for propulsion to the snake. Likewise, an investigation into multiple control methods allowed a realistic approach to the project. Building upon this knowledge, a list of criteria was established to guide the project's development. These were as follows:

- Design a biomimetic robot that matches the basic qualities of a biological snake with consideration to available resources.
- Snake skin and musculature are highly specialized and refined organs. It is not possible within the scope of this project to replicate it. However, attempts will be made to retain degree of authenticity in the final design.
- To create a scalable self-contained robotic snake capable of being programmed to approximate an arbitrary traveling waveform. However due to potential problems, such as power requirements or instruction transmission the snake may need to be tethered.

All of this information, in conjunction with the project objectives and assumptions, allowed for the creation of a specific set of parameters and specifications.

Design Specifications

- Resembles the generalized form of a biological snake
- Body form factor should be trapezoidal promoting submersion in media
- Minimum of 12 segments
- Section length to height ratio is approximately 3:2
- Total length of snake should not exceed 1.22 meters (4 feet)
- Perform lateral undulated motion at a depth of at least 22 cm (9 inches)
- Joints should be resistant to media penetration
- Skin that keeps the granular media out of the moving joints without restricting movement
- Minimized form factor tether provides for power and external communications
- Scalable
- Commonality and modularity between each segment
- Minimum runtime of 20 minutes
- Data Collecting
    - Orientation
    - Joint Angular Positions
    - Torque
    - Internal Temperature

Operation

- Must be safe to use for someone skilled and trained in its operation (pinch points etc)
- Must have safety redundancies to minimize the affect of human error
- Cannot use or require anything toxic or hazardous to human health
- Capable of being programmed or controlled easily such that motion parameters are easily changed and recorded, are able to match those of real snakes.

Manufacturability

- Commercially available materials
- Off the shelf component preference
- Able to be manufactured using standard techniques

Resources

- Cost less than $2000

Modification of Theory

As the MQP progressed, it became increasingly clear that Schiffer's model was incomplete. It started to become obvious that, at least when submerging parallelepipeds, the friction on the sides did effect on the outcome of the experiment. As a result, a proposed modification to the theory has been made for the purposes of this project:

$$F = \rho g \sum_i \mu_i \alpha_i A_i \langle z_i \rangle$$

In this equation, μ_i is the coefficient of friction, α_i is the pressure prefactor, A_i is the surface area, and $\langle z_i \rangle$ is the average immersion depth, all on the i face. The constants are the granular media density ρ, while g is the acceleration due to gravity.

Testing Methods

Figure 7: The experimental setup. The load cell is attached to the parallelepiped with a string of negligible mass.

The particles of the granular media, which were cylindrical and made of plastic, had an average length of 4.2 millimeters, having a width of 3.1 millimeters. The density of the granular media was ρ = 660 kg/m^3. In order to construct the geometries, LEGO bricks were used. In addition to testing various form factors, tests were conducted to test the change in motion based on varying friction forces. These forces were tested by selectively adding sandpaper of grit 60 to the block faces. The coefficients of friction for the LEGO bricks on the granular media was ρ = 0.45, and ρ = 1.4 for sandpaper.

To conduct the first test, several parallelepipeds were constructed from LEGO bricks, each of them with varying dimensions. A plot of the parameter space, shown in Figure 8, was made in order to ensure a good range of sizes.

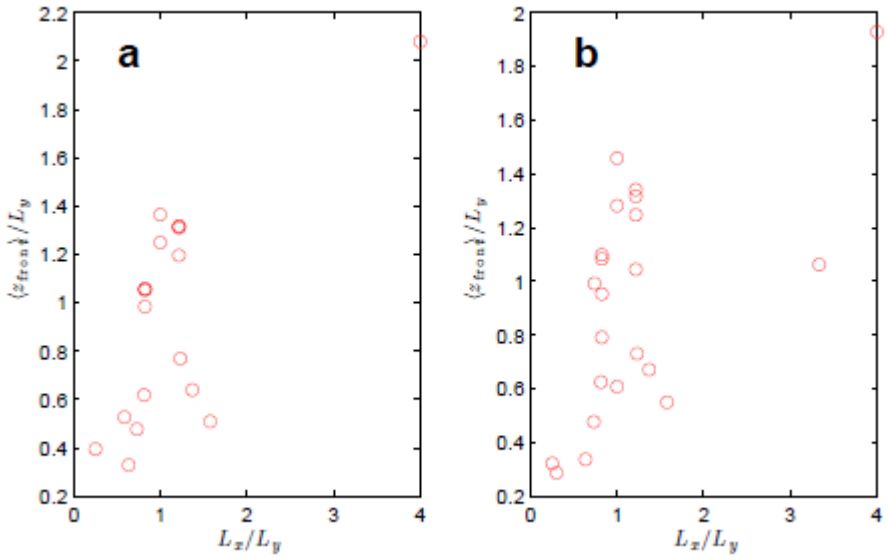


Figure 8: The parameter space for various geometries. The parameter space for smooth sides is on the left (plot a), while plot b displays the ones for rough sides (i.e. sandpaper).

A metallic pole was attached to each of the parallelepipeds, as shown in Figure 7, so that the load cell could be attached. The load cell used was a spring gauge, and was rated at 20 Newtons. For a couple of experiments involving really wide geometries, a 50 Newton load cell was utilized.

After attaching the load cell with a string with negligible mass, and partially immersing it, the parallelepiped was then pulled across the media. It continued to be pulled until the media stopped rising, and reached equilibrium. This coincided when the force reading on the load cell stopped fluctuating. Measurements were then taken on all sides to determine the final immersion depth of each of the faces. Afterwards, either the geometry was rotated, or a different one was used, and the same process was repeated for the next 47 trials. Sandpaper was applied on the

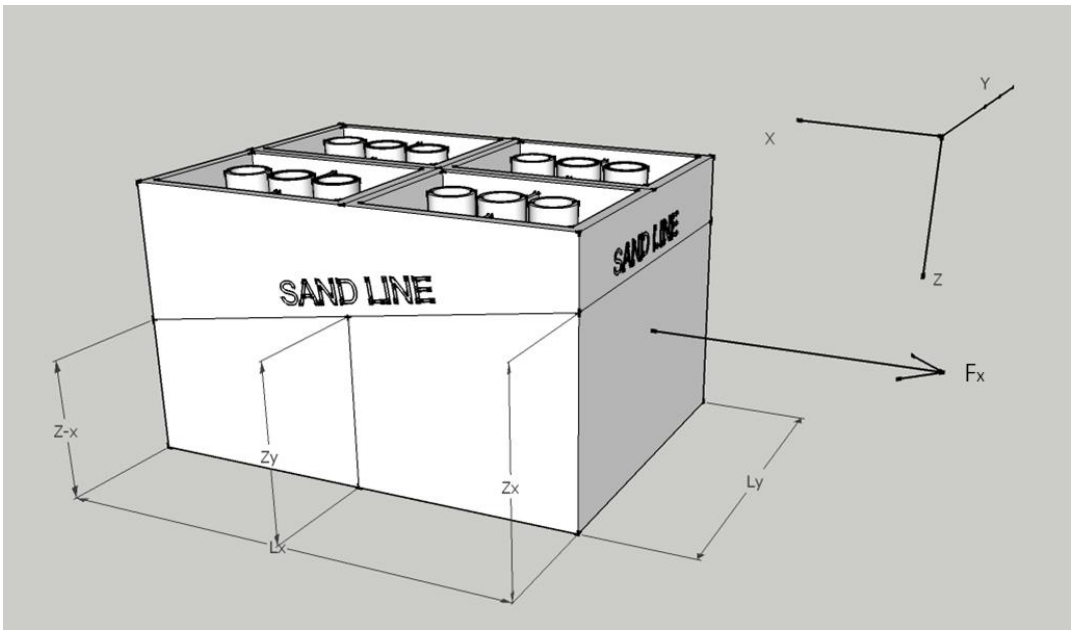
front, sides, and bottom so that it could be observed just what effect, if any, this had on the drag force. 

Figure 9: A detailed diagram of the parallelepiped while partially submerged in the medium. F_x indicates the direction of the force.

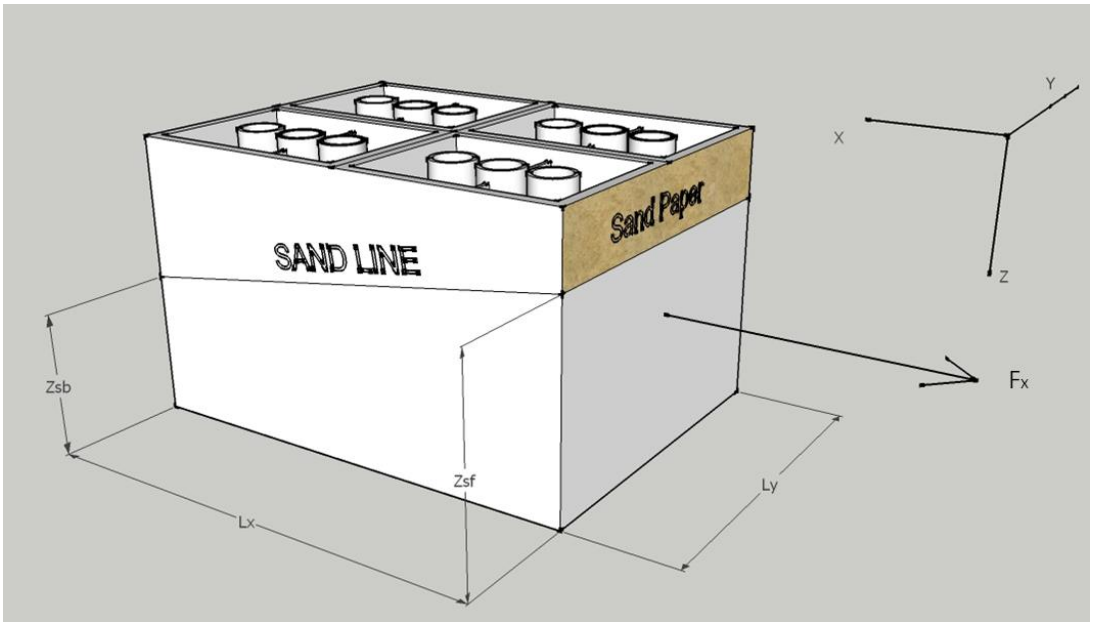


Figure 10: A detailed diagram of the parallelepiped, with sandpaper on the front, while partially submerged in the medium. F_x indicates the direction of the force.

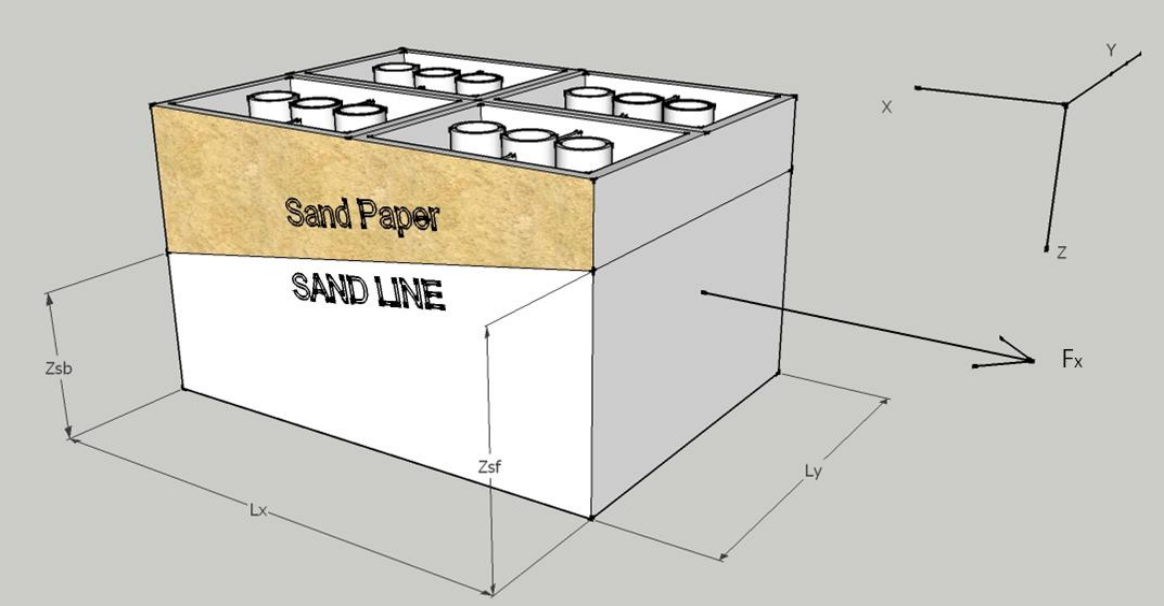


Figure 11: A detailed diagram of the parallelepiped, with sandpaper on the sides, while partially submerged in the medium. F_x indicates the direction of the force.

Design Specifications Discussion

When designing new products, it is necessary to investigate pre-existing theories and technologies as a means of advancing the goals of the design process. Much like the preceding MQP, the robotic snake is intended to mimic its biological counterpart. However, based upon lessons learned from the previous design, focus will also be placed on optimization of a body that will operate at depth for extended periods, while attempting to stay true to the anatomical properties of biological snakes.

Based upon the logical assessment of the previous project, it is unreasonable to attempt to fabricate a snake with the same near infinite degree of flexibility and scale as a biological snake. For scalability purposes, setting a minimal number of segments to be used in the robot would be appropriate. In order to define a minimal constraint on the number of segments, two considerations had to be taken into account. These considerations were the minimum desired waveform, and required resolution capability of the robot snake's wave approximation. Two full

periods of a sinusoidal wave was determined to be the minimum waveform necessary in order to achieve the desired resolution of 3 segments per half period. This implied that 12 segments were required.

Previous research has shown that the most efficient snakes have an overall length of less than 10-13 times the snake's circumference. Given the stipulation that the snake be no longer than 1.22 meters in length and in order to conform to the snake profile, a maximum dimension of 0.3 meters is set on the height and width dimensions of each robotic section.

Previous research has shown that biological snakes use a type of horizontal undulating motion when swimming in sand. Therefore the snake robot must, at a minimum, be able to simulate these same motion waveforms. As such, the robotic snake needs to be able to swim at a depth of at least 22 cm (9 inches) to ensure that it stays submerged throughout the experimental process.

Moving in granular media creates the challenge of requiring a sealed system to keep fine grains out of the moving components of the snake. In order to maximize the protection, the hard shell will form a complete barrier around these components to prevent the entrance of foreign material, while simultaneously allowing for ease of part manufacturability and assembly. Additionally, to cover any remaining exposed areas, a form of skin, such as spandex, will be required. This skin must remain flexible enough on the outside of the robot chassis to not add considerable restrictions to movement.

Based upon the outcome of the previous project, it has been deemed unreasonable to pursue a completely self-contained robotic system based on the limitations provided by the current battery technology. On the previous incarnation of the robotic snake, approximately 35% of the internal volume was allotted for power storage (i.e. Batteries). This impacted the

performance of the mechanical and electrical mechanisms two fold. First, smaller mechanisms were chosen to fit within the limited form factor, reducing the potential power output of each section. Secondly, given the nature of chemical batteries to steadily decrease current output as they become depleted, the snake's performance tended to diminish over prolonged periods of testing. Given the circumstances, it was deemed acceptable to trade off self-sufficiency and streamlining for an external power source through the use of a tether. While the tether does provide additional drag within the media, this can be compensated for by calibrating test data. Another positive attribute of a tether is that it eliminates a maximum operating time based upon power supply, allowing for virtually continuous operation. Additionally, the existence of a tether allows for the use of wired communication in the place of wireless. The internal volume reclaimed by these actions can be used to upgrade the drive system to a more powerful one and add additional sensor inputs and intelligence to each section.

The exact form factor required to optimize subterranean swimming motion remains unknown. Through long-term experimentation an optimal waveform may be determined, the exact properties cannot be determined in the short-term. Therefore, the snake needs to be designed in a scalable fashion to allow it operate with any number of sections. This stipulates that the snake should be able to accept an increase or decrease in the number of sections without any major overhaul to the system as a whole, within reason. The segments should be easy to add and remove, and the sequential placement of specialized segments should not affect the overall functionality of the snake.

In order to maximize the usefulness of the snake as a research tool, it needs to provide data logging capability for the sensor data it collects. The snake needs to be able to monitor and log the required current, angular position, and local temperature for each segment.

No researcher will make use of a tool if it is not reliable or safe. The final design of the robot snake must be capable of safe operation during routine functionality. It must be safe for use by someone trained and qualified to operate the device. To ensure the safety of the product, it must be designed in such a way to minimize the chance of injury. This includes designing in safety features for both the moving mechanical systems and the electrical components. Likewise the robot cannot make liberal use of materials that are generally toxic to biological life.

The goal of the snake is to be used for research purposes. Therefore steps should be taken to make the interface intuitive for an operator who has been trained in the robot's fundamental operation for testing purposes. The snake should be capable of being programmed or controlled easily such that motion parameters are easily changed and recorded and are able to match those of real snakes.

This project has limited resources in terms of budget, materials, and time. Therefore it needs to be designed and constructed with commercially available materials and components. Preference will be given to components that are off the shelf or require minimum modification. In addition any and all manufacturing should require standard techniques. This will allow future work on the snake to be performed with minimal effort.

The fabricated design budget to create the robotic snake is $2000. This means that each segment should be designed to cost no greater than roughly $166.

General Design Decisions

Having successfully defined the design parameters and project objective it was then possible to make assessments on the general design. These decisions further narrowed the scope of the project as to allow the engineering team to undergo the first stage of design work.

The first decision that was required was how to actualize motion within the robot. A variety of actuation methods were explored in the initial background research. Each method had its particular strengths and weaknesses relative to this particular application. Of all of the technologies studied, electro-magnetic motors were chosen for their high power to weight and cost ratio. Within the many types of motor actuators, careful attention was paid to motor output power, maximum torque, current load, and price. The vast array of possibilities was narrowed further by specifying a working voltage of 24V for the motor. This voltage was selected both for the availability of motors in this voltage range but also so that the maximum current running down the central power line would be minimized. Given these parameters, the HC315MG micro motor from Johnson Electric was decided upon as the best option. They offer the greatest functionality per size and cost of all of the options. This electric motor, like most, had a high rate of speed. However for the desired application much slower speeds are required. Attaining the desired output speed and torque was done via two methods. The first method was the implementation of a gear reduction, the specifics of which are discussed in the

The second method incorporated our electronics control system that created a closed loop control and implemented current regulation so that the direct output speed of the motor could be managed by the section processor. As part of this system to maintain a closed loop control over the motor, a custom electronics control system has been designed including a current measurement circuit and potentiometer for position monitoring.

There are countless manufacturing methods that would technically work for this type of project. However, only a limited number of methods are practical given the scope of this project. Methods that are most practical include standard machine tools such as milling machines and lathes. This type of manufacturing typically involves taking larger pieces of material and cutting

away from it to get desired shapes. The second method is to utilize sheet metal architecture in which the initial material is some sort of metal sheet, which is then cut to shape and bent to the desired shape. The final method is through the use of a rapid prototyping machine. Worcester Polytechnic Institute owns a fused deposition modeler (FDM). This technology allows for nearly any type of shape to be "printed" in three dimensions using a heated abs plastic filament. This in combination with an easily dissolvable support material allows for very complicated shapes, including overhangs and hollow sections, to be quickly and easily fabricated. After the design of the desired component is completed in a solid modeling program, it can be exported to the FDM which then prints it within spec. This method is the fastest, easiest, and most flexible of all available solutions. Its only limitation is that the printed abs plastic is only 60-80% the strength of standards injection molded abs and much weaker then metal counterparts. A table was created to compare the different resources available for this project.

Table 1: Manufacturing Resources Available

Resource	Capability	Time Required
Bridgeport Manual Mill	Milling	6 day turnaround time
Manual Lathe	Turning	Unable to cut desired components
Various HAAS Mills	CNC Milling	2-3 days turn around
Various HAAS Lathe's	CNC Turning	Unable to cut desired components
Dimension 1200ES fused deposition modeler	Rapid Prototyping	4 day turnaround time
Water jet	Cutting 2D shapes	3 weeks

Various hand tools etc	n/a	Eliminated due to time and reliability concerns

After an in-depth assessment of the available manufacturing resources a decision was made to use the rapid prototyping machine for the construction of the primary body cavity. The reasoning behind this decision is its ease of use, speed, and unparalleled flexibility in respect to imposed design constraints. Additionally, despite being comprised of hollow space, the component is rather large. As a result, the price of printing the material volume from a printer drastically outweighs the cost of a block from which the excess is removed. The main load supporting elements on either on the top and bottom side of the robot would be milled out of aluminum in the interest of speed and strength.

Control Methods

The goal of this project was to design a robotic snake capable of closely approximating arbitrary traveling waveforms. This task provides several challenges in the development of a control system. These include providing a user with the ability to develop a sequence of waveforms which are transferred onto the robotic snake, and then have the snake perform the desired motion. There is a wide spectrum of solution to these challenges. On one end of the spectrum would be to have a dummy snake completely controlled by a central computer node running through a tether. The other end of the spectrum would consist of numerous individual section controllers wirelessly networked together with an exterior director computer. The following will discuss the merits and feasibility of each technique.

Utilization of wireless communication can be accomplished via several technological methods. The easiest of these technologies to implement, due to availability and ease of use,

would be Bluetooth. Bluetooth utilizes a technology call frequency hopping, where the frequency through which it communicates is in constant flux. This technique allows for a fairly reliable form of communication in a technology that can fall victim to broadcast message gaps called shadow fading. While Bluetooth modems are relatively inexpensive and have been proven as reliable means of wireless communication, there are still several problems with them in comparison to wired communication. The first problem is that with the variable transmission rate, communication needs to be kept to a bare minimum in order to ensure that all necessary communication can be processed and transmitted when it needs to be and not bottleneck at the transmission point. The second problem is that as the distance between Bluetooth nodes increases, or the consistency of the material between the nodes becomes denser, the reliability and data transfer rate drop dramatically. The data rate is directly related to the distance and medium the signal has to pass through. Additionally, the transmission speed of even the fastest Bluetooth modem is slow in comparison to the data transfer rate available through a direct wired connection.

Utilizing a tether would provide a direct link to the robot from the outside and allow for the external supply of power and communication signals. As discussed previously, this option would remove the requirement for batteries and resolve issues of power consumption. On the down side, a tether is not the optimal testing solution since it adds to parasitic drag when the snake is submerged in the media. This parasitic drag has an effect on the resulting motion of the snake as it swims but it can be calculated and compensated for in the collection and analysis of test data. While added drag might imply that an un-tethered wireless version would be more desirable from a testing standpoint, it would add another layer of complexity in the form of batteries. Batteries require recharging and lead to unavoidable downtime. Battery power, if not

managed correctly will bleed off as the batteries are depleted resulting in a continually decreasing motor performance. Also, utilizing battery power limits the size of the drive components that can be loaded in the robot since space is a premium commodity and the more power needed to run a motor, the larger the battery supply must be.

There are multiple methods of exerting control over the snake. Having a singular "God" processor running the snake robot can prove problematic if the processor is not fast enough or doesn't have the internal memory required to process all the data being provided by multiple sections worth of sensors. In addition to the quantity of data being transmitted, there is the matter of how that data is transmitted. Having a singular processor running the robot equates to a large number of independent signal and power lines running along the length of the robot to the various sensors and actuators. This will likewise increase the size of the tether. The larger the tether, the greater the parasitic drag it creates. According to the project guidelines it is desirable to minimize this drag as much as possible. Therefore, utilizing the "god" processor seems counter-intuitive.

Independent section processors allow for a much more streamlined and efficient system in controlling the snake and organizing the sensor data. Each section, having its own processor to monitor and govern sensors, power, and motor output, must process fewer sensor interrupts than the "God" processor. This will result in more optimized and efficient control capabilities with each section. Communication between these individual processors and a "master" processor that correlates and directs the actions of the section "slave" processors can be attained through two wired methods. The first would be to have a transmit (TX) and receive (RX) line running between the master and each individual "slave" processors such that they could communicate openly at any time. The second option would be a shared communication line between all the

slaves and the master. This drastically reduces the number of wires that need to run through the umbilical but increases the complexity of the communication protocol since the sections must share the same TX line leading to the master.

In order for the robot to be able to interpret high level commands, execute the desired motions, and remain scalable; distributed embedded processing is necessary. This involves the master controller directing the movement actions of the slaves similarly to that of a conductor directing the actions of an orchestra. The slaves in turn act as dedicated processors for each segment and free up processing time. This results in more robust control and allows for more complicated instructions and more comprehensive sensory feedback from each of the sections.

One option for controlling the movement of the snake is through a discrete sequence of movements computed offline via MATLAB in terms of a lookup table which is downloaded onto the master controller. Alternatively, the master controller could be the research PC itself and thus simplify the transfer process of information from MATLAB to the section processors. The master controller would then communicate with each slave processor and inform them of their movement sequence. The moves would be choreographed by the master processor by broadcasting what position the snake sections should move to next at regular time intervals. The slave processors would then move to the appropriate location as indicated by the master's message. Having distributed processing in this manner would allow for greater flexibility in resource allocation. It would allow for each slave processor to log motor current, temperature, and position data. In addition it could monitor the supply voltage to ensure there is no problem with the electrical system and help localize the damaged area in the event damage does occur. This would allow for post-experiment data analysis and the ability to use real-time closed loop controls.

Communication between the master and slave processors can be accomplished in a variety of ways. One method would be serial communication. Serial is simple to use and fairly robust. However, serial communication requires the chain of communication to be unbroken so if one link should fail, operation of the rest of the snake shall cease as well, like electrical current in an open circuit. This can be avoided by creating an addressable serial, but much work would have to be done to achieve such a result. Another option is I2C or Inter-Integrated Circuit. The communication bus uses a clock and data line which can achieve speeds of up to four Mbps. I2C also has an additional benefit of allowing the joint processors to hold the clock line low putting communications on hold until the processor is ready to receive more instructions. Unfortunately, I2C is not conducive to working with a large number of section or over large distances. This is because multiple processors can attempt to communicate on the transmission line at the same time and garble each other's messages. Also, since the line is held low by many processors, the transmission distance along the line is not far, usually only a few meters. While I2C can attain data rates of over four Mbps, it is also not as high as other technologies. USART communication is a third option. USART is similar in many aspects to I2C but it holds the communication lines in a tri-state configuration. This allows for a greater effective range for communication and prevents multiple processors from talking on the transmission line at the same time. This makes USART communication highly attractive when utilizing shared communication wires between processors.

After carefully weighing all the options, it was decided that the robot would utilize a master-slave system with each section given its own processor and control circuit. These controllers would be linked in parallel along the same TX and RX communication lines to the

master processor using USART communication configuration. To provide ease of user interaction, the master processor would be the user interface computer.

A great deal of effort was put into the development of the section processors. In the last incarnation of the snake robot, there were an unexpectedly large number of electrical components that would fail while operating. This issue has been attributed to improper board layout as well as utilization of parts that were not optimally rated for on hand conditions. Learning from these experiences, great consideration was put into fault-tree analysis and preventative measures. As a result, numerous safeguards were designed into each section and along the primary power transmission line.

To optimize the effectiveness of identical electronics layouts in the robot sections, the slave processor is designed to be used in collusion with a physical dip-switch. This switch allows a binary declaration of the section position along the snake. This allows up to 16 sections to be included on the board at a time without any system overhaul. The embedded processor will check its location in the chain based on the switch setting every time it reboots.

Component Selection and Design

Motor

Table 2: Motor Comparison

Motor	cost	length	shaft diam	nom volts	stall amps	free amps	stall torque	free rpm	max power (calc)	Torque (calc)
		in					oz-in			n-m
GWS EM400 motor	5.99	1.466	0.091	7.2	32	0.001	16.3	19200	57.9	0.115
Speed 400 Motor	9.99	1.5	0.091	7.2	20.2	0.72	12	9200	20.4	0.085

(brushless?)										
GWS Speed 300 EM350 Motor	9.99	1.2	0.079	6	36	0.001	8.3	30500	46.9	0.059
ML-50 50:1 Geared Motor	26.95	2.24	0.236	12	3.3	0.115	320	120	7.1	2.263
19:1 Metal Gearmotor 37Dx52L mm	24.95	2.05		12	5	0.3	84	500	7.8	0.594
		mm					mNm			
HC385XLG-013		56	2.305	28	10	0.2	140	19000	69.6	0.14
QC381(0)XLG-001		56	3.175	36	16	0.32	292	18250	139.5	0.292
HF283LG-011		41	2	24	10	0.2	85	26000	57.9	0.085
		mm					mNm			
HC385XLG-27.5-46.5-002	17.37	56	2.305	28	10.8	0.197	149.5	18620	72.9	0.15
HC315MG-27.5-038-003	14.2	44	2.305	24	6.94	0.233	91.2	16933	40.4	0.091
HF283LG-24.2-036-001	11.83	41	2	26	10.7	0.245	85.3	28758	64.2	0.085

When looking for motors there are many specs to take into consideration. The important ones were the torque output, the cost, and the size of the motors.

The torque is the most important and has the most to discuss. It also relates to all the other specs. The torque is strictly in the design parameters. It needs to be as high as possible to

have the smallest gear reduction that can move the joints within the media. One of the issues with a high torque motor is that the higher the torque the larger the motor is. Having a large motor created two issues. The first is that a large motor is more expensive. The budget for the project was very limited and the cost of motors ate it up fast. Motors that were looked at ranged from 10 to 500 dollars per motor. With 11 joints that price would multiply to prices unaffordable with our provided budget.

The second issue with the size was simply being too big. The bigger the motor the higher the torque, naturally, but having a large motor will require a large segment to contain the motor. A larger segment will have more surface area resulting in more friction and displacing additional media while moving. As these parameters increase, so does the amount of motor output torque required to move the snake. Through experimentation and math modeling it was determined that the torque scales linearly in relation to section height and width. Any increases in section length, however, were shown to amplify the torque requirements by a factor of 4. This, in turn, increases the required torque. Also, one of the design parameters is that the snake does not exceed 4 feet. As a result, there is a threshold at which the motor is simply too big to used.

So these two issues bring up a new idea. What if we use significantly smaller motors? Smaller motors will be cheaper and require less torque to move the segments. There is a point however when the smaller the motor gets it starts to get more expensive but these types of motors were too small for consideration any ways. So now there was the debate as to what would be the smallest ratio of a motor's torque to the required torque from the segment size it would be housed in. It was a question of whether or not a smaller motor would reduce the gear ratio.

Other less important specs include motor speed, availability, shipping time, shape, and current draw requirements. If the motors speed was slow and it was geared down to increase the torque the snake would not move fast enough. The speed ratio is inversely proportional to the torque increase ratio through a gear train so that the more geared down a motor is the slower the output. There is a point where the snake would move too slowly to be practical. Some motors were out of stock or not enough of them were available for order of 11 or more. Some had an impractical shape that could not be easily implemented into the design.

After a while a new spec came into play. The voltage that the motor could run that became an important factor. Because the snake has 11 motors running along series electronics a lot of current was needed to power them all. The more current that is used requires larger wires running along the snake. Thicker wires meant that there was more resistance the wires had on the joint when it moved.

In order to bring down the amount of current the voltage needs to go up. So a new design parameter was created. The motors need to be able to operate with 24 volts. This limitation made only a small amount of motors qualify for use. A lot of the motors that used 24 volts were too big. They were not very common so some of them were too expensive.

Finally a motor was found that was small enough for practical use. The only problem was that it was an extremely high RPM motor with a low amount of torque. The motor would require a very large gear reduction to reach the desired torque.

One or two stages of gears would not be enough. A gearbox would be needed. One of the motors evaluate earlier had a very good gearbox on it. It was only a 12 volt motor but the gearbox could be removed. This gearbox could be used with the 24 volt motor. The only issue was that the gear box had different specs than the motor so there needed to be at least one gear

stage between the motor and the gearbox. The specs of the gearbox in the motor were compared in a spreadsheet.

Pneumatic/Hydraulic Joints

In the beginning of the project one of the main goals was to find a new method of moving the snake's joints that would be very powerful. One of the considerations would be to use some type of pneumatic or hydraulic system. Pneumatics and hydraulics use compressed air or fluids to move cylinders up and down. The force provided by them would be enough to move the snake through the media. Ideas came up to use of you and acts on either side of the joint, one side would expand will the other would contract. Another idea would be to use bag pneumatics. Instead of cylinders they use small bags that inflate. These bags would go on either side of the joints and operate in a similar manner. One bag would inflate as the other would deflate. This would also move the joints.

There were several issues with using pneumatics and hydraulics. In order for them to operate they require tubing to go to each end of the cylinders. Each joint would require at least four tubes with a total of 11 joints that could be at least 44 tubes along the snake. This would be too many tubes running through one area for the snake to move. A solution to this would be to have one tube running down the entire snake that provides all the compressed air. Then each segment would branch off to go to teach pneumatic cylinder. Still though this proved to be very complicated and still have the issue of the tubes have seen at the joint when the snake moved.

Another issue is that pneumatics and hydraulics require a housing that provides the compressed air or liquid. This may take up a lot of space and not fit anywhere conveniently on the snake. Also, if the hydraulics were to break anywhere the fluid would leak out. Pneumatics and hydraulics proved to be too complicated and impractical.

Gear train system

The old snake had a single stage of gears between the motor and the output joint. The pinion and gear that were used provided a gear reduction which decreased the speed and increase the torque. While looking for motors to be used for moving the snake several different types of gear configurations were considered. Worm gears provide a large gear reduction with a lot of torque. They changed the direction of rotation from the input. Worm gears are also not able to be back driven so they would be able to hold any position the snake would be at.

There were however several problems with a worm gears systems. Because of the angle that they use to translate power it was difficult to get them to fit into the design. When connected directly to the output shaft of the joint they would be exposed when the joint was bent to the end of one angle. Also the motors position would be offset from the center which would be more difficult to fit into the design. Worm gears also have very low efficiency. While a gear reduction on a worm gear system might provide a 100 to 1 reduction the output torque may only be 50 times as strong as the input torque. A worm and worm gear were ruled out because of these issues.

To get a high amount of torque output from the motors a gearbox could be used. Some gearboxes can provide a large gear reduction without losing a lot of efficiency. Some DC Motors come with a gearbox already attached to the end. Some of these gearboxes had planetary gears in line with the input shaft. Other gearboxes used a system of gears with the output shaft was parallel but not in line with the input shaft. Because of the extended gearboxes a lot of these motors were really long; too long in fact, to have the one dimension going vertically. So these motors needed to be lying on their side. The only issue what this is that the axis of the joint with vertical so the rotation of the motor needed change 90 degrees. To accomplish a bevel gear stage could be used. Bevel gears not only change the direction of rotation but they can also have a

gear reduction. So using bevel gears would solve two issues. They are also very easy and simple to implement.

At one point it was considered to manufacture gears to be used. This would theoretically save cost and allow any desired gears to be made. This, however, proved to be impractical because the machining tools available were not sufficient for making gears. Gear cutting requires special machining tools designed specifically for cutting gears.

The final consideration for gearing was also the same as the old snake's system. Having a gear and pinion stage to reduce the speed and increase the torque. This can be done with spur or helical gears. Helical gears are less noisy than spur gears but also less efficient so spur gears overruled helical gears.

Bearings, shafts, bolts, and Gears

Although this project aims to be a complete redesign of the old snake to a snake a lot of the old components were used in the design, more specifically the hardware. The shafts, bolts, nylon bearings, and brass bushings were chosen for use after haven been used on the old snake. As for the gears they are different but were purchased from the same company as the gears used in the old snake. They too came in a long single gear that could be cut up into many gears, as displayed in Figure 12.

Figure 12: Gear Chuck and Single Gear Cut From it.

Circuit Board

The circuit board was designed to include the following features: reference voltage sense, position sense, current sense, temperature sense, address selection, motor current regulation, and in-circuit programmability of the embedded processor. Board component layout was verified mathematically to ensure that the circuit would work prior to fabrication. The primary components were tested on prototype boards before moving to the fabrication of a custom printed circuit board.

Since the motors in each section operate at 24V and the control circuit requires a 5V power source, it was necessary to find a means to power both on the same board. Rather than connecting a second supply source and running extra umbilical lines for the 5V supply. The circuit board includes a voltage regulator able to except a voltage between 18V and 30V and output a regulated 5V supply. A voltage divider is also included in before and after this regulator so as to provide a means of referencing the source voltage for both the motor and the control circuitry. Additionally, to ensure that both the 24V and 5V supply lines are protected against power spikes, bumper capacitors have been placed on each line. The circuit is shown in Figure 13.

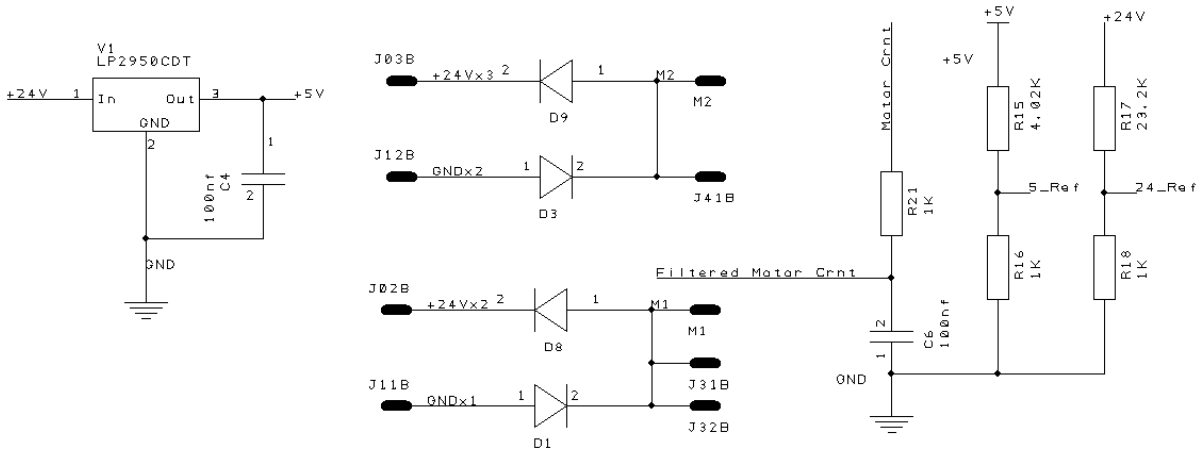


Figure 13: Power Management

Sensor feedback is critical to controlling and monitoring the robotic test platform. To provide independent closed loop control for the electric motor in each section, a current sense resistor was added to the output of the motor to provide a reference reading on the current passing through the motor. This in conjunction with a potentiometer placed on the output shaft of each section provides the necessary information for the motor to be driven in a closed manner. Providing accurate position data for the processor to control the motor is absolutely essential. As such, it was decided to use a potentiometer over an encoder because encoders are generally less efficient and require multiple trigger events to determine position and rotational velocity. The potentiometer selected is rated at 100,000 lifetime cycles.

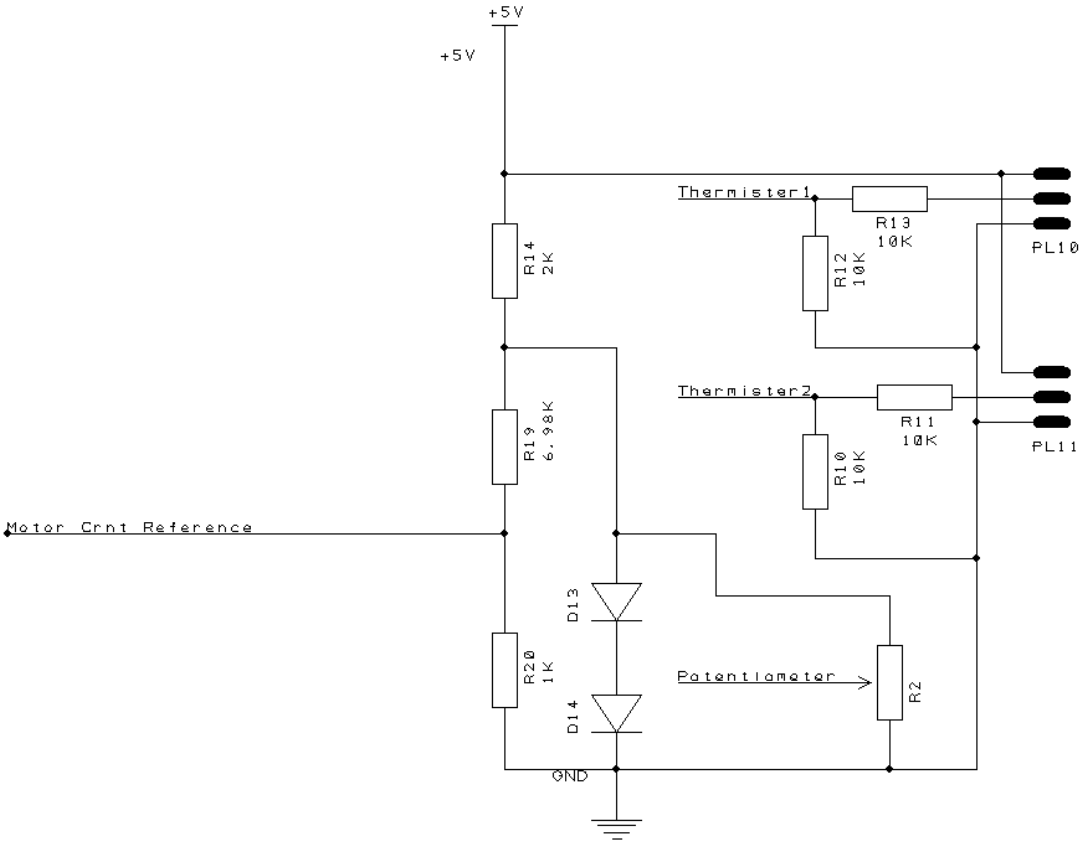


Figure 14: Sensor Feedback Lines

To collect the torque information for each joint, current sensing was added to each joint. To do this a sense resistor was added between the servo ground and the board ground. This converts the current going through the motor and the resistor into a voltage drop that can be measured by the processor. A current of 0 Amps relates to an output value of 2.5V and the current range of -5A to 5As is represented from 0 to 5V. The sense line was also filtered using a low-pass filter to reduce noise around the sampling frequency. The current sense circuit is demonstrated in Figure 14.

Many options for embedded processing were explored. These included various PIC processors and the DyIO integrated processor chip. Attributes selected as key attributes included pin count, chip size, price, and included attributes. After careful consideration of these factors, the ATmega328P processor chip was selected for use as the section processor. The ATmega 328P has a standard 28 through-hole pin layout. While there was a smaller scale surface mount component available, the through-hole was selected for ease of exchange in the event of processor failure. This minimizes maintenance down time. The chip has a comparatively small form factor, is reasonably affordable, and comes equipped with an ADC converter on 6 I/O ports. Additionally, the ATmega328P runs quite efficiently off of a standard 5V supply source. When selecting what section number the board is on, the input selector pins are driven high. A small dip switch was added to either leave the pins disconnected in a high state or to connect them directly to ground and drive them low. See Figure 15.

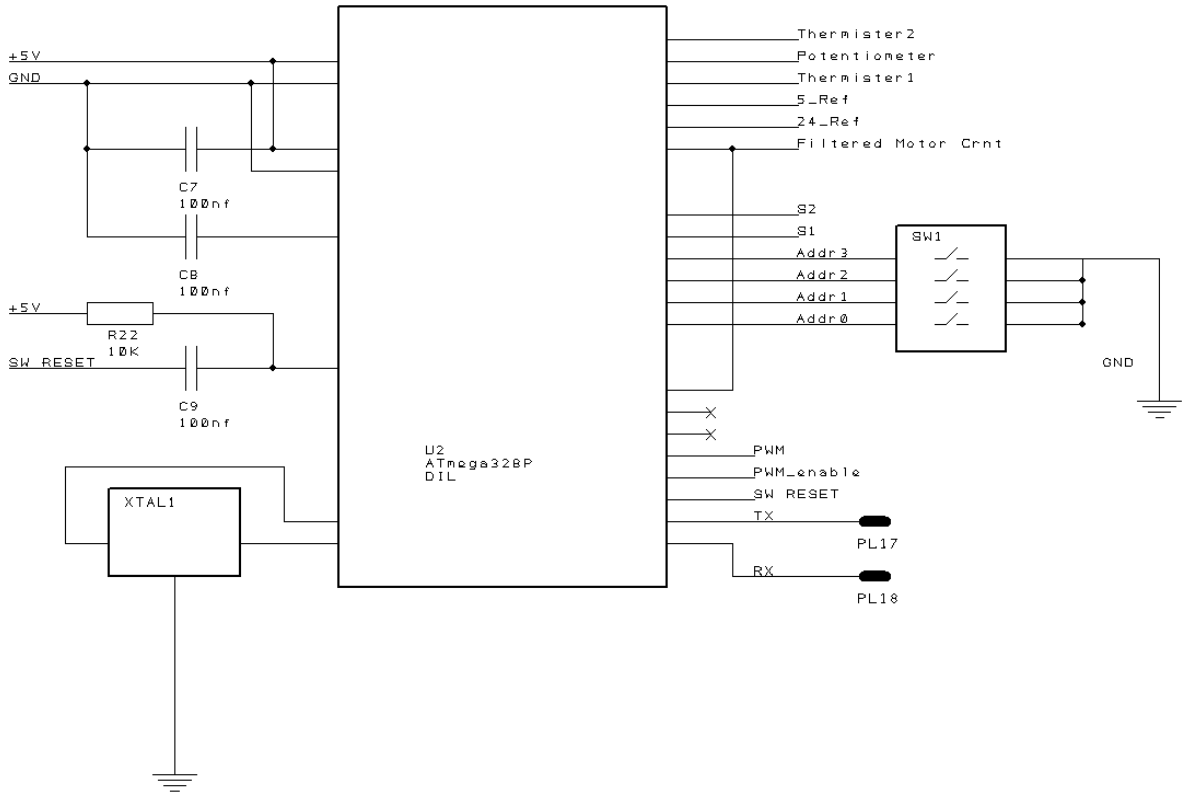


Figure 15: Processor, Crystal Oscillator, and Switch

The section processor controls the motor through the use of a full-bridge by controlling the S1 and S2 input lines. By pulsing the Full-bridge's enable line with a PWM signal the processor can regulate the current passing through the motor and thus regulate the output speed and torque. In addition, several safeguards have been placed on this circuit to ensure that damage cannot accidentally occur through technical fault in either the processor or full-bridge. In the event of the full-bridge failing while driving the motor, a thermal fuse has been placed in line with the 24V supply such that if current is overdrawn, the fuse will break the connection for a period of time during which the user will realize there is a problem. Additionally, to prevent mechanical damage from occurring in the event that the processor should fail while driving the motor forward, a dead-man switch has been implemented into the enable line. In order for the

motor to be operated b y the PWM signal the processor must be continually toggling the output of the PWM_Enable port. This port continually charges a capacitor that keeps the enable switch on. In the event that the processor should lock up and either holds the PWM_Enable line high or low, the capacitor will discharge and the dead man switch will disable the PWM signal line governing the motor. The full bridge is shown in Figure 16: Full Bridge and Relevant Hardware

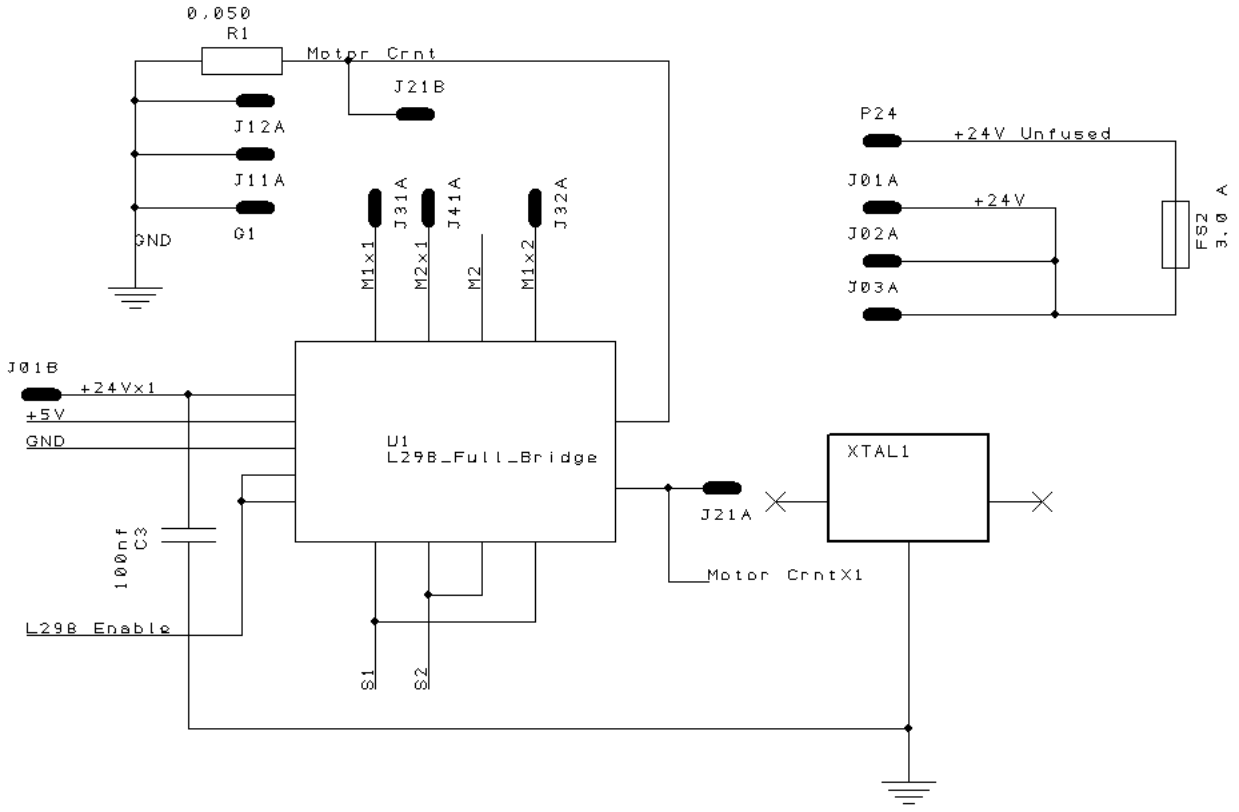


Figure 16: Full Bridge and Relevant Hardware

After each circuit component was designed and verified the printed circuit board (PCB) board layout was optimized. Basic design layout conventions were followed. Signal traces have a width of .010 inches while the power supplying the regulator is .025 inches to account for the increase in current. Likewise, the increased current running to the high powered motor is .05 inches. There are jumper points designed onto the board as well for external wires to be connected to connect areas that could not be traced due to the limitations of a two layer PCB.

While typically, as a rule of thumb the top-layer traces are horizontal and the bottom layer traces are vertical, this rule needed to be bent in order to optimize the traces on the board as best as possible. Also in addition to the circuits provided above the PCB also breaks out the necessary lines for motor control, serial USART communication, and off board sensors like the potentiometer and motor mounted thermistor. Figure 17 shows the PCB layout.

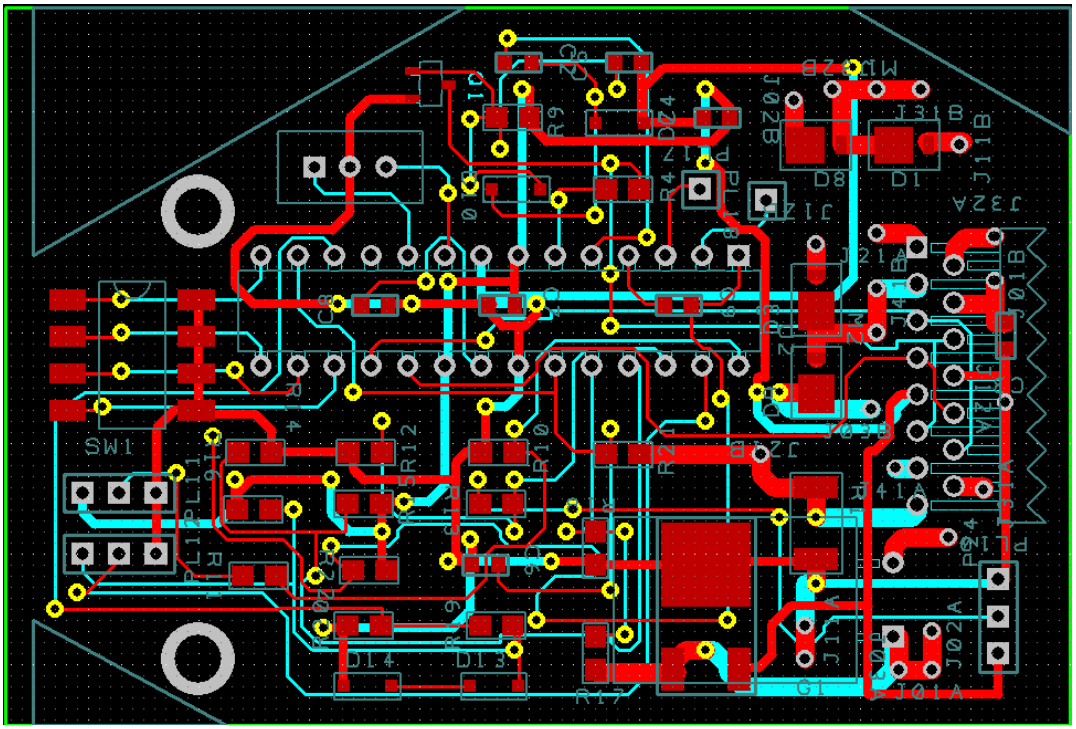


Figure 17: PCB Layout

Control System

A major factor in determining the success of the snake robot is the development of a control system to reliably guide the robot's actions and report back the sensory data for use by researchers. To this extent, the master-slave relationship was adopted between the user interface computer and the section processors within the robot. The purpose of the UI computer is to pre-calculate all the positions for the section controllers to move through given a specified waveform

to follow. This information would be stored within a text file. At this point, the computer's only tasks are to communicate at regular intervals the next desired angular position for each of the sections and then receive and display for the user the data values returned by the embedded processors.

The section processors in turn select the bytes of data in the computers position message relevant to their own numbered section and enter it as the desired position within an onboard PID loop to control the motor during the transition process. Meanwhile each section processor is regularly reading the values of the attached potentiometer, thermistors, and current sense circuitry. This data is used internal to the section in several safety features designed to prevent the robot from damaging itself. The data is also packaged at regular intervals in two byte pieces of code. These packets are then communicated with the similarly encoded section ID number back up to the UI terminal where they are decrypted and displayed as usable data for the research team.

Results and Analysis

Physics

In total, 47 experiments were undertaken; 19 for the geometries with smooth sides, 22 for the geometries with sandpaper on the sides, and 3 each for sandpaper on the front and the bottom. The results were then compared to Schiffer's model and with each other. The pressure prefactors on the frontal face were calculated to be $\alpha_f = 3.59$ for smooth sides, and $\alpha_f = 4.03$ for the sandpaper sides. For the side faces, the prefactors were $\alpha_s = 0.41$ for the sandpaper sides, and $\alpha_s = 0.87$ for the smooth sides. The prefactors on the bottom face were negligible for both cases. The modified theory shows a significant improvement over Schiffer's model, with 9 percent error on average for both the smooth and rough surfaces. By comparison, there was a 29 percent error for smooth surfaces, and 33 percent for rough surfaces, when using Schiffer's equation. When comparing smooth surfaces to rough surfaces, it was revealed that rough surfaces in general slightly increase drag force; for sandpaper it is increased by a factor of 1.08 on average, as seen on Figure 20.

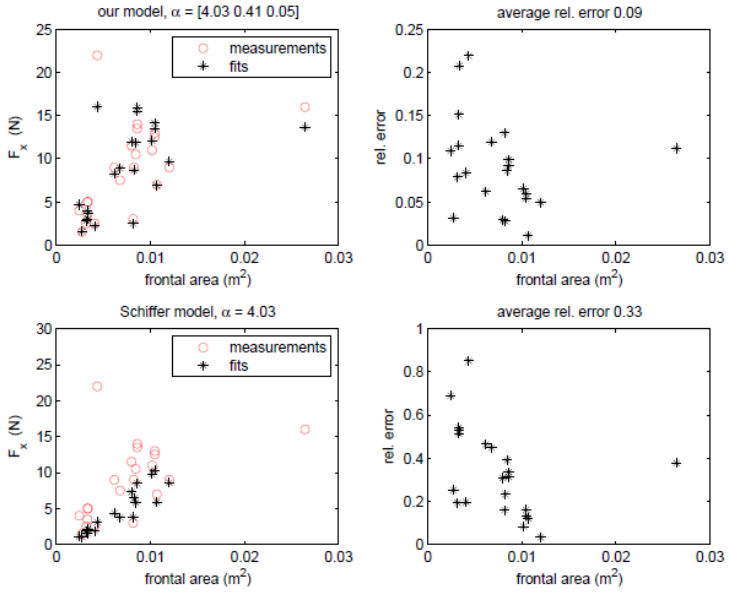


Figure 18: Results for the sandpaper experiments. The sandpaper is applied to the sides, and the experimental error is quite low compared with Schiffer's model.

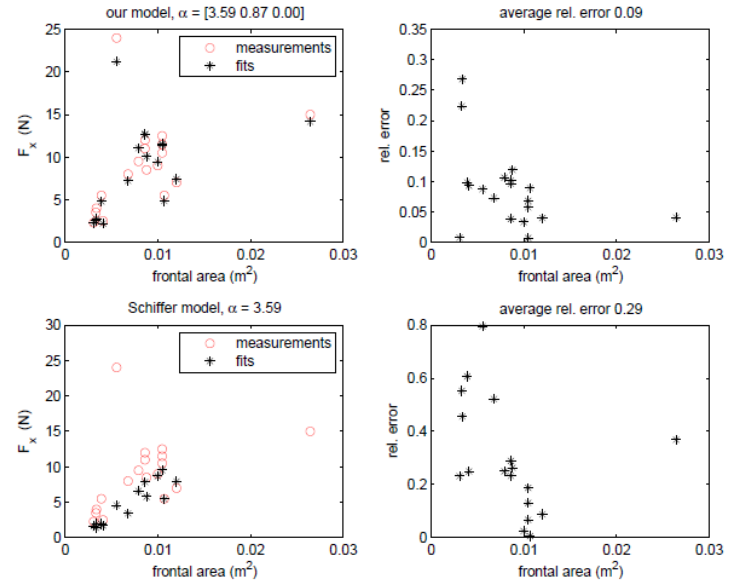


Figure 19: Experimental results for the geometries with smooth sides. As in Figure 8, the experimental error is quite low compared with Schiffer's model.

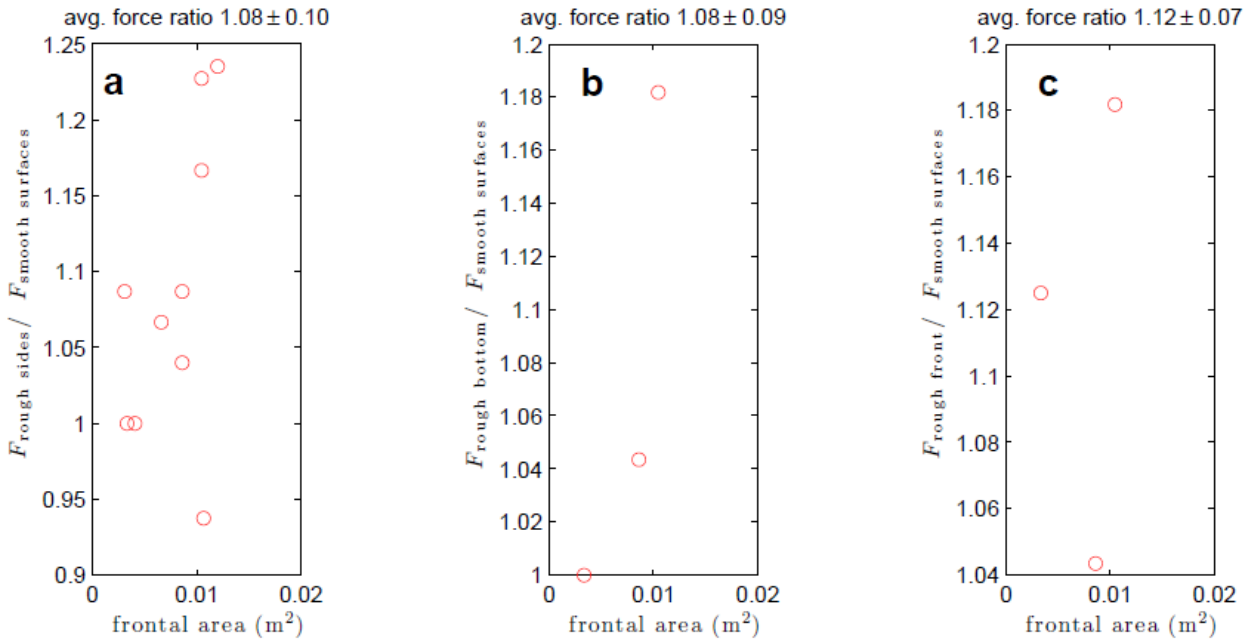


Figure 20: Average force ratio for the various trial runs. The left has the force ratio for sandpaper on the sides, while the plots in the middle and the right show the ratios for sandpaper on the bottom and the front respectively.

Discussion

The data shows that although Schiffer's original model could be improved on, it is quite unclear what effect, if any, the force due to friction has on the drag force. In Schiffer's case, he decided not to pay much attention to friction because it had such a negligible effect on the outcome of the experiment, no matter what geometry he decided to use. It would appear that it is also the same case here; namely that no conclusions can be drawn because the effects of friction appear to be quite small.

The other big mystery is the fact that according to the calculations made, the pressure prefactor on the bottom face was zero, i.e. $\alpha_b = 0$, thus implying that there is no contribution from the force acting on the bottom. Because it is unreasonable to assume that there is a negligible effect from the frictional force on the bottom, it is far more probable that our

instrumentation was not sophisticated enough to resolve the actual contribution of the surface friction from the other faces. Although this may make the modified theory suspect, it is important to note that the coefficient of friction has to be part of it. Otherwise the theory would be unphysical. Therefore, Ockham's Razor is inapplicable in this case, and friction cannot be discarded from the theory. More research has to be done in order to determine the true effects of friction on the drag forces.

Engineering Results

New Joint Design

The range that each individual segment on the old snake could bend from the neutral position was 45 degrees. This limited this snake's flexibility and kept it from achieving certain degrees of complex waveforms. There were two factors that limited his angle. The first was the servo motor that was used in the snake. Not only did the servo motor have a small angle of rotation, but it was also geared down. The range of motion was reduced by a factor of the gear ratio. This reduction however was necessary, for it not only reduced the speed of the servo motor, they also increased the torque. In fact, this torque was not even sufficient, because the servos would often stall and the snake could not move. Thus, the snake needed a larger gear reduction, one that would have limited the snake's joint bend to an even smaller range. With this limitation, it was decided that the DC motors be used instead of servo motors.

DC motors come in a large variety of sizes, speeds, and torque. But more importantly they can continuously rotate, so no matter how much of a gear reduction is used, the DC motors will still allow any desired angle bending at the joint. The only advantage that a servo motor has over a DC motor is that the position can be precisely controlled, and that they provide a feedback of their current position. In order to do this with a DC motor, an angle sensor is required. The

angle sensor will provide the same information as the servo, but this can be more difficult program and control.

The second thing that limited the old snake from bending too much was the pinching points. In order for the two sections to bend close to each other, part of the section needed to be removed. The larger the desired bending angle, the more material that needed to be cut away. And the more that needed to be cut, the larger the pinching point is.

The design parameters requested that the joint is as close to 60 degrees as possible. With each joint bent at 60 degrees in the same direction, the snake would be able to wrap around twice. This would allow a larger range of different waveforms to be achieved by the snake. DC motors allowed any angle range desired, as such, 60 degrees was certainly achievable. The only issue was in fact the pinching points.

In order to reduce the pinching point Different types of joints were looked at. One type of joint is the ball joint. The part consists of a ball that is inside of a socket that completely encloses it, thus it doesn't have any pinching points. Ball joints also have 3 degrees of freedom, and are able to rotate in any direction. The snake however needs to move in only one plane, meaning only 1 degree of freedom per joint is necessary. With this in mind, the ball joint can be reduced to a "cylinder joint".

The concept of a cylinder joint is really quite simple. Instead of the ends of each segment narrowing down towards the center of the joint, a half cylinder is placed at the end and centered at the pivot. Another advantage of this cylinder joint is that the snake maintains a consistent cross sectional area at any given angle bent at the joint. This new cylinder joint will maintain a rectangular cross section throughout the entire snake at every point during any angle bend.

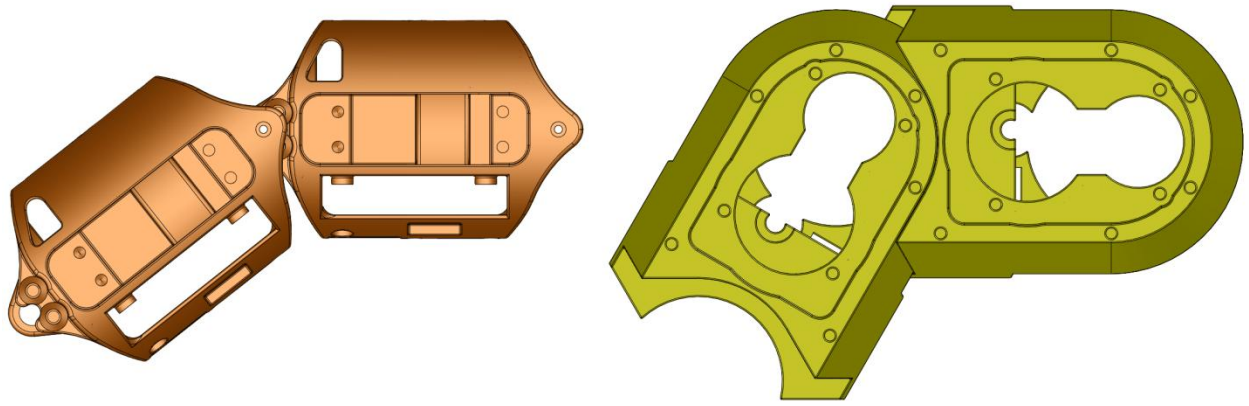


Figure 21: Comparison of the Old Joint to the New One with a Larger Angle and Smaller Pinching Point.

Final Design of the Segment

Ultimately, the Trapezoidal Ball Joint was used in the final design. Figure 22 shows the first sketch on the left, with its final extrusion on the right. It is clear that the sketch is far more complicated than the piece it represents.

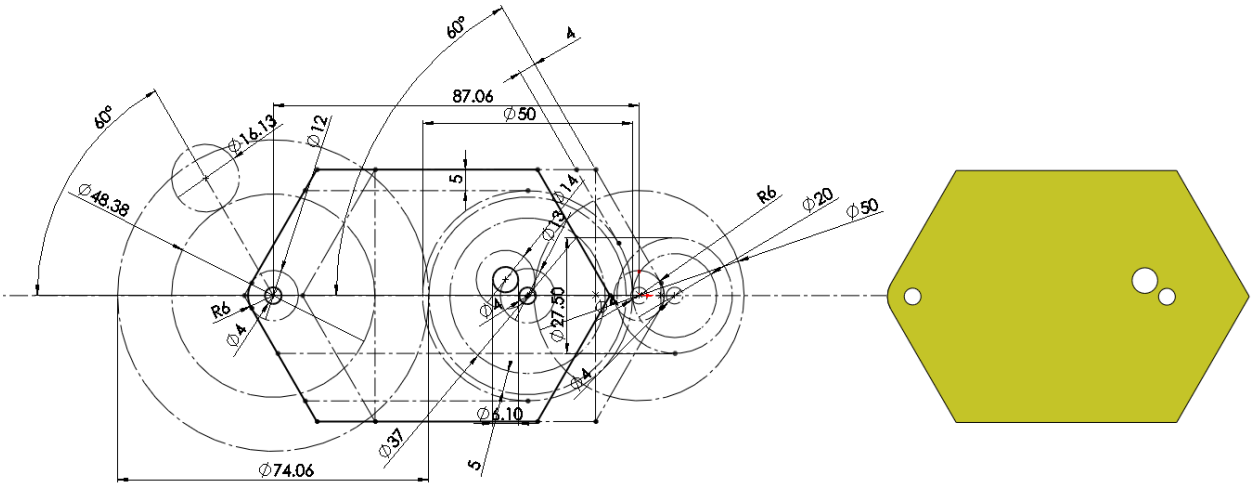


Figure 22: The First Sketch (Left) and its Extrusion (Right).

The reason for this complexity is that the dimensions were not specified. Instead, they were set to be the shortest length that was physically possible. Once all the components inside of the segment were decided on, this sketch was finalized. All of the dimensions in the sketch come from the different components housed in the segment. The sketch was set up so all the pitch

diameters of the gears were tangent. The rest of the circles represent the motor and gearbox components. Everything was arranged to take up as little space as possible, minimizing the size of the segment. The overall purpose of this piece is to house the output gears and determine the location of all the shafts in the segment for exact alignment.

Figure 23 displays the final segment piece.

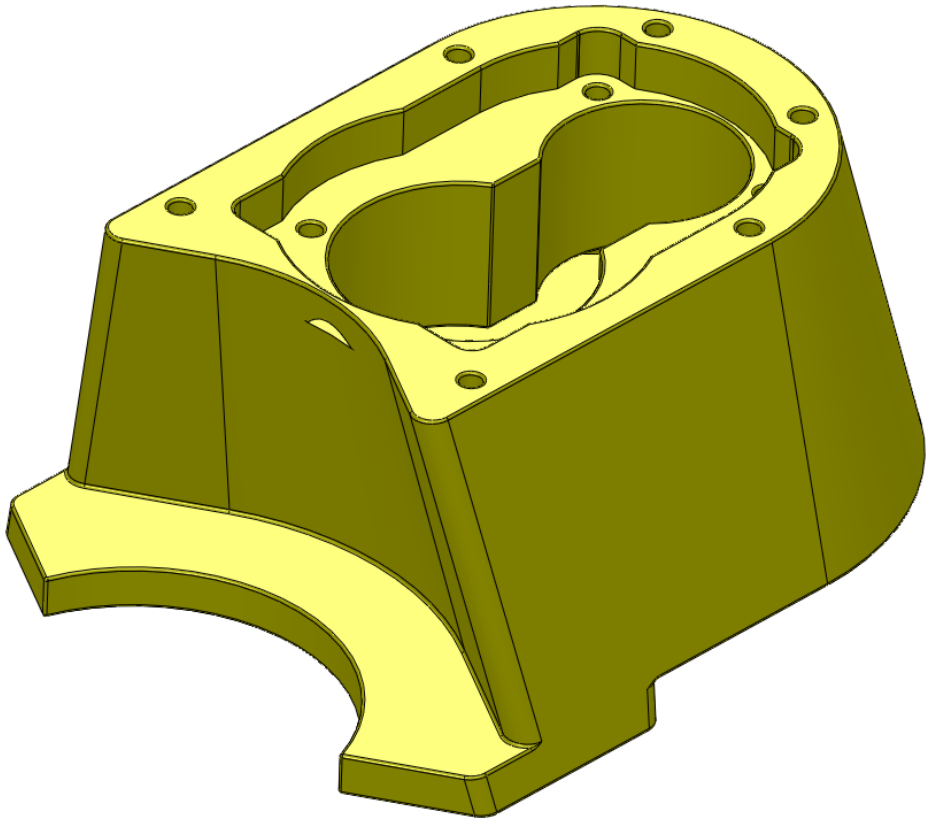


Figure 23: Final Segment Design

In addition to the central segment several more pieces needed to be designed. These pieces cover the top and bottom of the segment, as well as house the electronics. The pieces are to be made out of aluminum for strength and can be seen in Figure 24 through Figure 26.

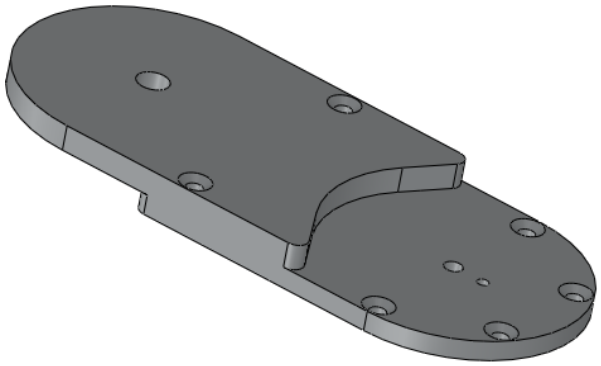


Figure 24: Final Design of Top Plate Created From Segment.

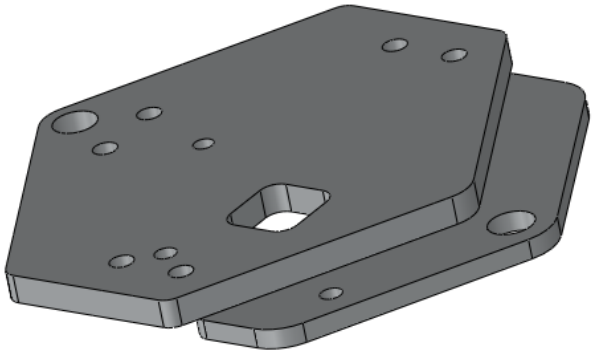


Figure 25: Bottom Aluminum Plate.

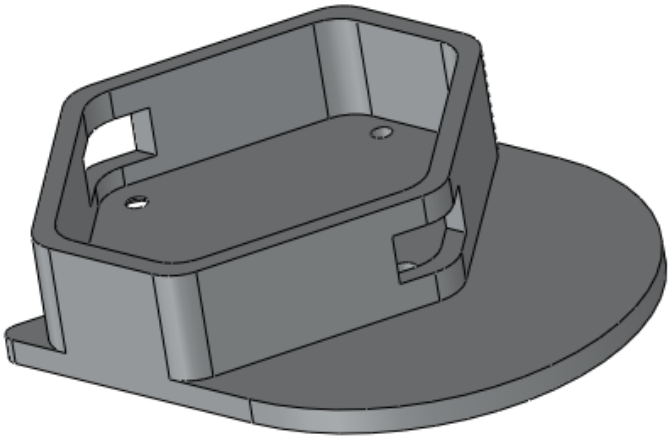


Figure 26: Bottom, Aluminum Plate to House the Electronics.

In the housing, it can be seen that there are square holes on the front and back. These holes are for the electronic components to pass between segments. They need to be as close to the joint as possible to keep them from flexing too much as the snake moves.

The final segment containing all the components can be seen in Figure 27. Its exploded view can be seen in Figure 28

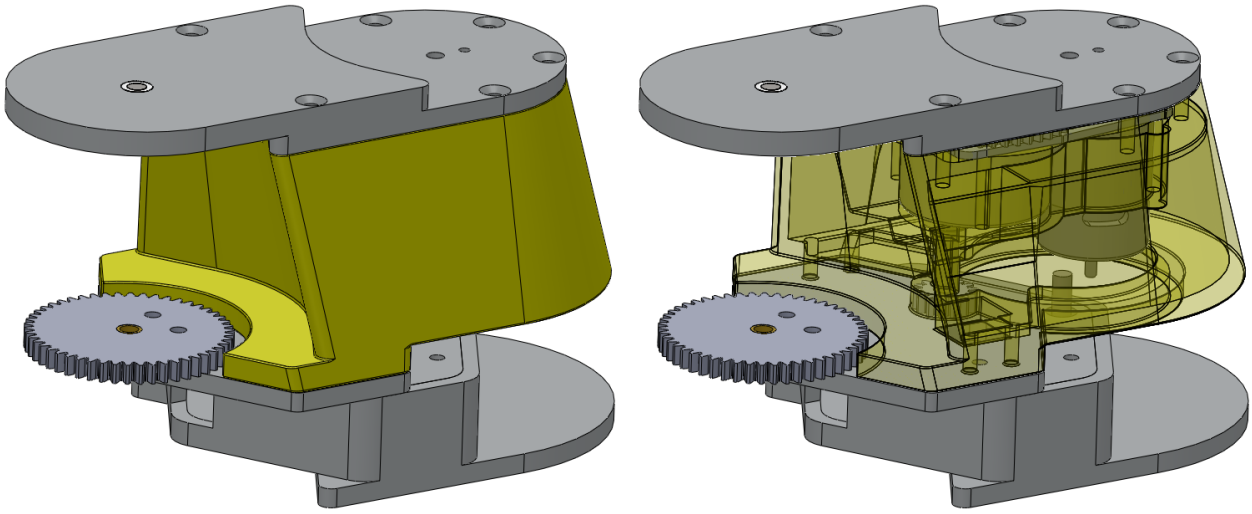


Figure 27: Whole Section and all of its Components along with a Transparent View.

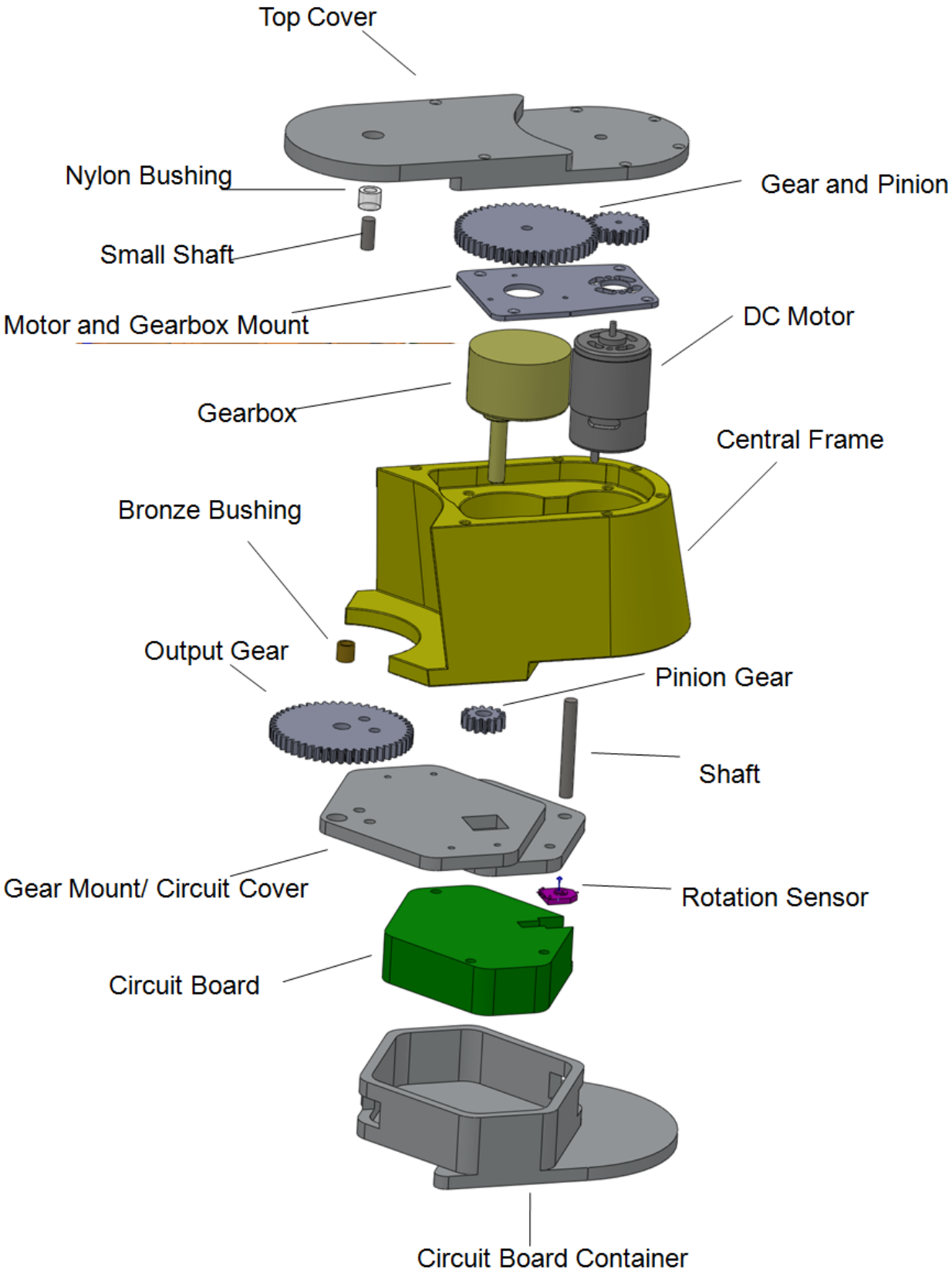


Figure 28: Exploded View of a Single Segment.

The final gear train is displayed in Figure 29. Each of the two pairs of gears and pinions had to be carefully designed in terms of size and gear ratio.

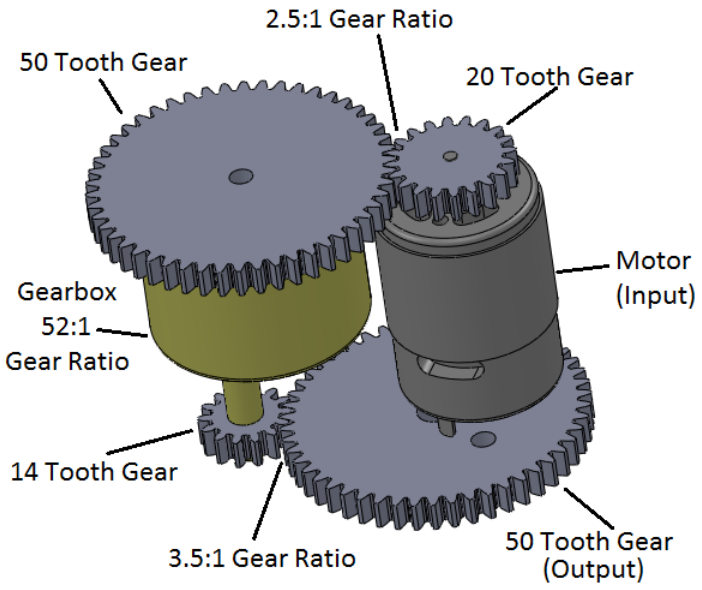


Figure 29: The Final Gear Train.

The most important gear ratio is the one between the motor and the gearbox, because the two have different specifications. The gear ratio needs to be large enough so that the motor does not exceed the maximum RPM of the gearbox, and it needs to be small enough so that it does not exceed the torque rating of the gearbox.

To calculate the minimal gear ratio between the motor and the gearbox, the maximum RPM of the motor, and the maximum input speed of the gearbox need to be compared. The maximum input RPM of the Tetrix gearbox can be found using the maximum RPM of the Tetrix motor with its gearbox attached, and the gear ratio. The maximum RPM of the Tetrix motor is 141 RPM. The gearbox reduces the motor's RPM so that the actual RPM is 52 times of the speed of the output RPM, with 52 being the gear ratio of the gearbox. This means that the final RPM of the Tetrix motor is 7332 RPM. The maximum RPM of the motor used in the snake is 16933

RPM. For this to be reduced to 7332 RPM, a gear ratio of at least 2.31 is required. 16933 RPM is actually the no load speed of the motor, therefore it is a good safety factor.

Now what was needed was an acceptable maximum gear ratio between the motor and gearbox. For this calculation, the maximum rating torque of the gearbox needed to be compared to the nominal running torque of the motor. This is because the motor will be limited to the maximum output torque it can provide, so that it does not stall. The gearbox was tested to operate up to 2.837 Nm. Again, the gear ratio is used to determine the allowed input torque. However, the efficiency also needs to be accounted for. The torque change is inversely proportional to the speed, so what happens is that the torque is reduced. With an efficiency of 71.4 percent, the gearbox can handle an input torque of 0.076 Nm$(2.837\ Nm/52)/0.714 = 0.076411334\ \text{Nm}$. The motor will be running with a torque of about 0.031 Nm. But in order to reach a torque of 0.076 Nm, a gear ratio of 2.46 is required. However, this does not take into account for the efficiency loss between the gear and pinion. Spur gears usually never have efficiency above 95%. With this factor, the maximum allowed ratio becomes 2.60. A lower efficiency would result in a higher allowed ratio.

So now the gear ratio cannot go below 2.31:1 and cannot exceed 2.60. A higher ratio is also desirable to have a greater torque output so a ratio of 2.5:1 was selected. The gears that were selected are a pinion with 20 teeth and a gear with 50 teeth. This equates to a gear ratio of exactly 2.5:1.

The final gear stage was a simple matter of what could work. The greatest possible gear reduction desirable was to get as close to 20 Nm as possible. For the pinion selection, the smallest pinion with a large enough diameter to fit onto the output shaft of the gearbox was selected. There were smaller pinions, but they simply were too small, so a 14 tooth gear was

selected. As for the gear, the same 50 tooth gear was selected. There was an option of using a 72 tooth gear, but this gear required that the entire segment exceed 4 inches, which was beyond the design limitations.

Electronics

The final electronics design entailed a complete overhaul and expansion of the capabilities of the section controller as well as rework of the wiring that comprises the robot's nervous system. Beginning anew, a bare bones circuit was designed to service the section controller, communication lines, motor, potentiometer, current sensor, and thermistors. From these base requirements, additional electronics hardware was devised in the form of selector switches, crystal oscillators, and Full-Bridges. This additional hardware was used in generating a control circuit for the motor as well instituting timer capabilities on each section. By this point in time, the motor voltage had been finalized as 24V. Since the PCB control components functioned off a 5V supply it was initially intended to run two power lines down the umbilical and through the spine to each section. In the interest of minimizing the drag coefficient of the umbilical it was in the best interest of the design team to run as few wires through the umbilical as possible. As a result of this decision, the 5V line was removed from the umbilical and its functionality was replaced by adding a 5V power converter to onto each section thereby allowing the 24V line to service both the motor and control components.

When calculating the copper area required to carry the expected current load going to each component on the board. The standard signal lines were easily laid out at a thickness of 0.3mm. The power lines supplying power to the motor proved problematic as they required upwards of 1.8mm of copper trace to supply the desired current without safety or depredation issues. When it was determined that with the size constraints of the robot it was not feasible to run traces for all of these connections, an alternative was devised. Jumper connection points were

set on the board. This allowed for external wires to be soldiered onto the board and reduce the current carrying load on the individual path ways. In doing so the trace issue was resolved. Components were ordered and tested on a bread board configuration at this point. Several of the electrical components ordered turned out to not be rated for the current load required, most notably the H-Bridge. As a result, research had to be done to find a replacement component that would handle the current load. Fortunately after some careful investigation replacements were found and purchased. The last issue that required consideration prior purchasing the final PCBs was the issue of safeguards. In order to fulfill the specification of designing a human safe product, analysis of the electronic fault tree was undertaken and several stages of safeguards including dead man switches, fuses, communications pulse monitoring were included to ensure the robot would never have a chain of catastrophic electrical failure in the course of normal operation.

Figure 30: Circuit Board (Final Product)

Social Implications

In order to be of any use to researchers in the future, the robot snake needed a human-friendly design. To this extent, the robot had to be designed in such a way as to ensure a safe working environment and reduce the risk of human injury resulting from its use. To minimize the threat of injury from contact with the snake, the electronics system was incorporated with safeguards to prevent the robot from acting in a manner that would cause bodily harm. Setting position boundary limits prevents the robot from over stressing its chassis and potentially breaking sharp pieces off. Additionally, current and temperature control prevent the robot from operating in a state that would cause serious shock or burns if it should be handled inappropriately. Hardware design was also undertaken with safety in mind. The physical chassis was designed to eliminate the pinch region present in the previous design model. Likewise, the gear system was enclosed within the plastic housing to prevent foreign materials from entering and getting caught by the gear teeth.

Recommendations

There are several recommendations for future work the robot snake to improve its functionality and capabilities. These recommendations include further development of waveform evaluation as well as continued development of hardware and software. The scope of this project was to develop a better understanding of the drag forces inherent in moving through granular media and to create a robotic platform for research.

For hardware, testing and perfecting the mechanical design to ensure structural integrity will improve the usability of the device. Additionally, applying a thin layer of rubber between the aluminum chassis mount and the back of the electronics board will aid in shock absorption and prevent short circuiting through electrical contact with the mounting section.

In addition it is recommended that the electrical contacts running down the spinal cord allowing for segment separation be checked for power loss and noise statistics. Should these factors be outside of desired limits then an alternative spinal system should be devised while allowing easy removal of sections from the snake body.

For software, a reliable MATLAB or like program should be instituted to generate the desired waveform tables to drive the snake. Currently the software is such that it can be used for testing, but the waveform generator is not functional. Additionally, some of the safety factors that have been enabled by hardware are not yet programmed into the code and therefore are not utilized as of this time.

With these changes the snake will be both far more robust and user friendly. Implementation of these recommendations will increase the overall effectiveness of the snake and provide for a configuration that is even more resistant to electrically stimulated malfunctions.

Conclusion

Physics

In light of these experiments, one has to conclude that in order to better understand better what is going on at the sides and the bottom, more sophisticated tools are required. LEGO's and spring-gauge load cells are a good way to get results quickly and cheaply, but they can be pushed only so far, particularly since the force readings often fluctuate. More sophisticated tools will better be able to determine what effect friction has on the drag forces. Despite the crudeness of the tools, there were some valuable discoveries made, and potential areas of further research found. Schiffer's theory is found to break down in instances when different geometries are used. Additionally, increasing the coefficient of friction on the body, in particular the sides, is shown to slightly increase the drag force. More research will have to be done in order to accurately determine the true effects of friction.

Engineering

A design was created for an interdisciplinary scalable snake. The snake is designed to consist of up to 16 powered joints, in addition to the head and tail segments. Initial testing was conducted validating the functionality of the electronics and onboard software. Utilizing Microsoft Visual Studio 2010 the snake was commanded to execute a specified traveling waveform step by step or continuously. The project encompassed a large number of challenges. Most notable of these challenges is the ever present lack of available time. With minimal additional time and resource investment in manufacturing, the snake will be fully capable of acting as a research platform for future studies of swimming in granular media. This project was a trying time for our group. One of our original team members, Ian J. Morse, became ill two

weeks before the end of term and was unable to help in the completion of the project. We also ran into some material acquisition issues upon reaching the construction portion of this project.

Works Cited

Dowling, K. (December 1997). *Limbless Locomotion: Learning to Crawl with a Snake Robot.*

Goldman, Daniel, Haldun Komsuoglu, and Daniel Koditscheck. "March of the SandBots." *Spectrum*Apr. 2009: 30-35. *Georgia Tech*. Web. 12 Dec. 2010. <http://crablab.gatech.edu/pages/press/SPEC_20090401_Apr_2009.PDF>.

Humphrey, Neal K., and Brian Benson. *Robotic Research Platform for Locomotion through Granular Media*. Worcester, MA: Worcester Polytechnic Institute, 2009. Print.

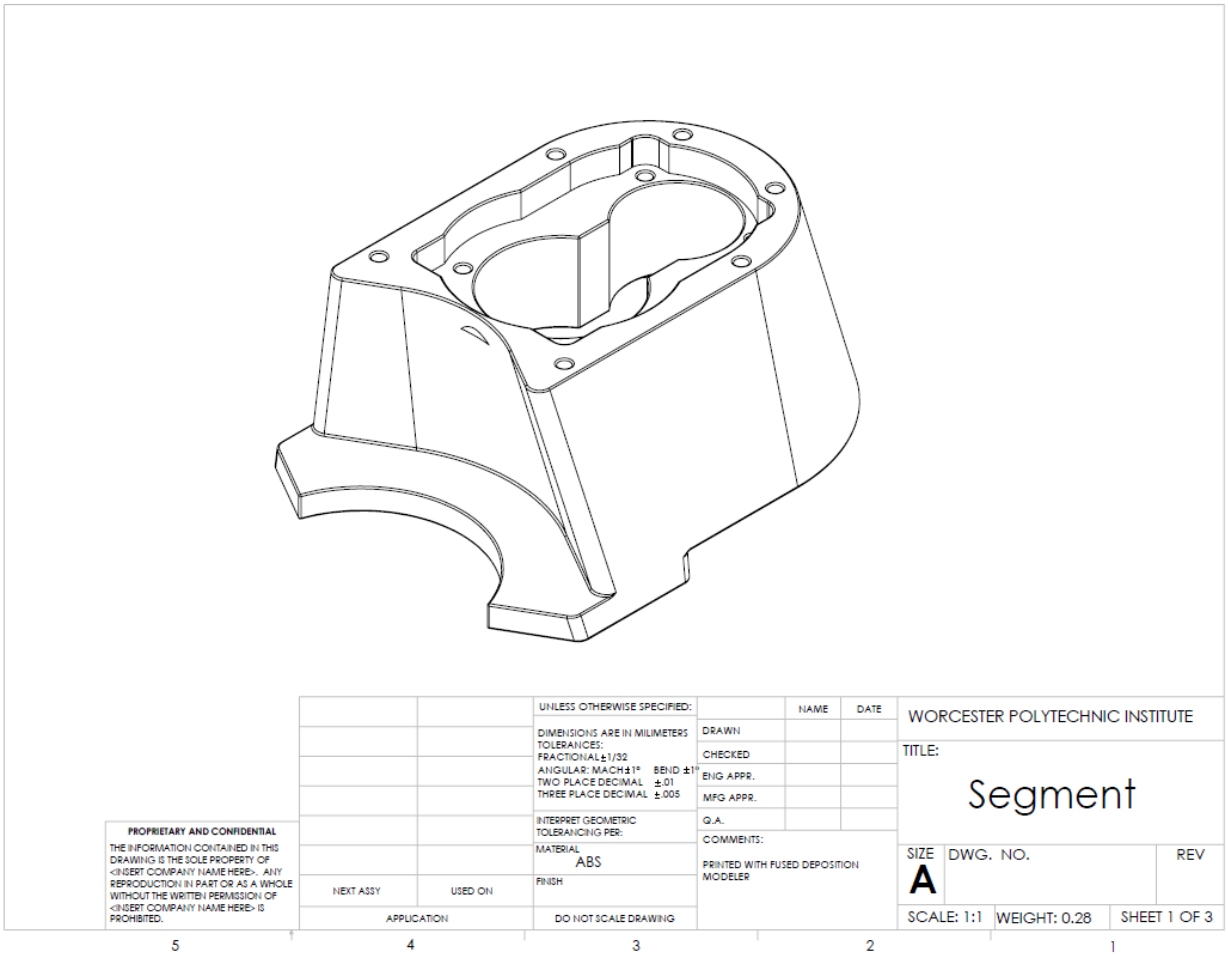
I. Albert, J. G. Sample, A. J. Morss, S. Rajagopalan, A. L. Barabasi, and P. Schiffer. Granular drag on a discrete object: Shape effects on jamming. Physical Review E, 64:061303, 2001.

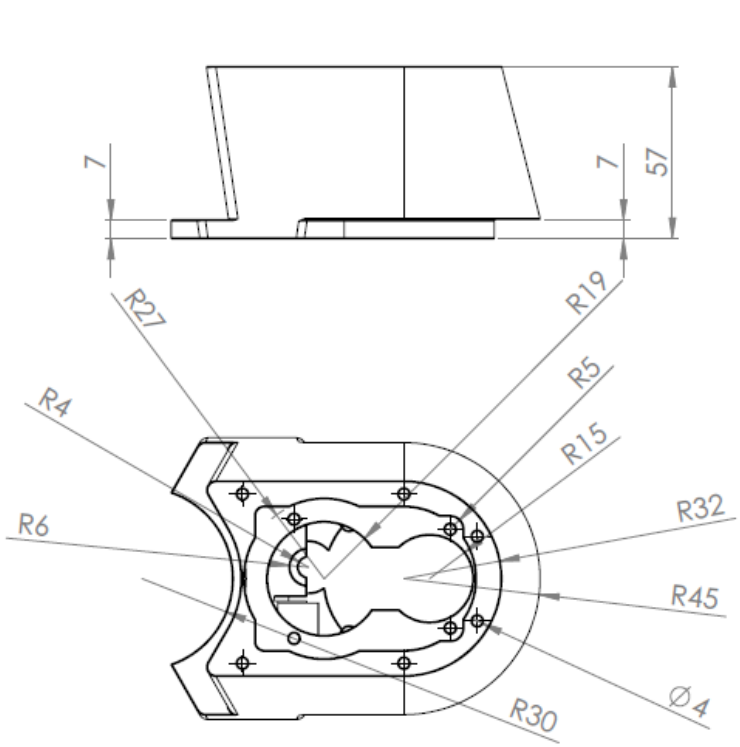
R. Albert, M. A. Pfeifer, A. L. Barabasi, and P. Schiffer. Slow drag in a granular medium. Physical Review Letters, 82:205{208, 1999.

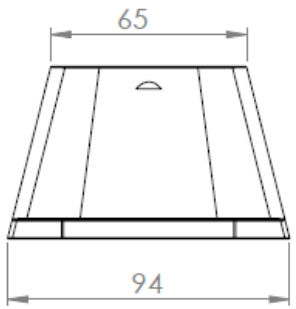
"Sandfish Robotics: the Story Continues." *Through The Sandglass*. 22 July 2010. Web. 25 Nov. 2010. <http://throughthesandglass.typepad.com/through_the_sandglass/2010/07/sandfish-robotics-the-story-continues.html>.

Appendices

Appendix A: CAD Drawings







		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN MILIMETERS TOLERANCES: FRACTIONAL ±1/32 ANGULAR: MACH ±1° BEND ±1° TWO PLACE DECIMAL ±.01 THREE PLACE DECIMAL ±.005	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL ABS	COMMENTS: PRINTED WITH FUSED DEPOSITION MODELER		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

WORCESTER POLYTECHNIC INSTITUTE

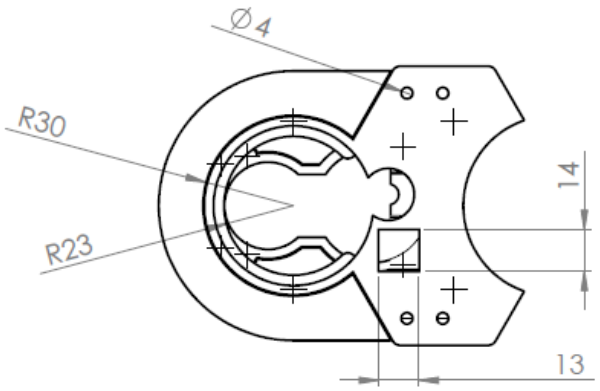
TITLE:

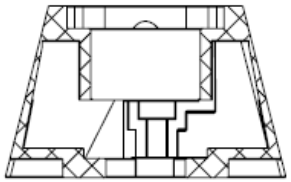
Segment

SIZE	DWG. NO.	REV
A		

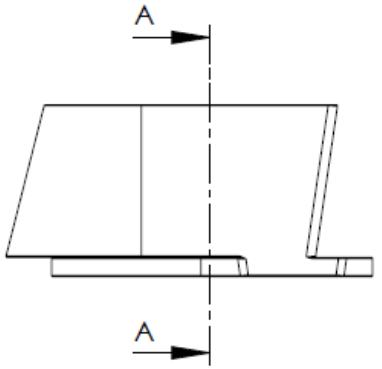
SCALE: 1:2	WEIGHT: 0.28	SHEET 2 OF 3

5 4 3 2 1





SECTION A-A



		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN MILIMETERS TOLERANCES: FRACTIONAL ±1/32 ANGULAR: MACH ±1° BEND ±1° TWO PLACE DECIMAL ±.01 THREE PLACE DECIMAL ±.005	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL ABS	COMMENTS: PRINTED WITH FUSED DEPOSITION MODELER		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

WORCESTER POLYTECHNIC INSTITUTE

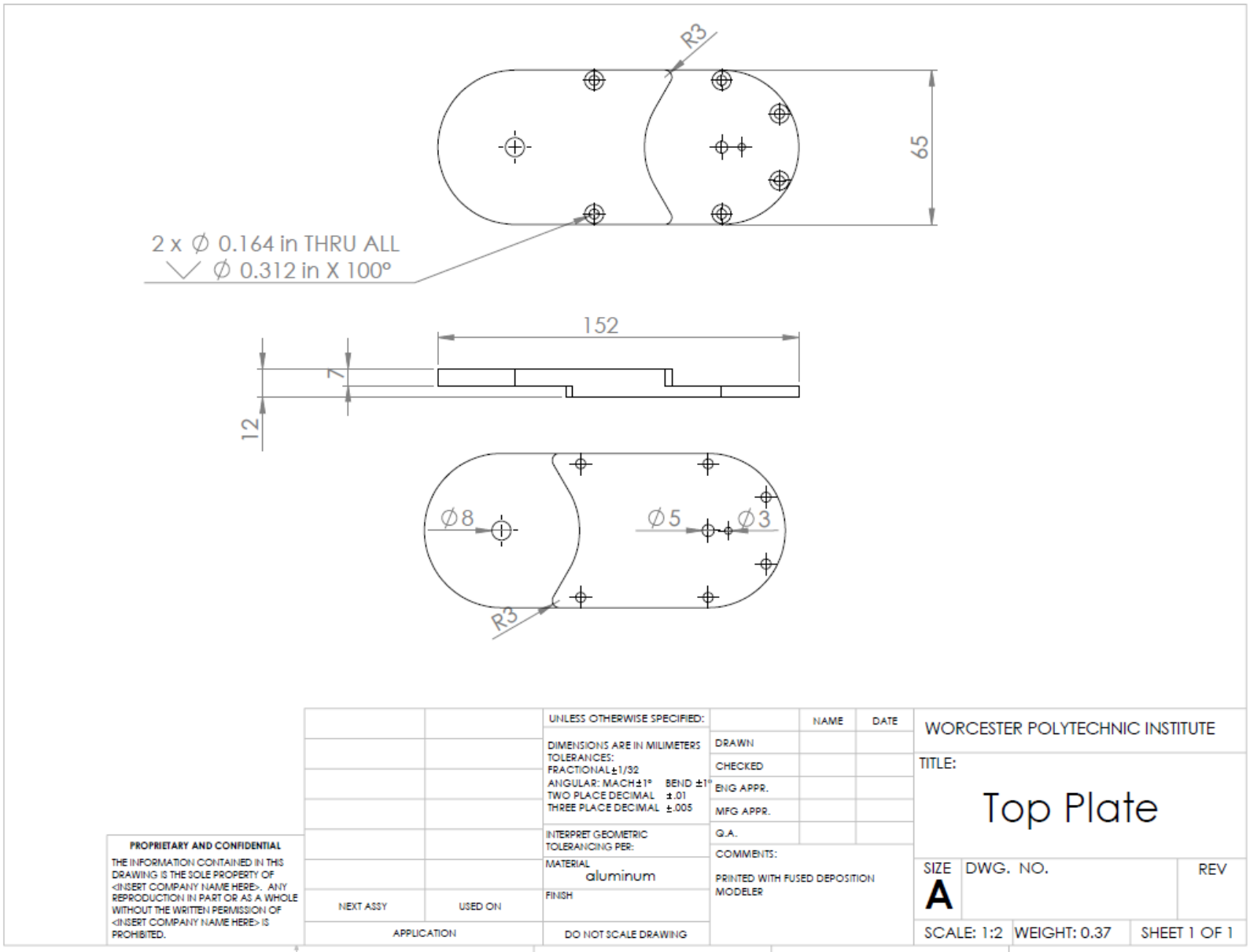
TITLE:

Segment

SIZE	DWG. NO.	REV
A		

SCALE: 1:2	WEIGHT: 0.28	SHEET 3 OF 3

5 4 3 2 1



		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN MILIMETERS TOLERANCES: FRACTIONAL ±1/32 ANGULAR: MACH ±1° BEND ±1° TWO PLACE DECIMAL ±.01 THREE PLACE DECIMAL ±.005	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL aluminum	COMMENTS: PRINTED WITH FUSED DEPOSITION MODELER		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

WORCESTER POLYTECHNIC INSTITUTE

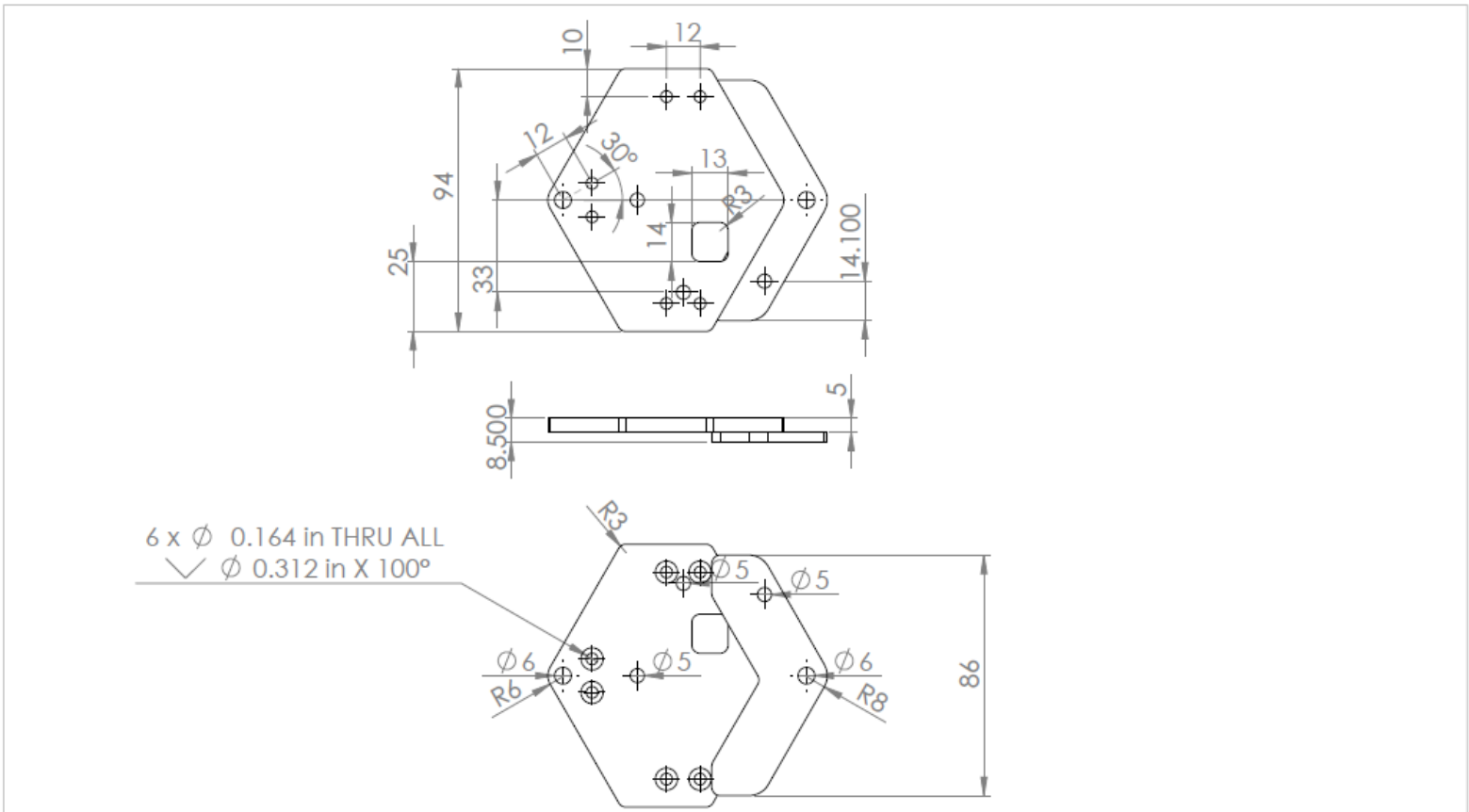
TITLE:

Top Plate

SIZE	DWG. NO.	REV
A		

SCALE: 1:2 | WEIGHT: 0.37 | SHEET 1 OF 1

5 4 3 2 1



		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN MILIMETERS TOLERANCES: FRACTIONAL ±1/32 ANGULAR: MACH ±1° BEND ±1° TWO PLACE DECIMAL ±.01 THREE PLACE DECIMAL ±.005	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
			Q.A.		
		INTERPRET GEOMETRIC TOLERANCING PER:	COMMENTS: PRINTED WITH FUSED DEPOSITION MODELER		
		MATERIAL aluminum			
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

PROPRIETARY AND CONFIDENTIAL

THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.

WORCESTER POLYTECHNIC INSTITUTE

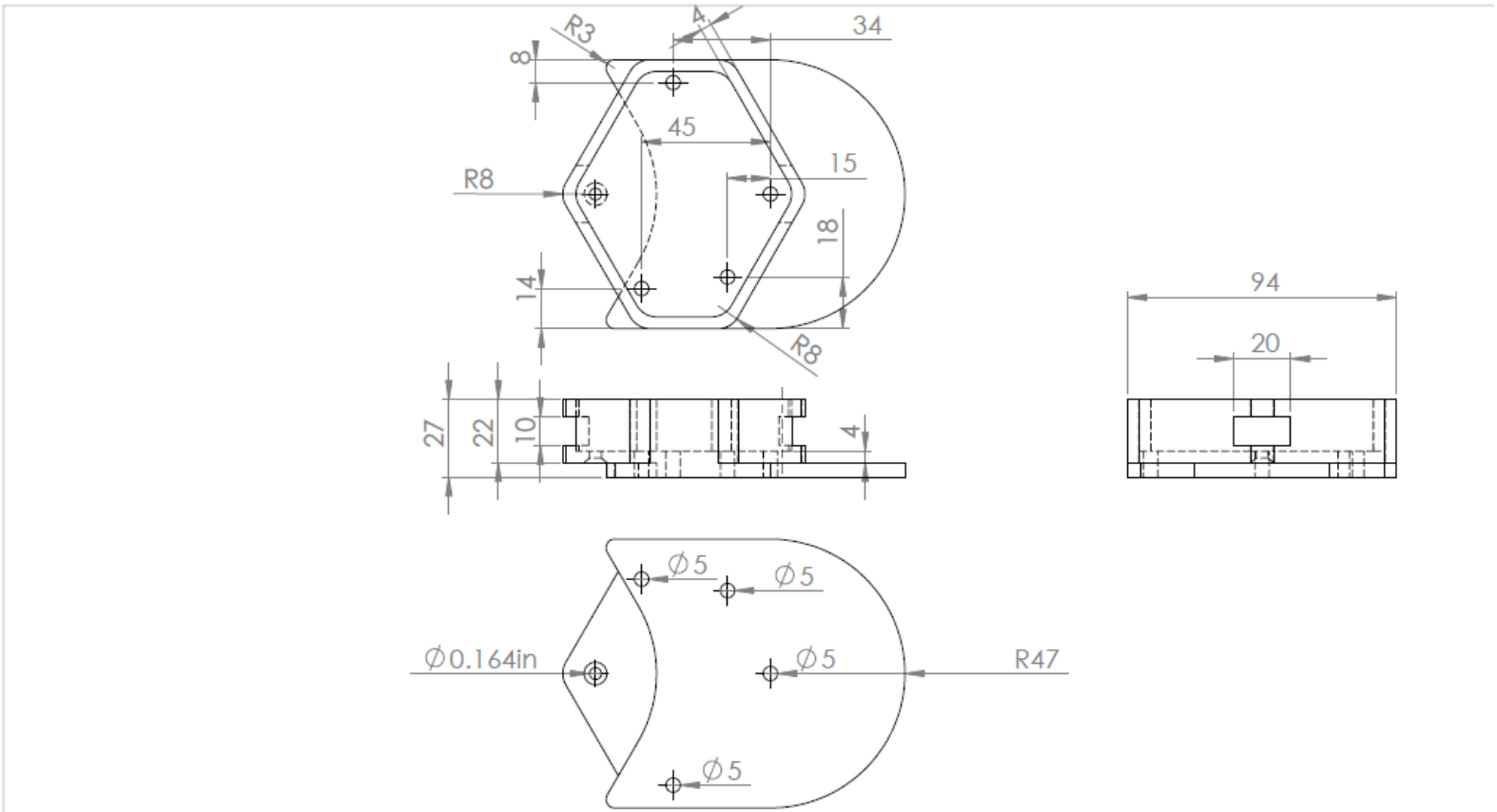
TITLE:

Middle Plate

SIZE	DWG. NO.	REV
A		

SCALE: 1:2	WEIGHT: 0.20	SHEET 1 OF 1

5 4 3 2 1



		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN MILIMETERS TOLERANCES: FRACTIONAL ±1/32 ANGULAR: MACH ±1° BEND ±1° TWO PLACE DECIMAL ±.01 THREE PLACE DECIMAL ±.005	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
			Q.A.		
		INTERPRET GEOMETRIC TOLERANCING PER:	COMMENTS: PRINTED WITH FUSED DEPOSITION MODELER		
		MATERIAL aluminum			
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

WORCESTER POLYTECHNIC INSTITUTE

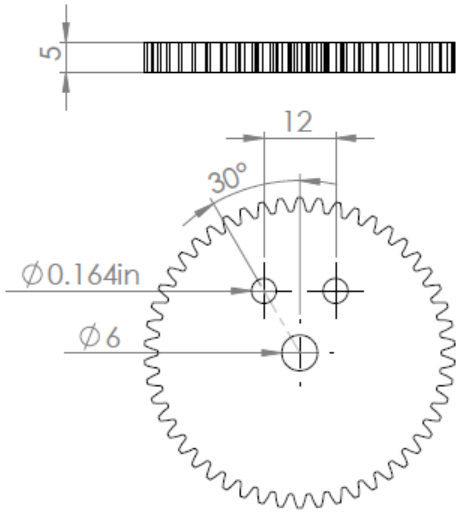
TITLE:

Bottom Plate

SIZE	DWG. NO.	REV
A		

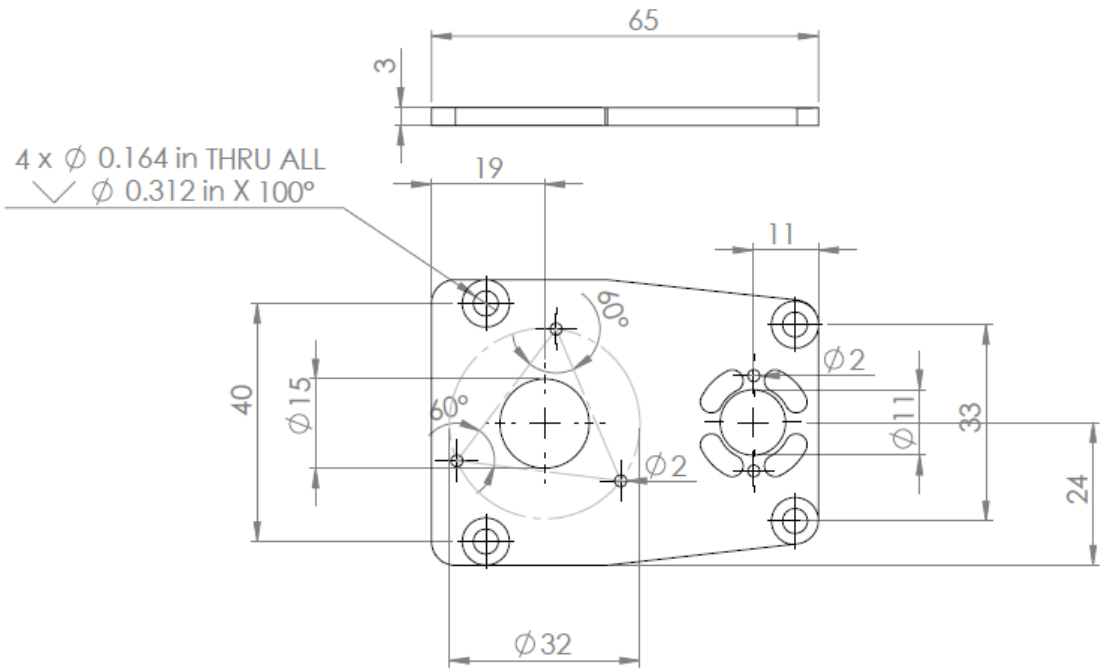
SCALE: 1:2	WEIGHT: 0.42	SHEET 1 OF 1

5 4 3 2 1



		UNLESS OTHERWISE SPECIFIED:		NAME	DATE	WORCESTER POLYTECHNIC INSTITUTE
		DIMENSIONS ARE IN MILIMETERS TOLERANCES: FRACTIONAL ±1/32 ANGULAR: MACH ±1° BEND ±1° TWO PLACE DECIMAL ±.01 THREE PLACE DECIMAL ±.005	DRAWN			TITLE: 50 Tooth Gear
			CHECKED			
			ENG APPR.			
			MFG APPR.			
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.			
		MATERIAL steel	COMMENTS: PRINTED WITH FUSED DEPOSITION MODELER			SIZE A / DWG. NO. / REV
NEXT ASSY	USED ON	FINISH				
APPLICATION		DO NOT SCALE DRAWING				SCALE: 1:1 / WEIGHT: 0.16 / SHEET 1 OF 1

5 4 3 2 1



		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN MILIMETERS TOLERANCES: FRACTIONAL ±1/32 ANGULAR: MACH ±1° BEND ±1° TWO PLACE DECIMAL ±.01 THREE PLACE DECIMAL ±.005	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL aluminum	COMMENTS: PRINTED WITH FUSED DEPOSITION MODELER		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

PROPRIETARY AND CONFIDENTIAL

THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.

WORCESTER POLYTECHNIC INSTITUTE

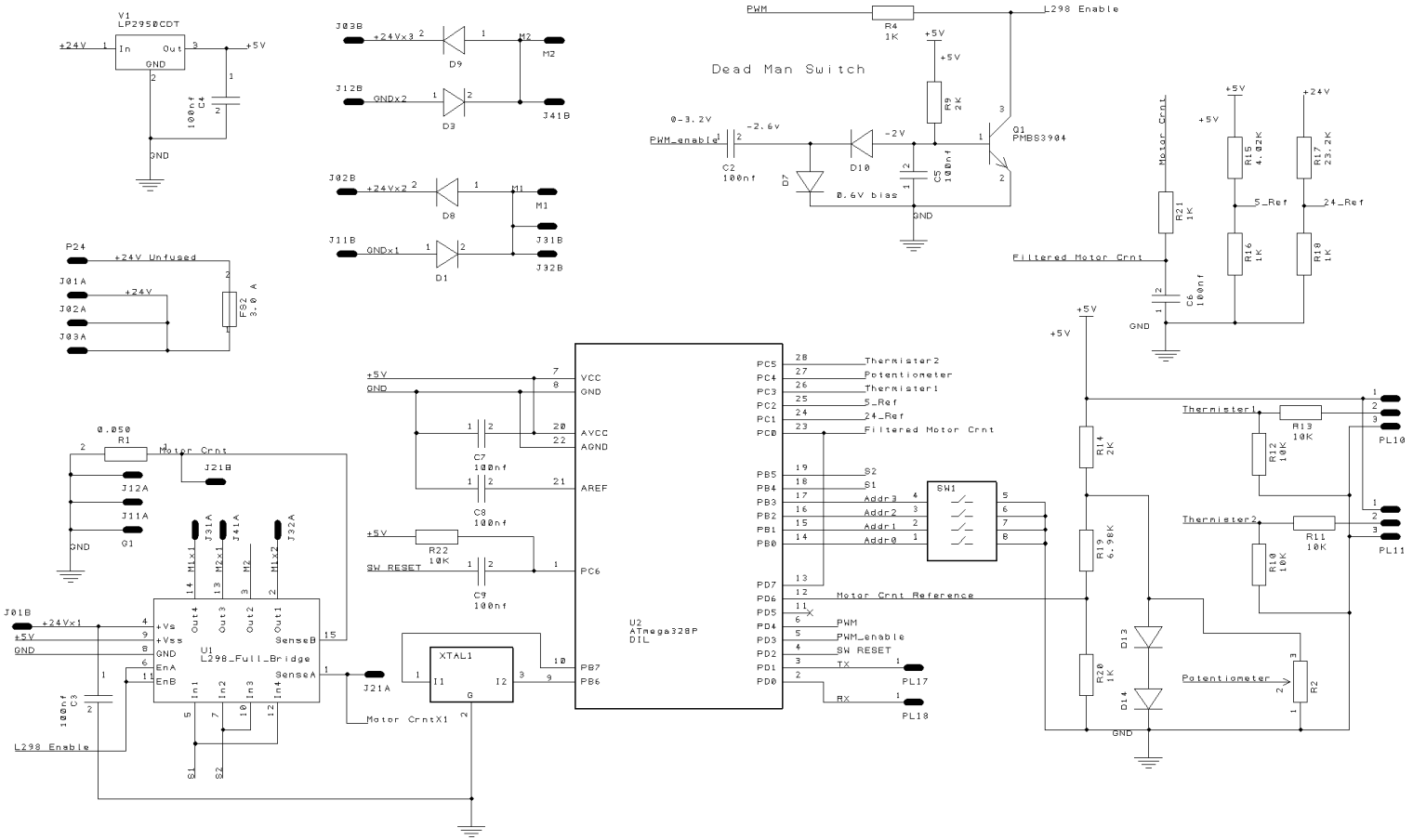
TITLE:

Motor Plate

SIZE	DWG. NO.	REV
A		
SCALE: 1:1	WEIGHT: 0.03	SHEET 1 OF 1

5 4 3 2 1

Appendix B: Electrical Diagram



Appendix C: Bill of Materials

Table 3: Electronics Bill of Materials

Reference Designator	Description	Component Value	Case / Package	Manufacturer	Manufacturer Part Number	Distributor	Distributor Part Number	Unit Price
V1	+5V Regulator		TO-252-3 (DPAK)	ON Semiconductor	LP2950CDT-3.0G	Mouser	863-LP2950CDT-3.0G	0.82
SOCKET	28 Pin DIP Socket	---	---	---	---	Anatools	Anarduino	
C2	Capacitor	100 nF	0603 (1608 metric)	TDK	C1608X7R1H104K	Mouser	810-C1608X7R1H104K	0.03
C3	Capacitor	100 nF	0603 (1608 metric)	TDK	C1608X7R1H104K	Mouser	810-C1608X7R1H104K	0.03
C4	Capacitor	100 nF	0603 (1608 metric)	TDK	C1608X7R1H104K	Mouser	810-C1608X7R1H104K	0.03
C5	Capacitor	100 nF	0603 (1608 metric)	TDK	C1608X7R1H104K	Mouser	810-C1608X7R1H104K	0.03
C6	Capacitor	100 nF	0603 (1608 metric)	TDK	C1608X7R1H104K	Mouser	810-C1608X7R1H104K	0.03

C6	Capacitor	100 nF	0603 (1608 metric)	TDK	C1608X7R1H104K	Mouser	810-C1608X7R1H104K	0.03
C7	Capacitor	100 nF	0603 (1608 metric)	TDK	C1608X7R1H104K	Mouser	810-C1608X7R1H104K	0.03
C8	Capacitor	100 nF	0603 (1608 metric)	TDK	C1608X7R1H104K	Mouser	810-C1608X7R1H104K	0.03
C9	Capacitor	100 nF	0603 (1608 metric)	TDK	C1608X7R1H104K	Mouser	810-C1608X7R1H104K	0.03
R1	Current Sense Resistor	50 mOhm	2010	TT Electronics	LR2010-LF-R050-F	Mouser	66-LR2010-LF-R050-F	0.54
SW1	DIP Switch / 4 Position	---	DIP-8-3	CTS Electronic Components	219-4LPST	Mouser	774-2194LPST	0.66
U1	Full Bridge Motor Driver	---	Multiwatt-15H	STMicroelectronics	L298HN	Mouser	511-L298HN	4.46
U2	Microprocessor / Arduino Bootloader	---	DIP-28-.3	Atmel	ATMega328P	Anatools	Anarduino	7

Motor	Motor	---	---	Johnson Electric	HC315MG-27.5-038-003	Testco	HC315MG-27.5-038-003	10.83
Q1	NPN Bipolar Transistor	---	SOT-23	NXP Semiconductors	PMBS3904,235	Mouser	771-PMBS3904235	0.03
R2	Potentiometer	10 Kohm	G-Style	Bourns	3382G-1-103G	Mouser	652-3382G-1-103G	1.51
PBC	Printed Circuit Board	---	---	Advanced Circuits	---	---	---	#DIV/0!
FS2	PTC Resettable Fuse	3 amp hold	Radial 5.1mm Space	Littlefuse	30R300UU	Mouser	576-30R300UU	0.33
R16	Resistor, Thick Film, 1%	1 Kohm	Series 292 0805 (2013 metric)	Xicon	292-1.0K-RC	Mouser	292-1.0K-RC	0.04
R18	Resistor, Thick Film, 1%	1 Kohm	Series 292 0805 (2013 metric)	Xicon	292-1.0K-RC	Mouser	292-1.0K-RC	0.04
R20	Resistor, Thick Film, 1%	1 Kohm	Series 292 0805 (2013 metric)	Xicon	292-1.0K-RC	Mouser	292-1.0K-RC	0.04

R21	Resistor, Thick Film, 1%	1 Kohm	Series 292 0805 (2013 metric)	Xicon	292-1.0K-RC	Mouser	292-1.0K-RC	0.04
R4	Resistor, Thick Film, 1%	1 Kohm	Series 292 0805 (2013 metric)	Xicon	292-1.0K-RC	Mouser	292-1.0K-RC	0.04
R10	Resistor, Thick Film, 1%	10 Kohm	Series 292 0805 (2013 metric)	Xicon	292-10K-RC	Mouser	292-10K-RC	0.04
R11	Resistor, Thick Film, 1%	10 Kohm	Series 292 0805 (2013 metric)	Xicon	292-10K-RC	Mouser	292-10K-RC	0.04
R12	Resistor, Thick Film, 1%	10 Kohm	Series 292 0805 (2013 metric)	Xicon	292-10K-RC	Mouser	292-10K-RC	0.04
R13	Resistor, Thick Film, 1%	10 Kohm	Series 292 0805 (2013 metric)	Xicon	292-10K-RC	Mouser	292-10K-RC	0.04
R22	Resistor, Thick Film, 1%	10 Kohm	Series 292 0805 (2013 metric)	Xicon	292-10K-RC	Mouser	292-10K-RC	0.04

R14	Resistor, Thick Film, 1%	2 Kohm	Series 292 0805 (2013 metric)	Xicon	292-2.0K-RC	Mouser	292-2.0K-RC	0.04
R9	Resistor, Thick Film, 1%	2 Kohm	Series 292 0805 (2013 metric)	Xicon	292-2.0K-RC	Mouser	292-2.0K-RC	0.04
R17	Resistor, Thick Film, 1%	23.2 Kohm	Series 292 0805 (2013 metric)	Xicon	292-23.2K-RC	Mouser	292-23.2K-RC	0.04
R15	Resistor, Thick Film, 1%	4.02 Kohm	Series 292 0805 (2013 metric)	Xicon	292-4.02K-RC	Mouser	292-4.02K-RC	0.04
R19	Resistor, Thick Film, 1%	6.98 Kohm	Series 292 0805 (2013 metric)	Xicon	292-6.98K-RC	Mouser	292-6.98K-RC	0.04
XTAL1	Resonator	16 MHz	MX	ECS	ZTT-16.00MX	Anatools	Anarduino	
D1	Schottky Diode, Fast / High Current	---	DO-220AA	Vishay	SS3P6HE3/84A	Mouser	625-SS3P6HE3	0.13

D3	Schottky Diode, Fast / High Current	---	DO-220AA	Vishay	SS3P6HE3/84A	Mouser	625-SS3P6HE3	0.13
D8	Schottky Diode, Fast / High Current	---	DO-220AA	Vishay	SS3P6HE3/84A	Mouser	625-SS3P6HE3	0.13
D9	Schottky Diode, Fast / High Current	---	DO-220AA	Vishay	SS3P6HE3/84A	Mouser	625-SS3P6HE3	0.13
D1	Switching Diode	1N4448	SOD-123	Vishay	1N4448W-V-GS08	Mouser	78-1N4448W-V	0.08
D13	Switching Diode	1N4448	SOD-123	Vishay	1N4448W-V-GS08	Mouser	78-1N4448W-V	0.08
D14	Switching Diode	1N4448	SOD-123	Vishay	1N4448W-V-GS08	Mouser	78-1N4448W-V	0.08
D7	Switching Diode	1N4448	SOD-123	Vishay	1N4448W-V-GS08	Mouser	78-1N4448W-V	0.08
T1	Temperature Sensor	---	TO-92 (SC-70-5 ??)	Microchip	MCP9700-E/TO	Mouser	579-MCP9700-E/TO	0.25

T2	Temperature Sensor	---	TO-92 (SC-70-5 ??)	Microchip	MCP9700-E/TO	Mouser	579-MCP9700-E/TO	0.25

Table 4: Electronics BOM Overview

Total Cost (per section)	33.10428571	(Total Cost less motors, Processor, PCB)
less selected parts		
Microprocessor / Arduino Bootloader	-7	
Motor	-10.83	
Full Bridge Motor Driver	-4.46	
Printed Circuit Board	-4.71	
Section cost (less selected parts)	6.1	
# Units	15	
Extended Cost	496.56	

Table 5: Mechanical Bill of Materials

Part	Part #	Quantity	Unit Cost	Total cost	Source
50 T Gear	https://sdp-si.com/eStore/PartDetail.asp?Opener=Order&PartID=57529&GroupID=593&Qty=0	1	83.82	83.82	https://sdp-si.com/eStore/
14 T Gear	https://sdp-si.com/eStore/PartDetail.asp?Opener=Order&PartID=1073&GroupID=593&Qty=0	1	24.34	24.34	https://sdp-si.com/eStore/

20 T Gear	https://sdp-si.com/eStore/PartDetail.asp?Opener=Order&PartID=51634&GroupID=593&Qty=0	1	32.75	46.48	https://sdp-si.com/eStore/
Segment middle		10	37.04	370.4	WPI RP
Tail		1	37.04	37.04	WPI RP
Head		1	37.04	37.04	WPI RP
TETRIX Gear Box		12	29.95	359.4	http://www.legoeducation.us/store/detail.aspx?ID=1610&bhcp=1
8-32 SS Flat Head 50pk	93085A197	4	9.12	36.48	
3/16" Dia 1' Shaft	88565K36	1	8	8	

Bronze Bearing	6391K122	26	0.41	10.66	
Nylon Bearing 5pk	6389K353	5	2.62	13.1	
Expandable Sleeving 10'	9284K393	1	14.95	14.95	
Spandex Sleeving		1	10	10	
Aluminum		2	145.35	145.35	
			Total Cost:	1197.06	

Table 6: Total Costs

Expenditure on Electronics	$496.56
Expenditure on Mechanical Components	$1197.06
Total Expenditures:	$1693.62

Appendix D: Robot Onboard Software

```
void setup() {
  // initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(3, OUTPUT);
  pinMode(2, INPUT);
  pinMode(14, INPUT);
  pinMode(15, INPUT);
  pinMode(16, INPUT);
  pinMode(17, INPUT);
  digitalWrite(3, LOW);    // set the LED off

  Serial.begin(115200);
  Serial.println("setup");
}

void loop() {

  int TX = 3;
  int RX = 2;
  int Addr0 = 14;
  int Addr1 = 15;
  int Addr2 = 16;
```

```
  int Addr3 = 17;
  int sectionNumber;
  int
switchCheck[]={digitalRead(Addr0),digitalRead(Addr1),digitalRead(Addr2),digitalRead(Addr3)
};


sectionNumber=((switchCheck[0]*1000)+(switchCheck[1]*100)+(switchCheck[2]*10)+switchC
heck);

  switch (sectionNumber) {
  case 1111:
    // statements
    break;
  case 1110:
    // statements
    break;                    //finish case statements for what section # this is.
  default:
    // statements
  }

}
```

```
void reportStats(int secID, int temp1, int temp2, int anglePos, int current){



  char message[10];
  format(secID,&message[0],&message[1]);
  format(temp1,&message[2],&message[3]);
  format(temp2,&message[4],&message[5]);
  format(anglePos,&message[6],&message[7]);
  format(current,&message[8],&message[9]);
  message[0]=0xFC;
  message[9]=message[9]-'a'+'!';
}


void format(byte input, char* upper, char* lower){
  if (input>239){
    input=239;
  }
  *upper = (input >> 4) & 0x0F + 'A';
  *lower = (input) & 0x0F + 'a';


}
```

Appendix E: Computer End Code

```
// MQP-PC-Main.cpp : Defines the entry point for the console application.
//
```

```
#include "stdafx.h"
#include <windows.h>
#include <stdio.h>
#include "ftd2xx.h"

int _tmain(int argc, _TCHAR* argv[])
{
	FT_HANDLE fthandle;
	FT_STATUS res;
	unsigned char buf[10];
	DWORD n, rq, tq, ev;
	long t, tmax = 300000L;

//	char	c, v;
//	int		num;
	int		position=10;
	int		msg[10], msgcnt, t1, t2, angle, current;


	res = FT_Open(0, &fthandle);
	if (res != FT_OK) {
		printf ("opening failed! with error %d\n", res);
		// printf ("handle1 is %x\n", fthandle);
		return 1;
	}

	res = FT_SetBaudRate(fthandle, 115200);
	if (res != FT_OK) {
		printf ("setbaudrate failed! with error %d\n", res);
		// printf ("handle1 is %x\n", fthandle);
		return 1;
	}

	res = FT_SetDataCharacteristics(fthandle, FT_BITS_8, FT_STOP_BITS_1, FT_PARITY_NONE);
	if (res != FT_OK) {
		printf ("setdatacharacteristics failed! with error %d\n", res);
		// printf ("handle1 is %x\n", fthandle);
		return 1;
	}


	t = tmax;
	do {
		if (t++ > tmax) {
			t = 0;
			position = -position;
			printf("\nposition=%3d ", position);

			buf[4] = 0xE0;
			buf[5] = position+75;

			buf[0] = (buf[4] >> 4) + '!';
			buf[1] = (buf[4] & 0x0F) + 'A';
			buf[2] = (buf[5] >> 4) + 'A';
			buf[3] = (buf[5] & 0x0F) + 'a';
			// printf (" sending (%c) (%c) (%c) (%c) ", buf[0], buf[1], buf[2], buf[3]);
			res = FT_Write(fthandle, buf, 4, &n);
			if (res != FT_OK) {
				printf ("write failed! with error %d\n", res);
```

```
                        return 1;
                }
        }

        rq = tq = ev = 0;
        res = FT_GetStatus (fthandle, &rq, &tq, &ev);
        if (res != FT_OK) {
                printf ("getstatus failed! with error %d\n", res);
                return 1;
        }
        if (rq == 0) continue;
        res = FT_Read(fthandle, &buf[0], 1, &n);
        if (res != FT_OK) {
                printf ("read failed! with error %d\n", res);
                return 1;
        }
        printf ("%c", buf[0]);

        if (buf[0] < 'A') msgcnt = 0;
        if (msgcnt < 10) msg[msgcnt++] = buf[0];
        if ((msgcnt == 10) && (msg[9] >= 'a')) {
                t1 = (((msg[2] - '!') & 0x000F) << 4) + ((msg[3] - '!') & 0x0F);
                t2 = (((msg[4] - '!') & 0x000F) << 4) + ((msg[5] - '!') & 0x0F);
                angle = (((msg[6] - '!') & 0x000F) << 4) + ((msg[7] - '!') & 0x0F) - 75;
                current = (((msg[8] - '!') & 0x000F) << 4) + ((msg[9] - '!') & 0x0F) - 75;
                printf (" t1: %3d  t2: %3d  angle: %3d  current: %3d ", t1, t2, angle, current);
        }


#if 0
        printf ("Position [L]eft [M]iddle [R]ight or [.] to exit? ");
        c = getchar ();
        printf ("[%02X]\n", c);
        if ((c == '.') || (c == 0x1B)) {
                printf ("exiting\n");
                continue;
        }

        c &= 0xDF;
        if (c == 'M') v=0;
        else if (c == 'L') v=-25;
        else v=+25;

        buf[4] = 0xE0;
        buf[5] = v+75;

        buf[0] = (buf[4] >> 4) + '!';
        buf[1] = (buf[4] & 0x0F) + 'a';
        buf[2] = (buf[5] >> 4) + '!';
        buf[3] = (buf[5] & 0x0F) + 'a';


        printf (" sending (%c) (%c) (%c) (%c) ", buf[0], buf[1], buf[2], buf[3]);
        res = FT_Write(fthandle, buf, 4, &n);
        if (res != FT_OK) {
                printf ("write failed! with error %d\n", res);
                return 1;
        }
        num = 0;
```

```
                for (num=0; num<10; num++) {

                        for (t=0; t<tmax; t++) {
                                rq = tq = ev = 0;
                                res = FT_GetStatus (fthandle, &rq, &tq, &ev);
                                if (res != FT_OK) {
                                        printf ("getstatus failed! with error %d\n", res);
                                        return 1;
                                }
                                if (rq == 0) continue;
                                res = FT_Read(fthandle, &buf[num], 1, &n);
                                if (res != FT_OK) {
                                        printf ("read failed! with error %d\n", res);
                                        return 1;
                                }
                                printf ("read [%02x] (%c) n=%d", buf[num], buf[num], n);
                                break;
                        }
                        printf (" ... t=%ld rq=%d tq=%d ev=%d\n", t, rq, tq, ev);
                }

                printf ("report: %02X %02X %02X %02X %02X %02X %02X %02X %02X %02X \n",
                        buf[0], buf[1], buf[2], buf[3], buf[4], buf[5], buf[6], buf[7], buf[8], buf[9]);

        } while ((c != '.') && (c != 0x1B));
#endif
        } while (1);

        FT_Close (fthandle);
        return 0;
}


/*++

Copyright © 2001-2010 Future Technology Devices International Limited

THIS SOFTWARE IS PROVIDED BY FUTURE TECHNOLOGY DEVICES INTERNATIONAL LIMITED "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES
OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL
FUTURE TECHNOLOGY DEVICES INTERNATIONAL LIMITED BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT
OF SUBSTITUTE GOODS OR SERVICES LOSS OF USE, DATA, OR PROFITS OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR
TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE,
EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

FTDI DRIVERS MAY BE USED ONLY IN CONJUNCTION WITH PRODUCTS BASED ON FTDI PARTS.

FTDI DRIVERS MAY BE DISTRIBUTED IN ANY FORM AS LONG AS LICENSE INFORMATION IS NOT
MODIFIED.

IF A CUSTOM VENDOR ID AND/OR PRODUCT ID OR DESCRIPTION STRING ARE USED, IT IS THE
```

RESPONSIBILITY OF THE PRODUCT MANUFACTURER TO MAINTAIN ANY CHANGES AND SUBSEQUENT WHQL
RE-CERTIFICATION AS A RESULT OF MAKING THESE CHANGES.

Module Name:

ftd2xx.h

Abstract:

Native USB device driver for FTDI FT232x, FT245x, FT2232x and FT4232x devices
FTD2XX library definitions

Environment:

kernel & user mode

Revision History:

13/03/01 awm	Created.
13/01/03 awm	Added device information support.
19/03/03 awm	Added FT_W32_CancelIo.
12/06/03 awm	Added FT_StopInTask and FT_RestartInTask.
18/09/03 awm	Added FT_SetResetPipeRetryCount.
10/10/03 awm	Added FT_ResetPort.
23/01/04 awm	Added support for open-by-location.
16/03/04 awm	Added support for FT2232C.
23/09/04 awm	Added support for FT232R.
20/10/04 awm	Added FT_CyclePort.
18/01/05 awm	Added FT_DEVICE_LIST_INFO_NODE type.
11/02/05 awm	Added LocId to FT_DEVICE_LIST_INFO_NODE.
25/08/05 awm	Added FT_SetDeadmanTimeout.
02/12/05 awm	Removed obsolete references.
05/12/05 awm	Added FT_GetVersion, FT_GetVersionEx.
08/09/06 awm	Added FT_W32_GetCommMask.
11/09/06 awm	Added FT_Rescan.
11/07/07 awm	Added support for FT2232H and FT4232H.
10/08/07 awm	Added flags definitions.
21/11/07 mja	Added FT_GetComPortNumber.
05/06/08 mja	Added EEPROM extensions for FT2232H.

--*/

```
#ifndef FTD2XX_H
#define FTD2XX_H

// The following ifdef block is the standard way of creating macros
// which make exporting from a DLL simpler.  All files within this DLL
// are compiled with the FTD2XX_EXPORTS symbol defined on the command line.
// This symbol should not be defined on any project that uses this DLL.
// This way any other project whose source files include this file see
// FTD2XX_API functions as being imported from a DLL, whereas this DLL
// sees symbols defined with this macro as being exported.

#ifdef FTD2XX_EXPORTS
#define FTD2XX_API __declspec(dllexport)
```

```
#else
#define FTD2XX_API __declspec(dllimport)
#endif


typedef PVOID    FT_HANDLE;
typedef ULONG    FT_STATUS;

//
// Device status
//
enum {
        FT_OK,
        FT_INVALID_HANDLE,
        FT_DEVICE_NOT_FOUND,
        FT_DEVICE_NOT_OPENED,
        FT_IO_ERROR,
        FT_INSUFFICIENT_RESOURCES,
        FT_INVALID_PARAMETER,
        FT_INVALID_BAUD_RATE,

        FT_DEVICE_NOT_OPENED_FOR_ERASE,
        FT_DEVICE_NOT_OPENED_FOR_WRITE,
        FT_FAILED_TO_WRITE_DEVICE,
        FT_EEPROM_READ_FAILED,
        FT_EEPROM_WRITE_FAILED,
        FT_EEPROM_ERASE_FAILED,
        FT_EEPROM_NOT_PRESENT,
        FT_EEPROM_NOT_PROGRAMMED,
        FT_INVALID_ARGS,
        FT_NOT_SUPPORTED,
        FT_OTHER_ERROR,
        FT_DEVICE_LIST_NOT_READY,
};


#define FT_SUCCESS(status) ((status) == FT_OK)

//
// FT_OpenEx Flags
//

#define FT_OPEN_BY_SERIAL_NUMBER   1
#define FT_OPEN_BY_DESCRIPTION             2
#define FT_OPEN_BY_LOCATION                4

//
// FT_ListDevices Flags (used in conjunction with FT_OpenEx Flags
//

#define FT_LIST_NUMBER_ONLY                0x80000000
#define FT_LIST_BY_INDEX                   0x40000000
#define FT_LIST_ALL                                0x20000000

#define FT_LIST_MASK (FT_LIST_NUMBER_ONLY|FT_LIST_BY_INDEX|FT_LIST_ALL)

//
// Baud Rates
//
```

```
#define FT_BAUD_300             300
#define FT_BAUD_600             600
#define FT_BAUD_1200            1200
#define FT_BAUD_2400            2400
#define FT_BAUD_4800            4800
#define FT_BAUD_9600            9600
#define FT_BAUD_14400           14400
#define FT_BAUD_19200           19200
#define FT_BAUD_38400           38400
#define FT_BAUD_57600           57600
#define FT_BAUD_115200          115200
#define FT_BAUD_230400          230400
#define FT_BAUD_460800          460800
#define FT_BAUD_921600          921600

//
// Word Lengths
//

#define FT_BITS_8               (UCHAR) 8
#define FT_BITS_7               (UCHAR) 7

//
// Stop Bits
//

#define FT_STOP_BITS_1          (UCHAR) 0
#define FT_STOP_BITS_2          (UCHAR) 1

//
// Parity
//

#define FT_PARITY_NONE          (UCHAR) 0
#define FT_PARITY_ODD           (UCHAR) 1
#define FT_PARITY_EVEN          (UCHAR) 2
#define FT_PARITY_MARK          (UCHAR) 3
#define FT_PARITY_SPACE         (UCHAR) 4

//
// Flow Control
//

#define FT_FLOW_NONE            0x0000
#define FT_FLOW_RTS_CTS         0x0100
#define FT_FLOW_DTR_DSR         0x0200
#define FT_FLOW_XON_XOFF        0x0400

//
// Purge rx and tx buffers
//
#define FT_PURGE_RX             1
#define FT_PURGE_TX             2

//
// Events
//
```

```
typedef void (*PFT_EVENT_HANDLER)(DWORD,DWORD);

#define FT_EVENT_RXCHAR                 1
#define FT_EVENT_MODEM_STATUS   2
#define FT_EVENT_LINE_STATUS    4

//
// Timeouts
//

#define FT_DEFAULT_RX_TIMEOUT   300
#define FT_DEFAULT_TX_TIMEOUT   300

//
// Device types
//

typedef ULONG   FT_DEVICE;

enum {
        FT_DEVICE_BM,
        FT_DEVICE_AM,
        FT_DEVICE_100AX,
        FT_DEVICE_UNKNOWN,
        FT_DEVICE_2232C,
        FT_DEVICE_232R,
        FT_DEVICE_2232H,
        FT_DEVICE_4232H
};


#ifdef __cplusplus
extern "C" {
#endif


        FTD2XX_API
                FT_STATUS WINAPI FT_Open(
                int deviceNumber,
                FT_HANDLE *pHandle
                );

        FTD2XX_API
                FT_STATUS WINAPI FT_OpenEx(
                PVOID pArg1,
                DWORD Flags,
                FT_HANDLE *pHandle
                );

        FTD2XX_API
                FT_STATUS WINAPI FT_ListDevices(
                PVOID pArg1,
                PVOID pArg2,
                DWORD Flags
                );

        FTD2XX_API
                FT_STATUS WINAPI FT_Close(
                FT_HANDLE ftHandle
```

```
        );

FTD2XX_API
        FT_STATUS WINAPI FT_Read(
        FT_HANDLE ftHandle,
        LPVOID lpBuffer,
        DWORD dwBytesToRead,
        LPDWORD lpBytesReturned
        );

FTD2XX_API
        FT_STATUS WINAPI FT_Write(
        FT_HANDLE ftHandle,
        LPVOID lpBuffer,
        DWORD dwBytesToWrite,
        LPDWORD lpBytesWritten
        );

FTD2XX_API
        FT_STATUS WINAPI FT_IoCtl(
        FT_HANDLE ftHandle,
        DWORD dwIoControlCode,
        LPVOID lpInBuf,
        DWORD nInBufSize,
        LPVOID lpOutBuf,
        DWORD nOutBufSize,
        LPDWORD lpBytesReturned,
        LPOVERLAPPED lpOverlapped
        );

FTD2XX_API
        FT_STATUS WINAPI FT_SetBaudRate(
        FT_HANDLE ftHandle,
        ULONG BaudRate
        );

FTD2XX_API
        FT_STATUS WINAPI FT_SetDivisor(
        FT_HANDLE ftHandle,
        USHORT Divisor
        );

FTD2XX_API
        FT_STATUS WINAPI FT_SetDataCharacteristics(
        FT_HANDLE ftHandle,
        UCHAR WordLength,
        UCHAR StopBits,
        UCHAR Parity
        );

FTD2XX_API
        FT_STATUS WINAPI FT_SetFlowControl(
        FT_HANDLE ftHandle,
        USHORT FlowControl,
        UCHAR XonChar,
        UCHAR XoffChar
        );

FTD2XX_API
```

```
        FT_STATUS WINAPI FT_ResetDevice(
        FT_HANDLE ftHandle
        );

FTD2XX_API
        FT_STATUS WINAPI FT_SetDtr(
        FT_HANDLE ftHandle
        );

FTD2XX_API
        FT_STATUS WINAPI FT_ClrDtr(
        FT_HANDLE ftHandle
        );

FTD2XX_API
        FT_STATUS WINAPI FT_SetRts(
        FT_HANDLE ftHandle
        );

FTD2XX_API
        FT_STATUS WINAPI FT_ClrRts(
        FT_HANDLE ftHandle
        );

FTD2XX_API
        FT_STATUS WINAPI FT_GetModemStatus(
        FT_HANDLE ftHandle,
        ULONG *pModemStatus
        );

FTD2XX_API
        FT_STATUS WINAPI FT_SetChars(
        FT_HANDLE ftHandle,
        UCHAR EventChar,
        UCHAR EventCharEnabled,
        UCHAR ErrorChar,
        UCHAR ErrorCharEnabled
        );

FTD2XX_API
        FT_STATUS WINAPI FT_Purge(
        FT_HANDLE ftHandle,
        ULONG Mask
        );

FTD2XX_API
        FT_STATUS WINAPI FT_SetTimeouts(
        FT_HANDLE ftHandle,
        ULONG ReadTimeout,
        ULONG WriteTimeout
        );

FTD2XX_API
        FT_STATUS WINAPI FT_GetQueueStatus(
        FT_HANDLE ftHandle,
        DWORD *dwRxBytes
        );

FTD2XX_API
```

```
        FT_STATUS WINAPI FT_SetEventNotification(
        FT_HANDLE ftHandle,
        DWORD Mask,
        PVOID Param
        );

FTD2XX_API
        FT_STATUS WINAPI FT_GetStatus(
        FT_HANDLE ftHandle,
        DWORD *dwRxBytes,
        DWORD *dwTxBytes,
        DWORD *dwEventDWord
        );

FTD2XX_API
        FT_STATUS WINAPI FT_SetBreakOn(
        FT_HANDLE ftHandle
        );

FTD2XX_API
        FT_STATUS WINAPI FT_SetBreakOff(
        FT_HANDLE ftHandle
        );

FTD2XX_API
        FT_STATUS WINAPI FT_SetWaitMask(
        FT_HANDLE ftHandle,
        DWORD Mask
        );

FTD2XX_API
        FT_STATUS WINAPI FT_WaitOnMask(
        FT_HANDLE ftHandle,
        DWORD *Mask
        );

FTD2XX_API
        FT_STATUS WINAPI FT_GetEventStatus(
        FT_HANDLE ftHandle,
        DWORD *dwEventDWord
        );

FTD2XX_API
        FT_STATUS WINAPI FT_ReadEE(
        FT_HANDLE ftHandle,
        DWORD dwWordOffset,
        LPWORD lpwValue
        );

FTD2XX_API
        FT_STATUS WINAPI FT_WriteEE(
        FT_HANDLE ftHandle,
        DWORD dwWordOffset,
        WORD wValue
        );

FTD2XX_API
        FT_STATUS WINAPI FT_EraseEE(
        FT_HANDLE ftHandle
```

```
);

//
// structure to hold program data for FT_Program function
//
typedef struct ft_program_data {

        DWORD Signature1;               // Header - must be 0x00000000
        DWORD Signature2;               // Header - must be 0xffffffff
        DWORD Version;                  // Header - FT_PROGRAM_DATA version
        //                      0 = original
        //                      1 = FT2232C extensions
        //                      2 = FT232R extensions
        //                      3 = FT2232H extensions
        //                      4 = FT4232H extensions

        WORD VendorId;                  // 0x0403
        WORD ProductId;                 // 0x6001
        char *Manufacturer;             // "FTDI"
        char *ManufacturerId;           // "FT"
        char *Description;              // "USB HS Serial Converter"
        char *SerialNumber;             // "FT000001" if fixed, or NULL
        WORD MaxPower;                  // 0 < MaxPower <= 500
        WORD PnP;                       // 0 = disabled, 1 = enabled
        WORD SelfPowered;               // 0 = bus powered, 1 = self powered
        WORD RemoteWakeup;              // 0 = not capable, 1 = capable
        //
        // Rev4 (FT232B) extensions
        //
        UCHAR Rev4;                     // non-zero if Rev4 chip, zero otherwise
        UCHAR IsoIn;                    // non-zero if in endpoint is isochronous
        UCHAR IsoOut;                   // non-zero if out endpoint is isochronous
        UCHAR PullDownEnable;           // non-zero if pull down enabled
        UCHAR SerNumEnable;             // non-zero if serial number to be used
        UCHAR USBVersionEnable;         // non-zero if chip uses USBVersion
        WORD USBVersion;                // BCD (0x0200 => USB2)
        //
        // Rev 5 (FT2232) extensions
        //
        UCHAR Rev5;                     // non-zero if Rev5 chip, zero otherwise
        UCHAR IsoInA;                   // non-zero if in endpoint is isochronous
        UCHAR IsoInB;                   // non-zero if in endpoint is isochronous
        UCHAR IsoOutA;                  // non-zero if out endpoint is isochronous
        UCHAR IsoOutB;                  // non-zero if out endpoint is isochronous
        UCHAR PullDownEnable5;          // non-zero if pull down enabled
        UCHAR SerNumEnable5;            // non-zero if serial number to be used
        UCHAR USBVersionEnable5;        // non-zero if chip uses USBVersion
        WORD USBVersion5;               // BCD (0x0200 => USB2)
        UCHAR AIsHighCurrent;           // non-zero if interface is high current
        UCHAR BIsHighCurrent;           // non-zero if interface is high current
        UCHAR IFAIsFifo;                // non-zero if interface is 245 FIFO
        UCHAR IFAIsFifoTar;             // non-zero if interface is 245 FIFO CPU target
        UCHAR IFAIsFastSer;             // non-zero if interface is Fast serial
        UCHAR AIsVCP;                   // non-zero if interface is to use VCP drivers
        UCHAR IFBIsFifo;                // non-zero if interface is 245 FIFO
        UCHAR IFBIsFifoTar;             // non-zero if interface is 245 FIFO CPU target
        UCHAR IFBIsFastSer;             // non-zero if interface is Fast serial
        UCHAR BIsVCP;                   // non-zero if interface is to use VCP drivers
        //
```

```
        // Rev 6 (FT232R) extensions
        //
        UCHAR UseExtOsc;                    // Use External Oscillator
        UCHAR HighDriveIOs;                 // High Drive I/Os
        UCHAR EndpointSize;                 // Endpoint size
        UCHAR PullDownEnableR;              // non-zero if pull down enabled
        UCHAR SerNumEnableR;            // non-zero if serial number to be used
        UCHAR InvertTXD;                    // non-zero if invert TXD
        UCHAR InvertRXD;                    // non-zero if invert RXD
        UCHAR InvertRTS;                    // non-zero if invert RTS
        UCHAR InvertCTS;                    // non-zero if invert CTS
        UCHAR InvertDTR;                    // non-zero if invert DTR
        UCHAR InvertDSR;                    // non-zero if invert DSR
        UCHAR InvertDCD;                    // non-zero if invert DCD
        UCHAR InvertRI;                     // non-zero if invert RI
        UCHAR Cbus0;                        // Cbus Mux control
        UCHAR Cbus1;                        // Cbus Mux control
        UCHAR Cbus2;                        // Cbus Mux control
        UCHAR Cbus3;                        // Cbus Mux control
        UCHAR Cbus4;                        // Cbus Mux control
        UCHAR RIsD2XX;                          // non-zero if using D2XX driver
        //
        // Rev 7 (FT2232H) Extensions
        //
        UCHAR PullDownEnable7;          // non-zero if pull down enabled
        UCHAR SerNumEnable7;            // non-zero if serial number to be used
        UCHAR ALSlowSlew;                   // non-zero if AL pins have slow slew
        UCHAR ALSchmittInput;           // non-zero if AL pins are Schmitt input
        UCHAR ALDriveCurrent;           // valid values are 4mA, 8mA, 12mA, 16mA
        UCHAR AHSlowSlew;                   // non-zero if AH pins have slow slew
        UCHAR AHSchmittInput;           // non-zero if AH pins are Schmitt input
        UCHAR AHDriveCurrent;           // valid values are 4mA, 8mA, 12mA, 16mA
        UCHAR BLSlowSlew;                   // non-zero if BL pins have slow slew
        UCHAR BLSchmittInput;           // non-zero if BL pins are Schmitt input
        UCHAR BLDriveCurrent;           // valid values are 4mA, 8mA, 12mA, 16mA
        UCHAR BHSlowSlew;                   // non-zero if BH pins have slow slew
        UCHAR BHSchmittInput;           // non-zero if BH pins are Schmitt input
        UCHAR BHDriveCurrent;           // valid values are 4mA, 8mA, 12mA, 16mA
        UCHAR IFAIsFifo7;                   // non-zero if interface is 245 FIFO
        UCHAR IFAIsFifoTar7;            // non-zero if interface is 245 FIFO CPU target
        UCHAR IFAIsFastSer7;            // non-zero if interface is Fast serial
        UCHAR AIsVCP7;                          // non-zero if interface is to use VCP drivers
        UCHAR IFBIsFifo7;                   // non-zero if interface is 245 FIFO
        UCHAR IFBIsFifoTar7;            // non-zero if interface is 245 FIFO CPU target
        UCHAR IFBIsFastSer7;            // non-zero if interface is Fast serial
        UCHAR BIsVCP7;                          // non-zero if interface is to use VCP drivers
        UCHAR PowerSaveEnable;          // non-zero if using BCBUS7 to save power for self-powered
designs
        //
        // Rev 8 (FT4232H) Extensions
        //
        UCHAR PullDownEnable8;          // non-zero if pull down enabled
        UCHAR SerNumEnable8;            // non-zero if serial number to be used
        UCHAR ASlowSlew;                    // non-zero if AL pins have slow slew
        UCHAR ASchmittInput;            // non-zero if AL pins are Schmitt input
        UCHAR ADriveCurrent;            // valid values are 4mA, 8mA, 12mA, 16mA
        UCHAR BSlowSlew;                    // non-zero if AH pins have slow slew
        UCHAR BSchmittInput;            // non-zero if AH pins are Schmitt input
        UCHAR BDriveCurrent;            // valid values are 4mA, 8mA, 12mA, 16mA
```

```
        UCHAR CSlowSlew;                // non-zero if BL pins have slow slew
        UCHAR CSchmittInput;            // non-zero if BL pins are Schmitt input
        UCHAR CDriveCurrent;            // valid values are 4mA, 8mA, 12mA, 16mA
        UCHAR DSlowSlew;                // non-zero if BH pins have slow slew
        UCHAR DSchmittInput;            // non-zero if BH pins are Schmitt input
        UCHAR DDriveCurrent;            // valid values are 4mA, 8mA, 12mA, 16mA
        UCHAR ARIIsTXDEN;               // non-zero if port A uses RI as RS485 TXDEN
        UCHAR BRIIsTXDEN;               // non-zero if port B uses RI as RS485 TXDEN
        UCHAR CRIIsTXDEN;               // non-zero if port C uses RI as RS485 TXDEN
        UCHAR DRIIsTXDEN;               // non-zero if port D uses RI as RS485 TXDEN
        UCHAR AIsVCP8;                  // non-zero if interface is to use VCP drivers
        UCHAR BIsVCP8;                  // non-zero if interface is to use VCP drivers
        UCHAR CIsVCP8;                  // non-zero if interface is to use VCP drivers
        UCHAR DIsVCP8;                  // non-zero if interface is to use VCP drivers

} FT_PROGRAM_DATA, *PFT_PROGRAM_DATA;

FTD2XX_API
        FT_STATUS WINAPI FT_EE_Program(
        FT_HANDLE ftHandle,
        PFT_PROGRAM_DATA pData
        );

FTD2XX_API
        FT_STATUS WINAPI FT_EE_ProgramEx(
        FT_HANDLE ftHandle,
        PFT_PROGRAM_DATA pData,
        char *Manufacturer,
        char *ManufacturerId,
        char *Description,
        char *SerialNumber
        );

FTD2XX_API
        FT_STATUS WINAPI FT_EE_Read(
        FT_HANDLE ftHandle,
        PFT_PROGRAM_DATA pData
        );

FTD2XX_API
        FT_STATUS WINAPI FT_EE_ReadEx(
        FT_HANDLE ftHandle,
        PFT_PROGRAM_DATA pData,
        char *Manufacturer,
        char *ManufacturerId,
        char *Description,
        char *SerialNumber
        );

FTD2XX_API
        FT_STATUS WINAPI FT_EE_UASize(
        FT_HANDLE ftHandle,
        LPDWORD lpdwSize
        );

FTD2XX_API
        FT_STATUS WINAPI FT_EE_UAWrite(
        FT_HANDLE ftHandle,
        PUCHAR pucData,
```

```
        DWORD dwDataLen
        );

FTD2XX_API
        FT_STATUS WINAPI FT_EE_UARead(
        FT_HANDLE ftHandle,
        PUCHAR pucData,
        DWORD dwDataLen,
        LPDWORD lpdwBytesRead
        );

FTD2XX_API
        FT_STATUS WINAPI FT_SetLatencyTimer(
        FT_HANDLE ftHandle,
        UCHAR ucLatency
        );

FTD2XX_API
        FT_STATUS WINAPI FT_GetLatencyTimer(
        FT_HANDLE ftHandle,
        PUCHAR pucLatency
        );

FTD2XX_API
        FT_STATUS WINAPI FT_SetBitMode(
        FT_HANDLE ftHandle,
        UCHAR ucMask,
        UCHAR ucEnable
        );

FTD2XX_API
        FT_STATUS WINAPI FT_GetBitMode(
        FT_HANDLE ftHandle,
        PUCHAR pucMode
        );

FTD2XX_API
        FT_STATUS WINAPI FT_SetUSBParameters(
        FT_HANDLE ftHandle,
        ULONG ulInTransferSize,
        ULONG ulOutTransferSize
        );

FTD2XX_API
        FT_STATUS WINAPI FT_SetDeadmanTimeout(
        FT_HANDLE ftHandle,
        ULONG ulDeadmanTimeout
        );

FTD2XX_API
        FT_STATUS WINAPI FT_GetDeviceInfo(
        FT_HANDLE ftHandle,
        FT_DEVICE *lpftDevice,
        LPDWORD lpdwID,
        PCHAR SerialNumber,
        PCHAR Description,
        LPVOID Dummy
        );
```

```
FTD2XX_API
	FT_STATUS WINAPI FT_StopInTask(
	FT_HANDLE ftHandle
	);

FTD2XX_API
	FT_STATUS WINAPI FT_RestartInTask(
	FT_HANDLE ftHandle
	);

FTD2XX_API
	FT_STATUS WINAPI FT_SetResetPipeRetryCount(
	FT_HANDLE ftHandle,
	DWORD dwCount
	);

FTD2XX_API
	FT_STATUS WINAPI FT_ResetPort(
	FT_HANDLE ftHandle
	);

FTD2XX_API
	FT_STATUS WINAPI FT_CyclePort(
	FT_HANDLE ftHandle
	);


//
// Win32-type functions
//

FTD2XX_API
	FT_HANDLE WINAPI FT_W32_CreateFile(
	LPCTSTR                                 lpszName,
	DWORD                                   dwAccess,
	DWORD                                   dwShareMode,
	LPSECURITY_ATTRIBUTES   lpSecurityAttributes,
	DWORD                                   dwCreate,
	DWORD                                   dwAttrsAndFlags,
	HANDLE                                  hTemplate
	);

FTD2XX_API
	BOOL WINAPI FT_W32_CloseHandle(
	FT_HANDLE ftHandle
	);

FTD2XX_API
	BOOL WINAPI FT_W32_ReadFile(
	FT_HANDLE ftHandle,
	LPVOID lpBuffer,
	DWORD nBufferSize,
	LPDWORD lpBytesReturned,
	LPOVERLAPPED lpOverlapped
	);

FTD2XX_API
	BOOL WINAPI FT_W32_WriteFile(
	FT_HANDLE ftHandle,
```

```
        LPVOID lpBuffer,
        DWORD nBufferSize,
        LPDWORD lpBytesWritten,
        LPOVERLAPPED lpOverlapped
        );

FTD2XX_API
        DWORD WINAPI FT_W32_GetLastError(
        FT_HANDLE ftHandle
        );

FTD2XX_API
        BOOL WINAPI FT_W32_GetOverlappedResult(
        FT_HANDLE ftHandle,
        LPOVERLAPPED lpOverlapped,
        LPDWORD lpdwBytesTransferred,
        BOOL bWait
        );

FTD2XX_API
        BOOL WINAPI FT_W32_CancelIo(
        FT_HANDLE ftHandle
        );


//
// Win32 COMM API type functions
//
typedef struct _FTCOMSTAT {
        DWORD fCtsHold : 1;
        DWORD fDsrHold : 1;
        DWORD fRlsdHold : 1;
        DWORD fXoffHold : 1;
        DWORD fXoffSent : 1;
        DWORD fEof : 1;
        DWORD fTxim : 1;
        DWORD fReserved : 25;
        DWORD cbInQue;
        DWORD cbOutQue;
} FTCOMSTAT, *LPFTCOMSTAT;

typedef struct _FTDCB {
        DWORD DCBlength;                        /* sizeof(FTDCB)
*/
        DWORD BaudRate;                                 /* Baudrate at which running
*/
        DWORD fBinary: 1;                       /* Binary Mode (skip EOF check)
*/
        DWORD fParity: 1;                       /* Enable parity checking
*/
        DWORD fOutxCtsFlow:1;           /* CTS handshaking on output                    */
        DWORD fOutxDsrFlow:1;           /* DSR handshaking on output                    */
        DWORD fDtrControl:2;            /* DTR Flow control
*/
        DWORD fDsrSensitivity:1; /* DSR Sensitivity                                     */
        DWORD fTXContinueOnXoff: 1;     /* Continue TX when Xoff sent                   */
        DWORD fOutX: 1;                                 /* Enable output X-ON/X-OFF
        */
```

```
        DWORD fInX: 1;                  /* Enable input X-ON/X-OFF
*/
        DWORD fErrorChar: 1;        /* Enable Err Replacement                  */
        DWORD fNull: 1;                 /* Enable Null stripping
*/
        DWORD fRtsControl:2;        /* Rts Flow control
*/
        DWORD fAbortOnError:1;      /* Abort all reads and writes on Error */
        DWORD fDummy2:17;               /* Reserved
                */
        WORD wReserved;                         /* Not currently used
        */
        WORD XonLim;                    /* Transmit X-ON threshold
*/
        WORD XoffLim;                   /* Transmit X-OFF threshold
*/
        BYTE ByteSize;                  /* Number of bits/byte, 4-8
*/
        BYTE Parity;                    /* 0-4=None,Odd,Even,Mark,Space
*/
        BYTE StopBits;                  /* 0,1,2 = 1, 1.5, 2
*/
        char XonChar;                   /* Tx and Rx X-ON character
*/
        char XoffChar;                  /* Tx and Rx X-OFF character
*/
        char ErrorChar;                 /* Error replacement char
*/
        char EofChar;                   /* End of Input character
*/
        char EvtChar;                   /* Received Event character
*/
        WORD wReserved1;                /* Fill for now.
*/
} FTDCB, *LPFTDCB;

typedef struct _FTTIMEOUTS {
        DWORD ReadIntervalTimeout;              /* Maximum time between read chars.
*/
        DWORD ReadTotalTimeoutMultiplier;   /* Multiplier of characters.       */
        DWORD ReadTotalTimeoutConstant;         /* Constant in milliseconds.       */
        DWORD WriteTotalTimeoutMultiplier;  /* Multiplier of characters.       */
        DWORD WriteTotalTimeoutConstant;    /* Constant in milliseconds.       */
} FTTIMEOUTS,*LPFTTIMEOUTS;


FTD2XX_API
        BOOL WINAPI FT_W32_ClearCommBreak(
        FT_HANDLE ftHandle
        );

FTD2XX_API
        BOOL WINAPI FT_W32_ClearCommError(
        FT_HANDLE ftHandle,
        LPDWORD lpdwErrors,
        LPFTCOMSTAT lpftComstat
        );

FTD2XX_API
```

```
        BOOL WINAPI FT_W32_EscapeCommFunction(
        FT_HANDLE ftHandle,
        DWORD dwFunc
        );

FTD2XX_API
        BOOL WINAPI FT_W32_GetCommModemStatus(
        FT_HANDLE ftHandle,
        LPDWORD lpdwModemStatus
        );

FTD2XX_API
        BOOL WINAPI FT_W32_GetCommState(
        FT_HANDLE ftHandle,
        LPFTDCB lpftDcb
        );

FTD2XX_API
        BOOL WINAPI FT_W32_GetCommTimeouts(
        FT_HANDLE ftHandle,
        FTTIMEOUTS *pTimeouts
        );

FTD2XX_API
        BOOL WINAPI FT_W32_PurgeComm(
        FT_HANDLE ftHandle,
        DWORD dwMask
        );

FTD2XX_API
        BOOL WINAPI FT_W32_SetCommBreak(
        FT_HANDLE ftHandle
        );

FTD2XX_API
        BOOL WINAPI FT_W32_SetCommMask(
        FT_HANDLE ftHandle,
        ULONG ulEventMask
        );

FTD2XX_API
        BOOL WINAPI FT_W32_GetCommMask(
        FT_HANDLE ftHandle,
        LPDWORD lpdwEventMask
        );

FTD2XX_API
        BOOL WINAPI FT_W32_SetCommState(
        FT_HANDLE ftHandle,
        LPFTDCB lpftDcb
        );

FTD2XX_API
        BOOL WINAPI FT_W32_SetCommTimeouts(
        FT_HANDLE ftHandle,
        FTTIMEOUTS *pTimeouts
        );

FTD2XX_API
```

```
        BOOL WINAPI FT_W32_SetupComm(
        FT_HANDLE ftHandle,
        DWORD dwReadBufferSize,
        DWORD dwWriteBufferSize
        );

FTD2XX_API
        BOOL WINAPI FT_W32_WaitCommEvent(
        FT_HANDLE ftHandle,
        PULONG pulEvent,
        LPOVERLAPPED lpOverlapped
        );


//
// Device information
//

typedef struct _ft_device_list_info_node {
        ULONG Flags;
        ULONG Type;
        ULONG ID;
        DWORD LocId;
        char SerialNumber[16];
        char Description[64];
        FT_HANDLE ftHandle;
} FT_DEVICE_LIST_INFO_NODE;

// Device information flags
enum {
        FT_FLAGS_OPENED = 1,
        FT_FLAGS_HISPEED = 2
};


FTD2XX_API
        FT_STATUS WINAPI FT_CreateDeviceInfoList(
        LPDWORD lpdwNumDevs
        );

FTD2XX_API
        FT_STATUS WINAPI FT_GetDeviceInfoList(
        FT_DEVICE_LIST_INFO_NODE *pDest,
        LPDWORD lpdwNumDevs
        );

FTD2XX_API
        FT_STATUS WINAPI FT_GetDeviceInfoDetail(
        DWORD dwIndex,
        LPDWORD lpdwFlags,
        LPDWORD lpdwType,
        LPDWORD lpdwID,
        LPDWORD lpdwLocId,
        LPVOID lpSerialNumber,
        LPVOID lpDescription,
        FT_HANDLE *pftHandle
        );
```

```
	//
	// Version information
	//

	FTD2XX_API
		FT_STATUS WINAPI FT_GetDriverVersion(
		FT_HANDLE ftHandle,
		LPDWORD lpdwVersion
		);

	FTD2XX_API
		FT_STATUS WINAPI FT_GetLibraryVersion(
		LPDWORD lpdwVersion
		);


	FTD2XX_API
		FT_STATUS WINAPI FT_Rescan(
		void
		);

	FTD2XX_API
		FT_STATUS WINAPI FT_Reload(
		WORD wVid,
		WORD wPid
		);

	FTD2XX_API
		FT_STATUS WINAPI FT_GetComPortNumber(
		FT_HANDLE ftHandle,
		LPLONG		lpdwComPortNumber
		);


#ifdef __cplusplus
}
#endif


#endif	/* FTD2XX_H */


// stdafx.cpp : source file that includes just the standard includes
// ftdi1.pch will be the pre-compiled header
// stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"

// TODO: reference any additional headers you need in STDAFX.H
// and not in this file

// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#pragma once

#include "targetver.h"
```

```
#include <stdio.h>
#include <tchar.h>


// TODO: reference additional headers your program requires here

#pragma once

// Including SDKDDKVer.h defines the highest available Windows platform.

// If you wish to build your application for a previous Windows platform, include WinSDKVer.h and
// set the _WIN32_WINNT macro to the platform you wish to support before including SDKDDKVer.h.

#include <SDKDDKVer.h>
```

Appendix F: MATLAB Code

Compare surfaces.m MATLAB CODE: This code was also utilized to compare the sandpaper results with the non-sandpaper results, producing the graphs in Figure 20.

```
function compare_surfaces

if ~true
    sides_smooth = [nan];
    sides_rough = [];
    bottom_smooth = [];
    bottom_rough = [];
    front_smooth = [];
    front_rough = [];

    save( strcat( mfilename, '.mat' ), 'sides_smooth', 'sides_rough',
'bottom_smooth', 'bottom_rough', 'front_smooth', 'front_rough' );
else
    load(strcat( mfilename, '.mat' ));
end
%%
cols = {'Lx', 'Ly', 'Lz', 'Zf', 'Zsf', 'Zsb', 'Zb', 'w', 'Fx'}
mks = [repmat( 1E-2, 1, 7 ), 1E-3*9.81, 1];
%%
val = @(data, str) data(:, strcmp( cols, str) )*mks( strcmp( cols, str ) );
%%
nan_mean = @(x) mean( x(~isnan(x) ) );
nan_std = @(x) std( x(~isnan(x) ) );

figure(sum(mfilename)); set( clf, 'name', mfilename );
subplot(1,3,1);
plt = plot( val( sides_smooth, 'Ly' ).*val(sides_smooth, 'Zf' ),
val(sides_rough, 'Fx' )./val(sides_smooth, 'Fx' ), 'or' );
xlabel( 'frontal area (m$^2$)', 'interpreter', 'latex' );
ylabel( '$F_\mathrm{rough~ sides}~~~ / ~~F_\mathrm{smooth ~surfaces}$',
'interpreter', 'latex' );
```

```
title( sprintf( 'avg. force ratio %.2f \\pm %.2f', nan_mean( get( plt,
'ydata' ) ), nan_std( get( plt, 'ydata' ) ) ) );
text( min(xlim), max(ylim), {'';' \bfa'}, 'fontsize', 17 )

subplot(1,3,2);
plt = plot( val( bottom_smooth, 'Ly' ).*val(bottom_smooth, 'Zf' ),
val(bottom_rough, 'Fx' )./val(bottom_smooth, 'Fx' ), 'or' );
xlabel( 'frontal area (m$^2$)', 'interpreter', 'latex' );
ylabel( '$F_\mathrm{rough ~bottom}~~~ /~~ F_\mathrm{smooth ~surfaces}$',
'interpreter', 'latex' );
title( sprintf( 'avg. force ratio %.2f \\pm %.2f', nan_mean( get( plt,
'ydata' ) ), nan_std( get( plt, 'ydata' ) ) ) );
text( min(xlim), max(ylim), {'';' \bfb'}, 'fontsize', 17 )

subplot(1,3,3);
plt = plot( val( front_smooth, 'Ly' ).*val(front_smooth, 'Zf' ),
val(front_rough, 'Fx' )./val(front_smooth, 'Fx' ), 'or' );
xlabel( 'frontal area (m$^2$)', 'interpreter', 'latex' );
ylabel( '$F_\mathrm{rough ~front}~~~ / ~~F_\mathrm{smooth ~surfaces}$',
'interpreter', 'latex' );
title( sprintf( 'avg. force ratio %.2f \\pm %.2f', nan_mean( get( plt,
'ydata' ) ), nan_std( get( plt, 'ydata' ) ) ) );
text( min(xlim), max(ylim), {'';' \bfc'}, 'fontsize', 17 )

set( gcf, 'papersize', [10 4], 'paperposition', [0 0 10 4] );
saveas( gcf, strcat(mfilename, '.pdf' ) );
open( strcat(mfilename, '.pdf' ) );
```

compare sandpaper.m MATLAB CODE: This code was also utilized to compare the sandpaper results with the non-sandpaper results, producing the graphs in Figure 19.

```
function compare_sandpaper
%%
% data = struct( 'smooth', [], 'sides60', [], 'front40', [], 'bottom40', []
);
% mu = struct( 'smooth', .45, 'sides60', 1.43, 'front40', 1.4, 'bottom40',
1.4;
%
% save( strcat( mfilename, '.mat' ) );

load( strcat( mfilename, '.mat' ) );
% data([10 11],:) = [];
% row_offset = 26;
%%
cols = {'Lx', 'Ly', 'Lz', 'Zf', 'Zsf', 'Zsb', 'Zb', 'w', 'Fx'}
mks = [repmat( 1E-2, 1, 7 ), 1E-3*9.81, 1];
%%
val = @(type, str) data.(char(type))(:, strcmp( cols, str) )*mks( strcmp(
cols, str ) );

%%
rhog = 668*9.8;
%%
```

```
clf;
[~, a, b] = intersect( data.smooth(:, ismember( cols, {'Lx', 'Ly', 'Lz' } )
),  data.sides60(:, ismember( cols, {'Lx', 'Ly', 'Lz' } ) ), 'rows' );
[data.smooth(a, ismember( cols, {'Lx', 'Ly', 'Lz', 'Zf', 'Fx' } ) ) ,
data.sides60(b, ismember( cols, {'Lx', 'Ly', 'Lz', 'Zf', 'Fx' } ) ) ]
subplot(1,3,1);
plot( 1:numel(a), data.smooth( a, ismember( cols, {'Fx' } ) ), 'or-' );
hold on;
plot( 1:numel(a), data.sides60( b, ismember( cols, {'Fx' } ) ), 'sk-' );
title( 'smooth rough sides' );
%%
[~, a, b] = intersect( data.smooth(:, ismember( cols, {'Lx', 'Ly', 'Lz' } )
),  data.front40(:, ismember( cols, {'Lx', 'Ly', 'Lz' } ) ), 'rows' );
[data.smooth(a, ismember( cols, {'Lx', 'Ly', 'Lz', 'w', 'Fx' } ) ) ,
data.front40(b, ismember( cols, {'Lx', 'Ly', 'Lz', 'w', 'Fx' } ) ) ]
legend( {'smooth', 'rough'} );
subplot(1,3,2);
plot( 1:numel(a), data.smooth( a, ismember( cols, {'Fx' } ) ), 'or-' );
hold on;
plot( 1:numel(a), data.front40( b, ismember( cols, {'Fx' } ) ), 'sk-' );
title( 'smooth rough front' );

[~, a, b] = intersect( data.smooth(:, ismember( cols, {'Lx', 'Ly', 'Lz' } )
),  data.bottom40(:, ismember( cols, {'Lx', 'Ly', 'Lz' } ) ), 'rows' );
[data.smooth(a, ismember( cols, {'Lx', 'Ly', 'Lz', 'w', 'Fx' } ) ) ,
data.bottom40(b, ismember( cols, {'Lx', 'Ly', 'Lz', 'w', 'Fx' } ) ) ]
subplot(1,3,3);
plot( 1:numel(a), data.smooth( a, ismember( cols, {'Fx' } ) ), 'or-' );
hold on;
plot( 1:numel(a), data.bottom40( b, ismember( cols, {'Fx' } ) ), 'sk-' );
title( 'smooth rough bottom' );
```

MATLAB Code lego_analysis.m: This code was utilized to analyze all of the data collected from the experiments.

```
function lego_analysis
%%
choice = 'smooth lego.mat';
choice = 'grit 60 sides.mat';
% choice = 'grit 40 front.mat';
if ~exist( choice, 'file' ) | ~true
    data = [];
    save( choice, 'data', 'mu', 'row_offset' );
end

load( choice );
% mu = [1 1 1];
% data([10 11],:) = [];
% row_offset = 26;
%%
cols = {'Lx', 'Ly', 'Lz', 'Zf', 'Zsf', 'Zsb', 'Zb', 'w', 'Fx'}
mks = [repmat( 1E-2, 1, 7 ), 1E-3*9.81, 1];
%%
val = @(str) data(:, strcmp( cols, str) )*mks( strcmp( cols, str ) );
%%
```

```
rhog = 668*9.8;
d = 4.2E-3;
aug = @(L, factor) L+factor*d;
frontal_area = val( 'Lx' ).*val( 'Zf' );
[alpha, factor] = deal( [3 1 1], 1 );

force = @(alpha, factor) rhog*( ...
    mu(1)*alpha(1)*aug( val( 'Ly'), factor).*aug( ( val( 'Zf' ) + val( 'Zsf'
) ) / 2, factor).^2/2 + ...
    2*mu(2)*alpha(2)*aug( val( 'Lx' ), factor).*aug( ( val( 'Zsf' ) + val(
'Zsb' ) )/2, factor).^2/2 + ...
    mu(3)*alpha(3)*aug( val( 'Lx' ), factor).*aug( val( 'Ly' ), factor).*aug(
( val( 'Zf' ) + val( 'Zb' ) )/2, factor)...
    );
% force = @(alpha, factor) rhog*( ...
%     mu(1)*alpha(1)*aug( val( 'Ly'), factor).*aug( ( val( 'Zf' ) + val(
'Zsf' ) ) / 2, factor).^2/2 + ...
%     2*mu(2)*alpha(2)*aug( val( 'Lx' ), factor).*aug( ( val( 'Zsf' ) + val(
'Zsb' ) )/2, factor).^2/2 ) + ...
%     mu(3)*val( 'w' );
force_schiffer = @(alpha, factor) rhog*( mu(1)*alpha(1)*aug( val( 'Lx'),
factor).*aug( val( 'Zf' ), factor).^2 )/2;

err_fnc = @(force, alpha_factor) mean( (val('Fx') - force(
alpha_factor([1:3]), alpha_factor(end) )).^2 )

% [xfinal,ffinal,exitflag,xstart] =  rmsearch(  @(alpha_factor)
err_fnc(force, alpha_factor), 'fminsearchbnd', [4 1 1/2 3], [0 0 0 0], [5 5 5
1] );
[alpha_factor,ffinal,exitflag,xstart] =  fminsearchbnd( @(alpha_factor)
err_fnc(force, alpha_factor),  [4 1 1/2 3], [0 0 0 0], [10 10 2 0] );
[alpha_factor_schiffer,ffinal_schiffer,exitflag,xstart] =  fminsearchbnd(
@(alpha_factor) err_fnc(force_schiffer, alpha_factor),  [4 1 1/2 3], [0 0 0
0], [10 10 10 0] );

%%
figure(sum(choice)); set( clf, 'name', choice );

subplot( 2,2,1);
plot( frontal_area, val( 'Fx'), 'or' );
hold on;
%     text( frontal_area, val( 'Fx' ), num2str( shiftdim( 1:size( data, 1) )
+row_offset ), 'horizontalalignment', 'left' );
plot( frontal_area, force(alpha_factor(1:3), alpha_factor(end) ), '*k' )
title( sprintf( 'our model, \\alpha = [%.2f %.2f %.2f]', alpha_factor(1:3) )
);
xlabel( 'frontal area (m^2)' );
ylabel( 'F_x  (N)' );
legend( {'measurements'; 'fits'} );
rel_err = @(a,b) sqrt( (a-b).^2./(a.^2 + b.^2 ) );

subplot( 2,2,2);
plot( frontal_area, rel_err( force(alpha_factor(1:3), alpha_factor(end) ),
val( 'Fx' ) ), '*k' )
xlabel( 'frontal area (m^2)' );
ylabel( 'rel. error' );
```

```
title( sprintf( 'average rel. error %.2f',  mean( rel_err(
force(alpha_factor(1:3), alpha_factor(end) ), val( 'Fx' ) ) ) ) );

subplot( 2,2,3);

plot( frontal_area, val( 'Fx'), 'or' );
hold on;
%     text( frontal_area, val( 'Fx' ), num2str( shiftdim( 1:size( data, 1) )
+row_offset ), 'horizontalalignment', 'left' );
plot( frontal_area, force_schiffer(alpha_factor_schiffer(1:3),
alpha_factor_schiffer(end) ), '*k' )
title( sprintf( 'Schiffer model, \\alpha = %.2f', alpha_factor(1) ) );
xlabel( 'frontal area (m^2)' );
ylabel( 'F_x  (N)' );
legend( {'measurements'; 'fits'} );

subplot( 2,2,4);
plot( frontal_area, rel_err( force_schiffer(alpha_factor_schiffer(1:3),
alpha_factor_schiffer(end) ), val( 'Fx' ) ), '*k' )
xlabel( 'frontal area (m^2)' );
ylabel( 'rel. error' );
title( sprintf( 'average rel. error %.2f', mean( rel_err(
force_schiffer(alpha_factor_schiffer(1:3), alpha_factor_schiffer(end) ), val(
'Fx' ) ) ) ) );
%%
set( gcf, 'papersize', [8 6], 'paperposition', [0 0 8 6] );
saveas( gcf, strcat(choice(1:4), '.pdf' ) );
open( strcat(choice(1:4), '.pdf' ) );
```

MATLAB Code parameter_space.m: This code was used to determine the parameter space for the various parallelepipeds constructed.

```
function parameter_space
%%
figure(sum(mfilename)); set( clf, 'name', mfilename );
subplot(1,2,1);

cols = {'Lx', 'Ly', 'Lz', 'Zf', 'Zsf', 'Zsb', 'Zb', 'w', 'Fx'}
mks = [repmat( 1E-2, 1, 7 ), 1E-3*9.81, 1];
load( 'smooth lego.mat' );
val = @(str) data(:, strcmp( cols, str) )*mks( strcmp( cols, str ) );
%%
plot( val('Lx')./val('Ly'), ( val( 'Zf' ) + val( 'Zsf' ) )./val('Ly')/2, 'or'
);
%     text( val('Lx')./val('Ly'), ( val( 'Zf' ) + val( 'Zsf' )
)./val('Ly')/2, num2str( shiftdim( 1:size(data, 1) )+row_offset ),
'horizontalalignment', 'left' );
xlabel( '$L_x/L_y$', 'interpreter', 'latex');
ylabel( '$\langle z_\mathrm{front} \rangle/L_y$', 'interpreter', 'latex' );
text( min(xlim)+.5, max(ylim), {'';'\bfa'}, 'fontsize', 20 )
subplot(1,2,2);
load( 'grit 60 sides.mat' );
val = @(str) data(:, strcmp( cols, str) )*mks( strcmp( cols, str ) );
plot( val('Lx')./val('Ly'), ( val( 'Zf' ) + val( 'Zsf' ) )./val('Ly')/2, 'or'
);
```

```
%     text( val('Lx')./val('Ly'), ( val( 'Zf' ) + val( 'Zsf' )
)./val('Ly')/2, num2str( shiftdim( 1:size(data, 1) )+row_offset ),
'horizontalalignment', 'left' );
xlabel( '$L_x/L_y$', 'interpreter', 'latex');
ylabel( '$\langle z_\mathrm{front} \rangle/L_y$', 'interpreter', 'latex' );
text( min(xlim)+.5, max(ylim), {'';'\bfb'}, 'fontsize', 20 )

set( gcf, 'papersize', [7 4], 'paperposition', [0 0 7 4] );
saveas( gcf, strcat(mfilename, '.pdf' ) );
open( strcat(mfilename, '.pdf' ) );
```