Project Number: ME-KZS-0209

ROBOTIC 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

by

Brian Benson

Neal Humphrey

_____ _____

Date: 4/30/2009

Approved:

_____

Prof. K. A. Stafford, Major Advisor

_____

Prof. S. A. Koehler, 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 in-depth knowledge of all things sand-swimming was an invaluable resource. In addition his guidance and eye for detail contributed to a much smoother running project.

Ineos Nova LLC and Mark Price, for their very generous donation of 450 kg of polystyrene pellets. This donation made the testing environment possible.

Brian and Cheryl Benson, for their assistance in creating the composite skin and unending support.

Russell Morin, for his unmatched dedication in the printing of all of the snake's rapid prototyped parts. His incredibly fast turnaround times made the timely completion of this project possible.

Professor Taskin Padir, for his help answering a seemingly endless stream of electrical engineering questions.

Abstract

The motivation for this project is to provide a means to study the physics of sand-swimming, which is a behavior seen in certain desert snakes. A biomimetic self-contained scalable robotic snake was designed and built with the capability to move below the surface of granular media. Its ability to match arbitrary traveling waveforms while recording data for analysis makes it a first step towards understanding the physics of sand-swimming through experimental studies.

Table of Contents

List of Tables

List of Figures

Authorship

Introduction

Motivation

Snakes are among a select few species that achieve locomotion without the use of external appendages. Nonetheless they are able to traverse a far greater variety of environments than animals with appendages. A snake's relatively thin and elongated body allows it to be stealthy, highly mobile, and enables it to overcome obstacles that other animals would find impossible. For example, snakes can travel over a variety of surfaces, swim both on water's surface and underneath, climb, jump, glide through air, bridge large spans, burrow, and even swim through sand. In the field of robotics these traits are highly desirable. Technologies related to robotics are continuing to advance and as such the areas in which robotics are applicable continue to grow. Applications may include research and surveillance of extreme environments, emergency response in hazardous or confined environments as well as Lunar or Martian exploration. Therefore research and development of robotic snakes holds much promise in the future where conventional wheeled robots are unsuitable.

A robotic snake has the potential to be the ultimate method of locomotion in terms of the variety of obstacles and environments it could traverse. Moreover, such a snake can be easily and completely sealed inside a "skin" to the outside world. This would allow it to navigate locations such as underwater or hazardous environments where volatile chemicals are present. A snake is extremely low to the ground and therefore has a uniquely low center of gravity. In addition to this a large proportion of the body lengthwise is always in contact with the ground. This means that a robotic snake could negotiate terrain with little chance rolling over or falling off. In addition the inherent design of a robotic snake lends itself well to high redundancy. If one segment fails then the remaining segments can continue to function and propel the snake forward at a slightly reduced efficiency. Finally a robotic snake could have the ability to swim in granular media, such as sand, similar to certain desert snakes, such as the Chionactis Chilominiscus. Research of biological and robotic snakes that swim through water and traverse solid surfaces has been performed over the past several decades; however there

has been minimal research on the physics underlying locomotion below the surface of sand.

Objective

The goal of this project was to provide a means to fill the discussed gap in research through the design and construction of a robotic snake capable of following arbitrary traveling waveforms through granular media for the purpose of research. As such the snake can be used to test various waveform sequences with the goal of achieving locomotion, or swimming. Specifically this robotic snake was designed to assist the co-advisor of this project, Assistant Professor Koehler at WPI, in his research of dense granular flow (Koehler, 13). This goal was accomplished by designing and building a robotic snake capable of following the shape of an arbitrary traveling wave in depths of plastic beads at least 10 cm deep.

Background Research

Biological Snake Locomotion and Physiology

Locomotion

Snakes have perhaps one of the most unique methods of locomotion and it is often perplexing to the casual observer how they can possibly move. Any type of motion results from applying forces on the environment to create reaction forces (Hu, 2). In the case of snakes, forces are applied at certain points on the terrain, called push points. One primary gait is lateral (serpentine) undulation which is used by most sand-swimming snakes. 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). All points on the snake move continuously at the same speed and experience continuous sliding with the ground. 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 want to travel through. Therefore it can be concluded that because snakes use lateral undulatory motion for movement through sand it is the most suitable method of locomotion (Dowling, 20).

The efficiency of the snake is positively proportional to the length of the snake. However there is a limit to the maximum efficiency as a function of snake length and it has been found that the fastest snakes have a length no more than 10 to 13 times their circumference. The shape of the wave that the snake produces is greatly dependent on its environment and often changes in real time as a result to changing conditions (Bauchot, 64). However it has been shown that the curvature of the body is a key element of lateral undulation and a snake pushes off the environment the greatest amount at points of highest curvature change (Dowling, 19). Other gaits include skidding, side winding, and straight or rectilinear progression, also shown in figure 1. However these gaits are not

applicable to sand swimming and therefore will not be discussed further. In some situations such as in deserts and flat terrain there are no push-points for a snake using lateral undulation and it is unknown how snakes are able to achieve the high speeds that are observed. Robotic snakes with wheels have been able to approach the speed of biological snakes on flat surfaces but it remains unexplained how snakes are able to achieve such high efficiency (Hu, 3). In a numerical simulation using experimentally obtained data a virtual snake is only able to travel at half the speed of actual snakes. This means that the numerical simulations have omitted some important physical mechanism. Experiments have shown that the ratio of forward friction and transverse friction have the greatest effect on a snake's speed. This property has been used effectively in robotic snakes in which passive wheels were added to each segment in order to reduce the forward friction while maintaining transverse (sideways) friction. In the numerical simulation it was not until values comparable to a wheeled robotic snake were used that comparable speed values for live snake speeds occur.

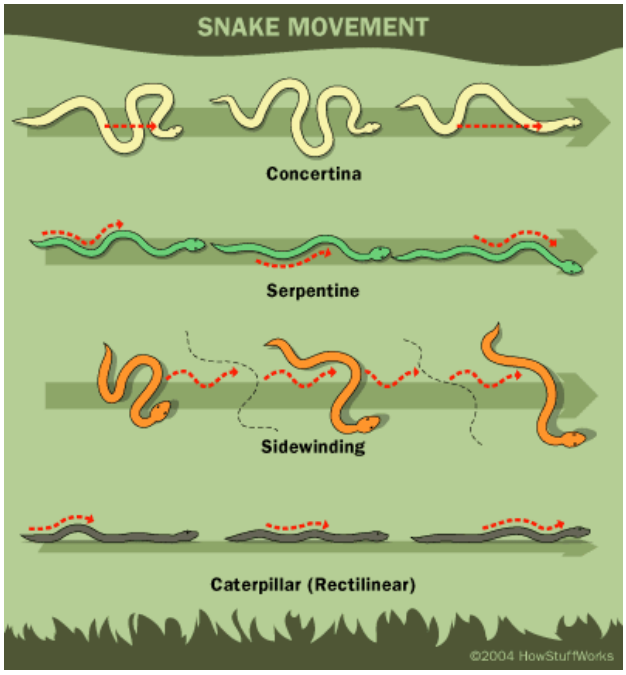


Figure 1: Snake Motions

(Howstuffworks)

Sand Dwelling Snakes

One theory as to the origins of snakes is that they are decedents of burrowing lizards. As such, it is commonly believed that snake ancestors lived in muddy environments and could burrow as well as swim. Certain modern snakes are successful burrowers including the Typhlopidae and Leptotyphlopidae, both of which are from primitive families, and some Aniliidae. These snakes are often less than three feet long, have a small compact head whose tip is a soil-boring apparatus, and have no narrowing at the neck. The Xenopeltis unicolor has a smooth textured skin which facilitates sand-swimming by acting as a dry lubricant. This provides evidence that sand swimming snakes may not rely heavily on anisotropic frictional forces. Another example of an animal that is able to move through sand is the worm lizard. This animal is serpentine in form and has cutaneous grooves or rings which make it look like a large worm. (Bauchot, 27). Other examples of sand-swimming snakes include the Glossy snake (Leptotyphlops macrorhynchus), the Sand Boa (Eryx), sand snakes like the Lytorhynchus Diadema, the Mexican Dwarf Python (Loxocemus), Colubers (Heterodon and Prosymna), Horned Viper (Cerastes cerastes), Western Shovel-Nosed Snake (Chionactis Chilominiscus) and Burrowing Viper (Atractaspididae). Figure 2 shows a western shovel-nosed snake moving through the desert and figure 3 shows the tracks of a snake swimming a few centimeters beneath the sand's surface. The Sabulicole species which includes the Lytorhynchus Maynardi lives in the sand rather than on it and has smooth keeled body scales (Bauchot 132).

Figure 2: Western Shovel-Nosed Snake (Chionactis Chilominiscus) Tracks

Figure 3: Sand-Swimming Snake Tracks

General Physiology

A snake's skeleton is unique in that it consists of at least 130 vertebrae each capable of small movements (Bauchot, 27). This can be seen in figure 4. Each segment is limited to between 15 and 20 degrees of movement side to side and only a few degrees up and down. Any rotation between vertebrae is very limited and extension is prevented by ligaments and muscles. This combination of small movements allows a python's spine to curve up to 60 degrees over 40 vertebrae. This information is critical in the development of a robotic snake because it gives a starting point for the structure and powered motion that may be necessary to imitate snake's movement (Bauchot, 61). It also serves as an example of a system where the combined action of many limited, simple components results in great abilities. Mathematical analysis by Dowling (78) found that horizontal

angular movement of joints in a robotic snake and the aspect ratio of the length to the width of each segment should be small. He found that joints between segments with the ability to pivot more than +- 20 degrees are unnecessary. This is important to realize in the design phase and matches the findings in biological snakes. The control methods that biological snakes use is also worth taking note of. The spinal cord runs through the vertebral canal and controls a number of motor functions on its own. This is important to realize because it means that some of the snake's motor control is decentralized (Bauchot, 19). This may be an important feature in any robotic snake.

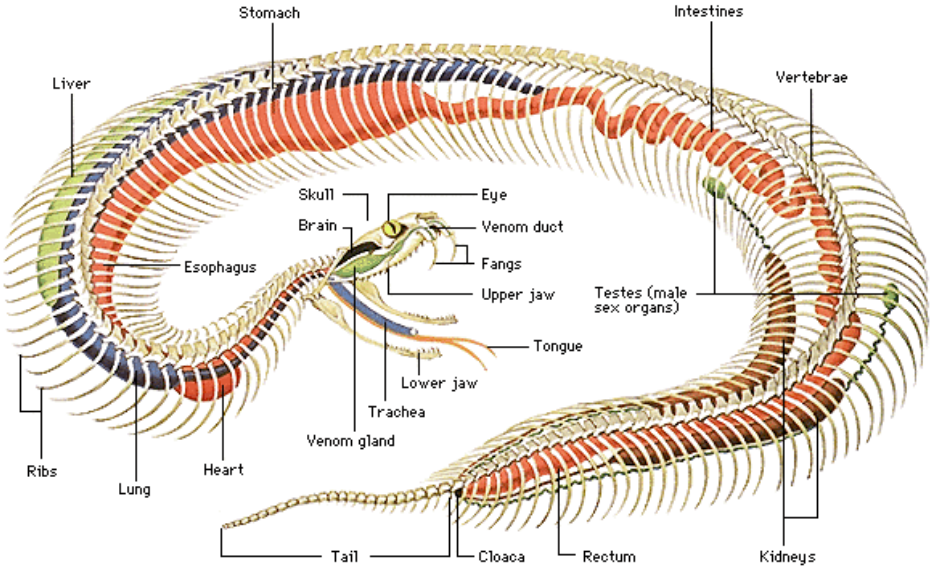


Figure 4: Snake Skeletal Structure

(WorldBook)

Snake Skin

A snake's skin plays a very important role in its locomotion. For most gaits a snake's body remains in constant contact with the ground, so in order to optimize movement, a snake's coefficient of friction is anisotropic, which means that depending on the direction of movement the force acting on the snake per unit weight is different. This is a result of the skin's surface configuration. The skin is highly elastic and made up of several layers (Dowling, 18). It is covered with scales whose backs are loose and partially covered by the scales behind them. This can be seen in figure 5. Each scale's geometry and transverse distribution allow the snake to move forward with less friction then

backwards. Additionally on the microscopic level the scales are covered with tiny indentations that create gliding tracks (Bauchot, 62). Tests show that sand skinks have a forward coefficient of friction of .3 and a reverse coefficient of friction of 1.3 on wood. These are dramatically different from one another and probably play a large role in forward propulsion (Hu, 3). However the Xenopeltis has smooth skin and can also swim through sand. Further experimentation is necessary to determine the role of the skin's frictional properties. Moreover, no studies have been performed on the role of skin friction for sand-swimming.

After performing an extensive study of different types of skins such as bellows, cable chains, flexible ducts, rubber, fabrics, and braided materials Dowling (85) decided to use a Lycra spandex sleeve over a polyethylene-based braided sleeve that is often used for wire protection.

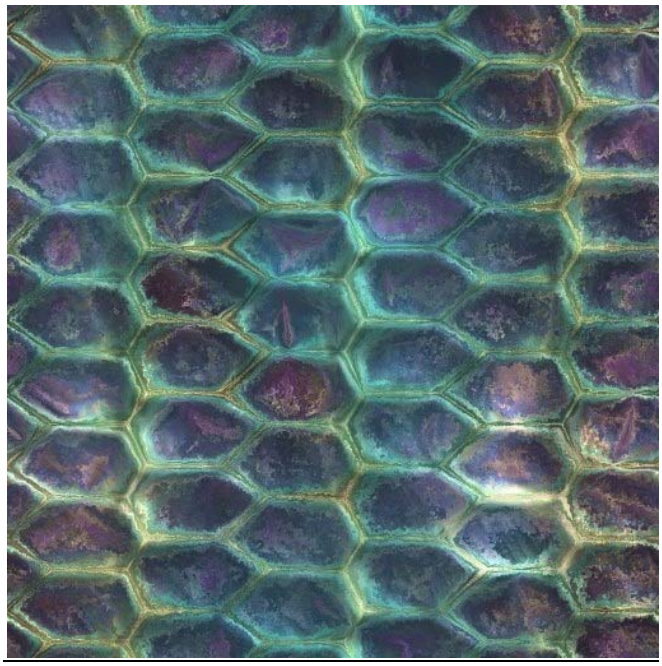


Figure 5: Snake Skin
(Flickr)

Swimming in Granular Media

The physics of swimming in granular media is poorly understood because there is no general theory for granular flow. Observations show that sand-swimming snakes employ a lateral undulatory motion, similar to snakes swimming in water. Therefore it can be concluded that there are some universal flow principals that are applicable to both granular and hydrodynamics flows (Koehler, 2). Although rudimentary experiments have shown that simple stroking strategies can result in granular swimming it is necessary to include several mechanisms such as packing configuration and force chains that are not present in hydrodynamics. Research has also shown that in particular cases viscous approaches can serve as a guide for understanding granular swimming (Bzdega, 7). Therefore it is assumed that on a very rudimentary level sand swimming can be modeled as swimming at very low Reynolds number where viscous forces are dominant in comparison to inertial forces (Bzdega, 1). Movement of snakes is supported by this when looking at the Froude number. The Froude number is the ratio of the inertia to friction of a snake. Using experimental data a Froude number of about .003 was calculated, implying frictional forces dominate over inertial ones (Hu, 7). However there are differences in high viscosity fluids and granular media. In granular media, forces are propagated along force chains unlike fluids whose stresses vary smoothly in space. Also there is no time reversibility in granular material as seen in viscous fluids. There is no cohesion and often the density is non-uniform (Koehler, 7).

Amanton's friction law states that the friction between dry sliding contacts is independent of the speed and only depends on the contact pressure between surfaces. Therefore the issue of swimming through granular media is quasi-static and has no relation to the speed at which the swimming occurs. In other words the shape change will directly affect the net displacement independent of the speed of the change. Another key difference is that in granular media any previous movements will have a significant effect on the next movement whereas for highly viscous fluids any movement is independent of any previous movements (Bzdega, 4).

Due to the differences in viscous fluids and granular media, comparisons can be drawn between the two but one cannot be used to directly model the other. Currently the

only method to perform optimization studies is through molecular-dynamic type simulations or through actual experiments (Koehler, 11).

Previous Robots

Hirose

In the early 1970's Hirose and Umetami worked on what they termed the Active Cord Mechanisms or ACMs. These were snake-like mechanisms that could perform lateral undulation. Hirose developed mathematical models of force and power as a function of distance and torque along the curve followed by the snake and then compared them to real snakes. He also developed models for the distribution of the muscular/actuator forces along the body. His models closely matched real snakes. He realized that snakes vary their weight distribution in order to optimize efficiency. Hirose also studied the relationships between amplitudes and wavelengths along with friction conditions. He built robotic snakes up to 20 segments in length. An example of one of these can be seen in figure 6. Hirose's work is probably the most complete research done on snake locomotion (Dowling, (24).

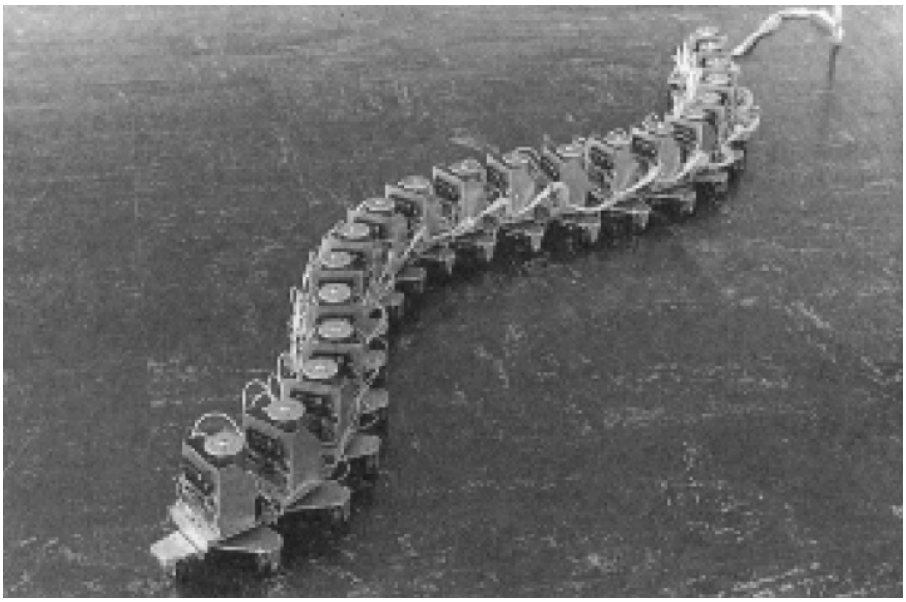


Figure 6: Hirose

(Dowling, 22)

Ikeda and Takanashi

The Japanese electronics company NEC in 1995 announced that they were developing a snake robot capable of entering the rubble typical of earthquakes and explosions. The robot used an active universal join that was specifically designed for the project. It was based on Hooke's joint. The robot consisted of seven segments and is said to be one of the best mechanical design for serpentine robots. It can be seen in figure 7. This robot is a prime example of how well a robot can be designed for packaging and modularity (Dowling 28).

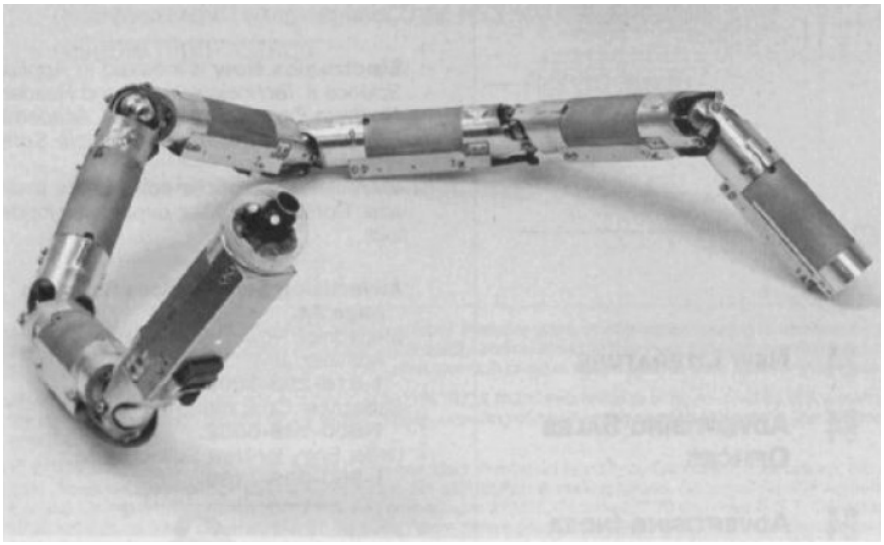


Figure 7: Hooke's Joint Snake
(Dowling, 28)

Nilsson

Martin Nilsson at the Swedish Institute for Computer Science in Sweden, developed a universal serpentine link that allowed for roll-pitch-roll movement as shown in figure 8. This was very unique and gave the joints high functionality. The robot was able to wrap itself around a pole and then climb it by rotating its segments and using them as wheels. However because this projects intent is sand swimming this functionality is not necessary (Dowling, 29).

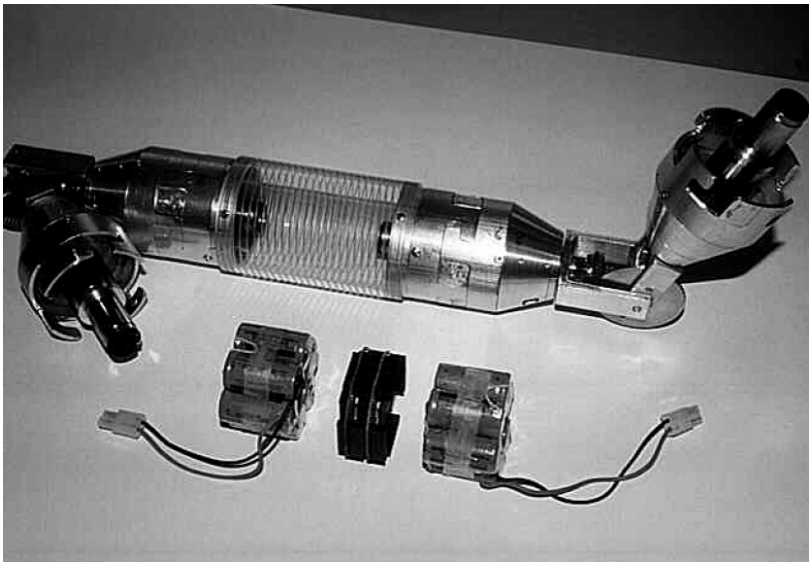


Figure 8: Nilsson's Universal Serpentine Link

(Dowling, 30)

Paap

Karl Paap at GMD in Germany along with his group created a snake-like robot that was actuated using a cable system in order to create the necessary curvature. However issues related to the complicated cable system resulted in limited locomotion (Dowling, 29). This snake can be seen in figure 9.

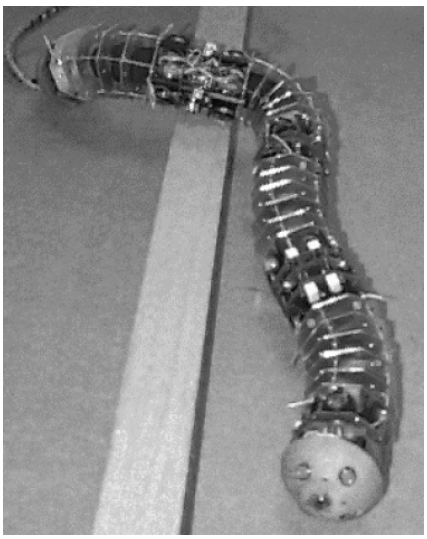


Figure 9: Paap's Cable Driven Snake

(Dowling, 30)

Dowling

Dowling created a robotic snake consisting of 20 servos, two per link, allowing for both vertical and horizontal motion. The goal of this snake was to do further research into locomotion of snake robots. He sought to fill research gaps such as skin materials and how different motions effect forward progression. After a broad range of actuation techniques Dowling choose to use hobby servos for their simplicity and power-to-weight. He ran into problems such as wiring, electrical noise between wires, and computer power for simulation. His completed project is shown in figure 10. His work provided much information, background, and lessons for setting the scope of this project (Dowling, 1...143).

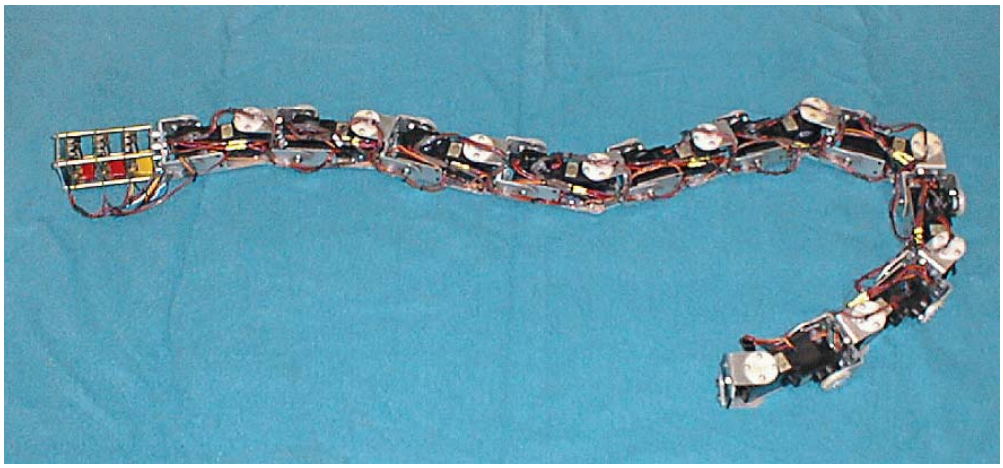


Figure 10: Dowling's Robotic Snake

(Dowling, 83)

Actuation Technologies

There are a number of technologies available for actuation in mobile robotics. Each of which has its own set of benefits and limitations. Dowling (70) explored a number of different technologies in 1997 including polymer gels, shape memory alloys, piezoelectric devices, electrostriction devices, magnetostriction, micro-electro-mechanical systems (MEMS), thermal actuators, and electro-magnetic motors. For this application many of these are not practical given their technological limitations such as magnitude of displacement. This is true for MEMS, magnetostriction, electrostriction devices, and piezoelectric devices. Polymer gels are inadequate due to issues with

strength, response, fatigue life, thermal and electrical conductivity, efficiency, power, and force densities. Thermal actuators are inappropriate because of the need to generate and release large quantities of heat and their very low efficiency. Shape memory alloys such as NiTiNOL are limited due to their short fatigue life and requirement of fast heat dissipation. Additional technologies include pneumatics through the use of pistons or bellows but the need for a constant air supply is limiting. Cable-driven system offer unique possibilities but involve equally unique difficulties including range and size requirements.

Finally electro-magnetic motors offer rotational actuation with a high power density. A variety of different motor technologies exist including permanent magnet direct current (pmdc) motors, brushless dc motors, stepper motors, alternating current motors, and servo motors which are typically pmdc motors with built in closed-loop control. These different types of motors come in a variety of different sizes and price points. They are commonly used in commercial, hobby, and industrial products. Because they are often mass produced they are optimal in terms of cost effectiveness and redundancy. Additionally performance requirements have led to high power to weight/size ratios. There are also many off the shelf components available for their control.

Control Methods

The goal of this project was to build a robotic snake capable of closely approximating arbitrary traveling waveforms which presents several challenges. 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. These challenges have many potential solutions. Available options range from having the robot tethered with each joint controlled by a central computer, to wireless communication with said computer and a distributed control system in the snake with sensory feedback. The following will describe possible techniques for achieving this goal and comment on their feasibility.

A tether acts as a link to the robot to supply constant signals and power. This would remove the requirement for batteries and resolve issues of power consumption.

However, a tether is not the most elegant solution as it would add parasitic drag and generally interfere with the sand-swimming. Thus an un-tethered wireless version is much more desirable but adds the complexity of batteries. Batteries require recharging and lead to unavoidable downtime.

In order to achieve wireless communications there are a variety of technologies available. These technologies include 802.11b, 802.11g, and Bluetooth. 802.11b and 802.11g are both powerful wireless technologies. However, the amount of hardware and setup needed to implement either one on the snake would be difficult. Also 802.11b and 802.11g use a range of frequencies, but do not change them on the fly. As a result it can cause the wireless communication to fall victim to gaps, often called shadow fading, in the spectrum created by the beads. The most readily available and easiest to use is Bluetooth. This is an established and well-documented technology, making it simple and easy to use. Bluetooth uses a technology call frequency hopping, where the frequency through which it communicates constantly changes. This allows for much more robust communication in an environment where shadow fading is a common occurrence. Another advantage of using Bluetooth communication is that a variety of different modems with high data rates are available at low prices. One key disadvantage of Bluetooth is that with increasing separation between transmitter and receiver the data rate drops and reliability degrades. The data rate is directly related to the distance and medium the signal has to pass through. As the distance increases or the medium becomes less transparent to radio waves the slower the data rate.

This varying data rate introduces the problem of the user's detailed commands bottlenecking at the wireless transmission point. In order to solve this problem the Bluetooth can be reserved for high level commands such as "begin test" or "stop" which need to be transmitted fairly infrequently. Thus the robot will operate semi-autonomously, and during down-time instructions can be downloaded and sensory feedback data uploaded.

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 a master controller capable of controlling the local slave processors for each segment. Dedicated processors for each segment will free up processing time, resulting in

more robust control and allowing for more complicated instructions and more comprehensive sensory feedback from each of the servos.

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. The master controller would then communicate with each slave processor and inform them of their sequence of moves. The moves would be choreographed by the master processor by broadcasting which step or column the snake is at. The slave processors would then move to the appropriate location as indicated by the lookup table.

Communication between the master and slave processors can be accomplished in a variety of ways. One of these methods is serial communication. Serial is simple to use and fairly robust. However, if used then if one link fails, then the rest will as well. 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. This ensures that data is not lost. However this has limited use during the actual running of the motion because the robot should not stop completely if one of the segments fails. Limitations of I2C include that number of links that can be included and the data rate that can be achieved. While I2C can attain data rates of over four Mbps, it is not as high as other technologies. Also, there is a limitation of the length of the cable that is used for communication. This limit typically is a couple meters.

Having distributed processing would allow for greater flexibility in resource allocation. It would allow each slave processor to log servo current and position data. In addition it could monitor the battery voltage to ensure it does not drop too low. This would allow for post-experiment data analysis and the ability to use real-time closed loop controls.

In order to save development time and money, the slave processor has the option of being designed identical to the master processor in terms of hardware. The only difference between the master and slaves would be the population of a few additional

components on the circuit board and a different program. Many options for embedded processing were explored. These include the PICAXE18X and dsPIC30F4013. Benefits of the PICAXE include its small size and simplicity; however the PICAXE does not have equivalent functionality when compared to other processers. It does not have on board EEPROM and the number of external pins are limited. Strengths of the dsPIC30F4013 are its capabilities and power consumption. The dsPIC30F4013 is a powerful processor that includes an ADC, hardware PWM generator, non-volatile memory, hardware I2C, and hardware UART. However, it draws 120 mA of current while operating and as a result runs at a fairly high temperature.

Fused Deposition Modeling (FDM)

Due to the perceived modeling complexity of each snake robot segment, the preferred method of manufacturing for the body of the robotic snake is through rapid prototyping technology. Worcester Polytechnic Institute has a 1200ES fused deposition modeler with a workspace of .254x.254x.305 meters. It has the ability to print using ABSplus plastic in layer thicknesses of .01 or .013 inches with the Z axis being the least accurate. A maximum deviation of .012 inches can be seen. This method of 3d printing consists of extruding thermoplastic as a semi-molten filament which is deposited layer-by-layer to build the prototype. It has been reported that the ABS prototypes have demonstrated strengths 60-80% of typical injection molded ABS parts. FDM parts do not change with time or environment exposure unlike processes like stereolithography and PolyJet and can withstand temperatures up to 200 degrees F. In addition FDM parts can be milled, drilled, tapped, and turned with little consideration (Grimm, 1...6). The largest drawback with FDM parts is that they are anisotropic. The printing process results in layers of materials. Depending on the orientation of the layers the printed parts will be stronger in one direction then another so care must be taken in the design process to account for this.

Methodology

Project Scope

Background research provided the necessary information to appropriately set the project scope. A thorough literature review of previous snake robots gave insight into different challenges that snake robots encounter and how they can be overcome and avoided. Understanding the physics of granular media allowed for a good understanding of the requirements a sand-swimming robotic snake would need. A broad review of actuation technologies gave a variety of options to provide mechanical power to the snake. Determination of different options for control methods allowed for a realistic approximation of what would be possible for this project. These prior steps allowed for a set of assumptions to be drafted that dictated what the project objectives included and didn't include. These assumptions and objectives were as follows:

- Build a biomimetic robot that matches the basic qualities of a biological snake with consideration to available resources.
- Snake skin and musculature are very specialized and highly refined. It is not possible within the scope of this project to replicate it. However attempts will be made to be reasonably accurate.
- The intent of this project is to create a scalable self-contained robotic snake capable of being programmed to approximate an arbitrary traveling waveform. To avoid locomotion interference wireless operation is desired. However due to potential problems, such as power requirements or instruction transmission, the snake may need to be tethered.

All of this information in addition to the project objectives and assumptions allowed for the creation of a specific set of project parameters and specifications.

Design Specifications

Design

- Modeled after the general anatomy of biological snakes.
- Minimum 12 segments
- Length of the body is less than 13 times the circumference of the snake
- Total length of snake should not exceed 1.82 meters (6 feet)
- Perform lateral undulatory motion at a depth of at least 15 cm (6 inches)
- Skin that keeps the granular media out of the moving joints without restricting movement
- Self-contained (wireless) or minimal tether
- Scalable
- Commonality and modularity between each segment
- Minimum 30 minute run time
- Data Collecting
    - Orientation
    - Joint Angular Positions
    - Torque
    - Battery Voltage

Operation

- Must be safe to use for someone skilled and trained in its operation (pinch points etc)
- Cannot use or require anything toxic or hazardous to anyone's health
- 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.

Manufacturability

- Commercially available materials
- Off the shelf component preference
- Manufactured using standard techniques

Resources

- Cost less than $2000

Design Specifications Discussion

When trying to design something it is often best to look at what has been done in the past and try to use that as a starting point to develop ideas. In this case the desired robotic snake is intended to mimic its biological counterpart. For this reason it was decided that the robotic version should imitate the general anatomy of biological snakes as closely as possible.

Biological snakes typically have upwards of 130 vertebrae which allow them almost unparalleled flexibility. However given the resources of this project this would not be reasonable to imitate. Therefore it was decided that because of scalability, building a minimum number of segments would be appropriate. Two considerations had to be taken into account when determining this minimum number of segments. These are the minimum desired waveform, and resolution possible in the snake's approximation. A period and a half of a sine wave was determined to be the minimum form and in order to achieve the desired resolution 4 segments per half period were necessary which meant that 12 segments were required.

Research has shown that the most locomotion efficient snakes have an overall length of less than 10-13 times the snake's circumference. In order to make the snake manageable and able to be tested in available labs at Worcester Polytechnic Institute a total snake length of less than 1.8 meters was necessary.

Previous research has shown that sand-swimming snakes use a type of horizontal undulatory motion when swimming. Therefore the snake needs at minimum to be able to perform these same types of motions. In order to perform the required testing the robotic snake needs to be able to swim at a depth of at least 15 cm (6 inches) to ensure that it stays submerged throughout testing.

Moving in granular media creates the challenge of requiring a sealed system to keep the grains out of the moving mechanisms of the snake. To do this a skin of some type is necessary. However the skin cannot overly restrict the movement of the snake. For example greatly increasing the torque requirements of each segment or limiting the snake's range of motion is not desirable.

The snake needs to be self-contained (i.e. wireless) because any types of tether would interfere with the swimming motion and associated experimental results. In

addition the overall snake length necessary to perform swimming motions is unknown. After experimentation it may be determined that there is an optimum length and waveform for swimming. However this cannot be known until such experiments can take place. Therefore the snake needs to be scalable so that it can operate with as few or as many segments as desired. This requirement includes the ability to easily increase or decrease the number of segments without any drastic changes to the snake. For example having to add wires to the entire snake or modify a pre-existing segment in order to add one more vertebrae would not be acceptable.

To increase the simplicity and scalability of the snake modular and identical segments are necessary. The segments should be easy to put in and remove and the order of the segments are should not affect the overall functionality of the snake.

This snake will be self-contained which typically means it will be battery powered. However this introduces the problem of battery life. The end goal of the snake is to perform experiments for research purposes. Therefore a minimum run time of 30 minutes is required.

In order to maximize the capability of the snake for research purposes it needs to have data logging capability. The snake needs to be able to monitor and log the torque required, position, and battery voltage for each segment. In addition the global orientation of the snake is helpful.

The final snake needs to be safe to use and not cause injury in ordinary operating conditions. Therefore it must be safe to use for someone skilled and trained in its operation. Unnecessary pinch points, sharp edges etc are unacceptable. The snake cannot use or require anything toxic or hazardous to anyone’s health.

The goal of the snake is to be used for research purposes. Therefore it needs to be able to be operated by someone who is not intimately familiar with all of details of the snake. They should be able to do all desired testing after sufficient training. Therefore 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 limited budget to create the robotic snake is $2000. This means that each segment should cost no greater than roughly $166.

General Design Decisions

The development of the project objective, background research, and design parameters allowed for the general design decisions to be made. These decisions further narrowed the project parameters and scope as to allow for initial design work to begin.

The first of these decisions was the actuation method. 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 reviewed, electro-magnetic motors were chosen for their high power to weight and cost ratio. A review of different motor types was performed and hobby servos were found to be the best option. They offer the greatest functionality per size and cost of all of the options. Typically electric motors rotate at speeds in excess of a thousand revolutions per minute. However for this application much smaller speeds are required. Hobby servo motors have a gear reduction built in. In addition they offer closed loop control. They require only a steady voltage source, and signal wire for movement. Whereas other options typically require a second circuit to control the voltage input to the motor in order to control its speed. Since closed-loop control is necessary for the snake robot, an encoder and the associated logic is needed.

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 using a sheet metal type approach in which the initial material is some sort of metal sheet, which is then cut and bent to the desired shape. The final method is through the use of a rapid prototyping machine. As discussed in the background research section Worcester Polytechnic Institute now owns a fused deposition modeler. 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 created. After the design of the desired component is completed in a solid modeling program such as Solidworks it can be exported to the machine which then simply prints it. 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	n/a
Manual Lathe	Turning	n/a
Various HAAS Mills	CNC Milling	n/a
Various HAAS Lathe’s	CNC Turning	n/a
Dimension 1200ES fused deposition modeler	Rapid Prototyping	4 day turnaround time
Waterjet	Cutting 2D shapes	3 weeks
Various hand tools etc	n/a	n/a

After a comparison of the available manufacturing resources available a decision was made to use the rapid prototyping machine. The reasoning behind this decision is its ease of use, speed, and unparalleled flexibility in respect to imposed design constraints.

To accomplish wireless control, Bluetooth technology was found to be ideal. It was decided that wireless communication could be used for high level commands and data transmission when speed isn’t as critical. This meant that the snake needed a master controller capable of receiving commands from the computer and controlling slave controllers for each segment. The master controller would take care of all low level commands. This distributed control allowed for more redundancy, increased scalability, and improved performance.

Preliminary Testing

Determining Required Torque

Before servos could be selected to actuate each joint, the required torque at each joint had to be known. The force required to move through granular material increases linearly with depth. However the nature of granular media and flow is such that the governing equations are limited in their accuracy due to the narrow number of parameters that they take into account. For this reason a theoretical mathematical model was created to calculate the required torque, and experimental testing was performed for comparison and analysis.

Testing Methodology

A testing methodology was created with the goal of having the results match the final application as closely as possible. To do this a cylinder had to pivot and move through granular media at various depths. This was accomplished with a custom testing rig. The top view of the design can be seen in figure 11.

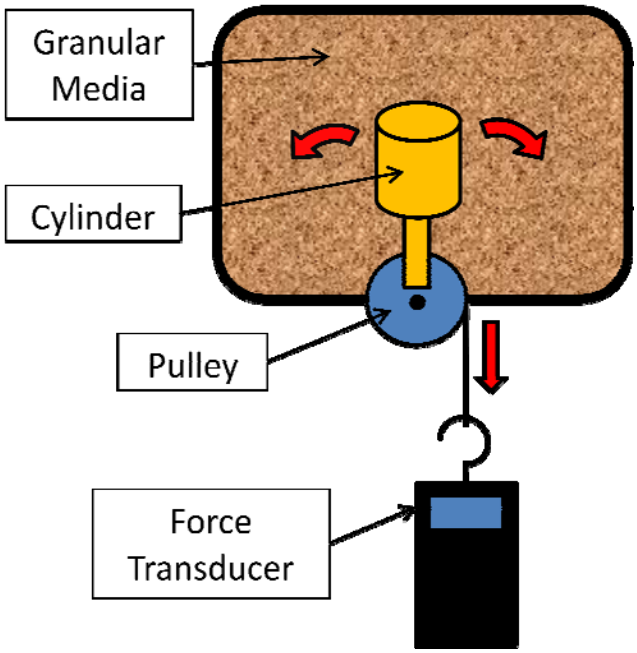


Figure 11: Torque Analysis Testing Rig

The basic concept behind the design is that there is a rectangular container capable of holding various depths of granular media. On the edge of this container is a bearing block holding a shaft and pulley such that the pulley is parallel with the ground plane. Attached to this pulley is an arm. A cylinder is then attached at the end of the arm. The point of the arm is to move the cylinder away from the wall to reduce the wall effects. A string is then fixed to the pulley and is attached to a force transducer. With this testing rig the cylinder can be buried at various depths and rotated through the media by pulling on the force transducer. The force transducer will output a force reading which can then be converted into the required torque.

Theoretical Mathematical Model

A mathematical model was created to calculate the theoretical torque required for the aforementioned testing rig. As a basis for the mathematical model the governing equations for an object moving through granular media and basic moment definitions were used. From these an equation was derived that allowed for the torque required at a joint to be calculated as a function of depth.

Definitions:

$$\rho_{eff} = effective\ density$$

$$g = gravity$$

$$z = depth$$

Derivations:

$$df = \rho_{eff}\ g\ z\ dx\ dz$$

$$dM = r\ df$$

$$M = \int_{x1}^{x2} (r)\ df = \int (\rho_{eff}\ g\ z\ \ x)\ dx\ dz$$

$$M = (\rho_{eff} g\ \frac{({x_2}^2 - {x_1}^2)}{2})\ \int_{z1}^{z2} z\ dz$$

$$M = (\rho_{eff}\ g\ \frac{({x_2}^2 - {x_1}^2)}{4})\ (({z_2}^2 - {z_1}^2))$$

Where,

$$x_2 = distance\ to\ cylinder\ end$$

$$x_1 = distance\ to\ cylinder\ beginning$$
$$z_2 = depth\ of\ bottom\ of\ cylinder$$
$$z_1 = depth\ to\ top\ of\ cylinder$$

Experimental Testing

The testing rig was then built in order for experimental testing to be performed. The container aspect of the rig was Poland Springs water container with the top cut off to create an open top rectangular container. A square slot was then cut out of the side roughly 10 cm up and a polyethylene bracket was attached with a matching slot. A vertical hole running down the center of the bracket contained a .625mm pin that a pulley and arm could rotate about. The arm was a 2.5 cm wide 1 mm thick piece of aluminum 10 cm long. The arm was screwed to a 28 mm diameter plastic pulley. Wrapped around the pulley and secured with a screw was 30 cm of high strength braided string. At the end of the arm a 6 cm diameter by 7 cm long piece of PVC piping was attached via two 6 mm bolts. The ends of the PVC was then sealed using duct tape. The container was filled with various depths of 6mm plastic beads with a density of 1g/cm^3. A Berkley digital tension force gauge was then used to pull on the string and determine the torque required to begin to pivot the PVC through the granular media. This experimental setup can be seen in figure 12. The bracket and associated pulley and string can be seen in figure 13.

Figure 12: Torque Testing Experimental Setup

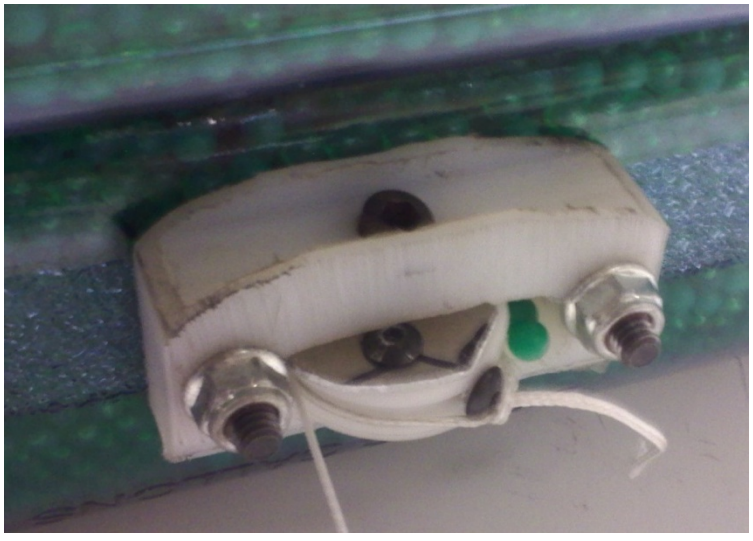


Figure 13: Torque Testing Experimental Pivot Bracket

Results and Analysis

The results of the experimental testing can be seen in figure 14. As can be seen a minimum of 5 tests were done per depth and five depths were tested. The tested depths include 6.4, 8.9, 11.4, 14 and 19 cm, measured from the bottom of the PVC cylinder.

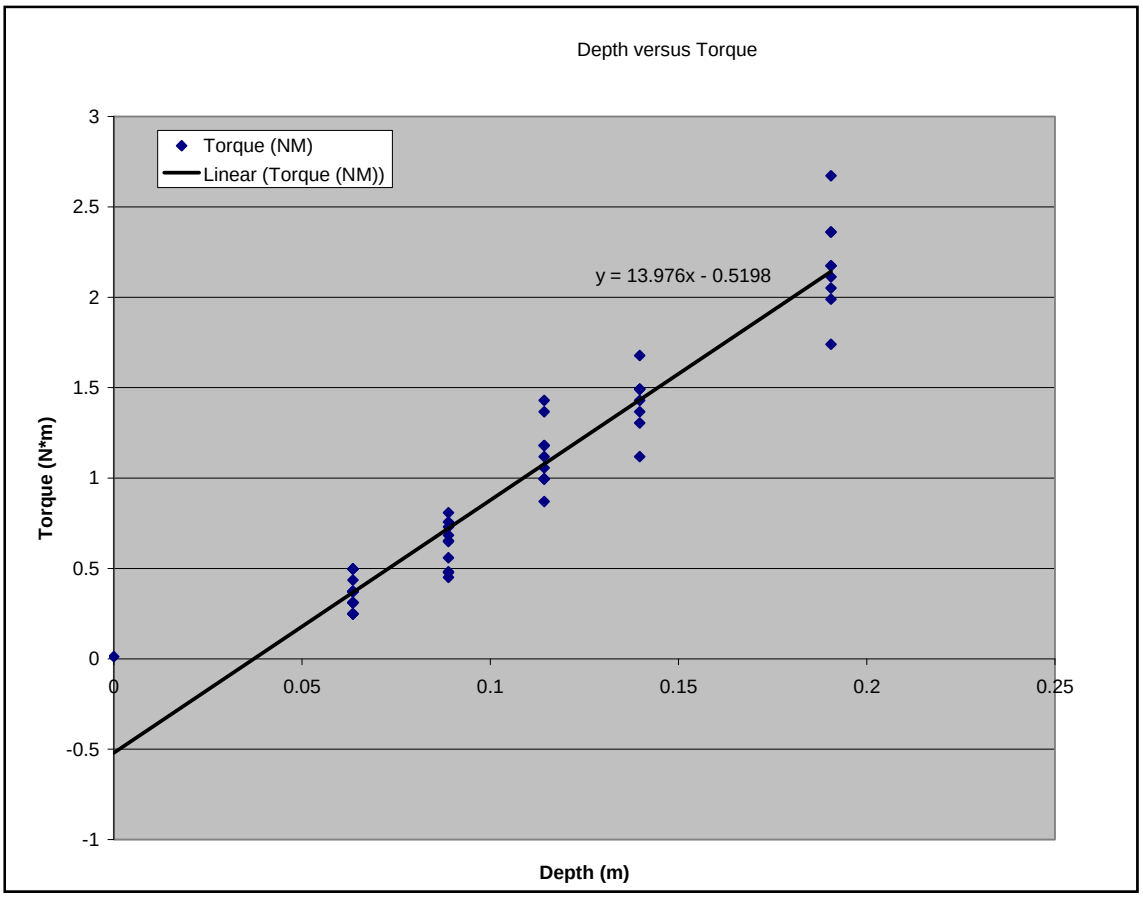


Figure 14: Depth versus torque required to pivot PVC pipe through beads

As expected the results were linear. The torque required to move through the material increases at a rate of 13.9 Newton meters per depth of a meter. Next the theoretical required torque was calculated.

Using the derived equation and values used for experimental testing the theoretical values were calculated for the experimental setup. This allowed for a direct

comparison between theory and practice. This was done in MathCAD; the code for doing so was as follows:

$$Dia := 2.5in = 0.064m$$

$$Radius_{pulley} := .55in = 0.014m$$

$$Density_{plastic} := 1{\cdot}.64\frac{gm}{cm^3}$$

$$x_1 := 1in = 0.025m$$

$$z_2(z) := z$$

$$x_2 := 4in = 0.102m$$

$$z_1(z) := z_2(z) - Dia$$

$$Torque_{th}(z) := \frac{Density_{plastic}\cdot g\cdot\left(x_2^{\,2} - x_1^{\,2}\right)}{4}\cdot\left(z_2(z\cdot m)^2 - z_1(z\cdot m)^2\right)$$

The theoretical and experimental results were then directly compared using MathCAD's graphing tools; this can be seen in figure 15.

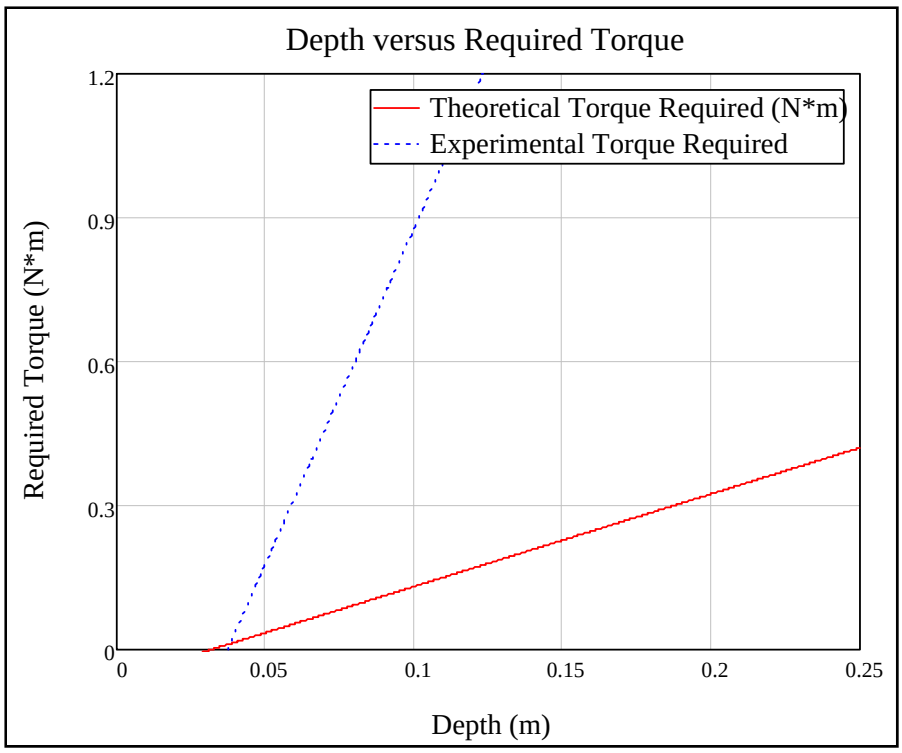


Figure 15: Theoretical and Experimentally Determined Torque Required

As can be seen the experimental results are far greater than the theoretical results. A quantitative comparison showed that experimental is 5 times theoretical. This can be attributed to a number of things. The first of which is the arm that attaches the cylinder and the pulley. The theoretical model does not take the arm into account, although it does have a very small cross-sectional area. Moving the arm requires additional torque to displace the beads. Wall affects may also play a large role in the discrepancy. The walls of the container act effectively as an infinite counterforce when the media pushes up against it. This causes beads to have to move directly up rather than sideways. The theoretical model assumes an infinitely large container. Finally the frictional forces between the granular media are not taken into account in the theoretical model. It takes only the displacement of the weight of the beads into account, not the shape, size, or material of the grains. For example it has been found that larger grain sizes can result in greater force required. Another hypothesis for the discrepancy is that the governing

equation used for granular flow is only intended for modeling a flat surface, versus a curved one as we are using.

Detailed calculations for the required torque can be found in appendix A.

Wireless Bluetooth Transmission Range

One of the challenges this project faced was wireless communication with the snake. Wireless Bluetooth technology was chosen as the best method to communicate with the snake, but it had to be verified that the technology would in fact work as intended. To do this a BlueSMIRF Gold Bluetooth Modem was purchased from www.sparkfun.com. This was then placed in a 38L bucket filled with plastic beads. A laptop was used to connect with the modem and connectivity was tested at various ranges. A satisfactory signal was obtained at up to 6 meters through a .3 meter thick wall. At a distance of 9.1 meters and 2 walls the connection was lost. From this testing it was determined that the Bluetooth modem would be perfect for this application.

Component Selection and Design

Component selection and design is always a very iterative process, as was true for this project. As the design became more developed it led to certain parts working and others requiring a second look. For the snake vertebrae the servo used to actuate each joint was the alpha component. The alpha component is the part from which you base the rest of the design. It can be changed, but it dictates the requirements of a majority of the rest of the design. After choosing the servo the gearing was decided. Next the appropriate batteries for each segment were selected. These parts set the overall size and shape of each vertebra. From here the available space for the circuit board was known and that could be laid out. Finally with all components selected and the general layout known the wiring layout was developed and necessary changes to the design were made. This entire process was extremely iterative with many parts of the design remaining fluid while the finalized component selection was worked out.

Servos

The primary criteria for selecting the servo was cost, size, and stall torque. Also because this is a high torque application great preference was given to servos with metal

gears. The required joint torque at a depth of 10 cm or is .6 Newton meters (85 oz-in). Servos typically come with the capability for either 90 or 180 degrees of rotation. To achieve 45 degrees of rotation at each joint this meant that the maximum additional gear reduction possible is 2:1 and 4:1 respectively. Servos are rated by their stall torque, however this is the absolute maximum torque they can output. If an electric motor is run at stall for too long it can overheat and fail. Therefore the maximum torque capability of the servos was considered to be 75% of the rated stall torque. This meant that servos with 90 degrees of rotation need at minimum to be rated for a stall torque of .395 Nm and for 180 degrees of rotation .197 oz-in. Table 2 contains a list of servos reviewed and their specifications.

Table 2: Servo Comparison

Model	Range (deg)	Stall Torque @ 6V (Nm)	Cost	Max Joint Torque (Nm)	$ Per Nm
HexTronik HX12K	180	0.97	$10.16	2.92	$3.48
HS-311 Standard	180	0.35	$8.99	1.04	$8.66
HS-475HB Super Pro BB	180	0.54	$17.99	1.61	$11.17
HS-645MG Ultra Torque	180	0.94	$39.99	2.82	$14.16
RS404PD	150	0.88	$31.99	2.19	$14.61
HS-225MG Mighty Mini	180	0.47	$27.99	1.41	$19.82
HS-81MG Micro	180	0.29	$23.49	0.88	$26.62
RS403PR	180	1.38	$119.99	4.15	$28.90
HS-85 MG+ Mighty Micro	180	0.34	$30.99	1.03	$30.10
HS-5085MG	180	0.42	$46.99	1.26	$37.15

S9402 Hi-Speed MG BB	90	0.78	$74.99	1.18	$63.78
S9405 Hi-Torque MG BB	90	0.71	$69.99	1.06	$66.08
S3004 Standard Ball Bearing	90	0.40	$39.90	0.60	$66.32

As can be seen in table 2 the servos are listed by their price per max joint torque. The best deal is the HexTronik HX12K. It is 33% the cost of the next best deal and 4% the cost of the least best deal. In other words the HexTronik is three times a better deal than any other available servo looked at. In addition it is only about $1 more expensive then the least expensive servo. Taking into account the 75% of stall torque limit, the max joint torque available for the HexTronik HX12K is 2.92 Nm if a 4:1 reduction is used. This is 4.87 times the required torque, meaning that the snake can theoretically go to a depth of .74 meters. The chosen servo can be seen in figure 16 and its specifications in table 4.

Figure 16: HexTronik HX12K Servo

Table 3: HexTronik HX12K Servo Specifications

Input	6V
Type	Analog
Dimensions	1.57x1.5x.79”
Weight	1.98 oz
Gears	Metal
Spline	Futaba
Speed	.13 sec/60 degrees
Stall Torque	.974 Nm (138 oz-in)
Idle Current	Under 20mA
Average Current	1000mA
Stall Current	2000mA

Gearing

The previous servo selection allowed for the gear train analysis and selection to begin. The chosen servo has sufficient torque so that a 4:1 reduction is not absolutely necessary, however with greater the reduction comes better control and reduced battery requirement. After much iteration in the design process it was determined that the largest gear reduction possible within the space restrictions is 3.5:1. The limiting factor in gear selection was the cost of gears. The high torque requirements for this application dictated higher strength gears then would typically be used for servos. However the greater the strength of the material used the more expensive the gears become. Initially a number of gear options were researched in order to compare prices and determine suitable options. A compilation of the gears researched can be found in table 4.

Table 4: Gear Option Comparison (Items in bold are *not* stock)

Type	Material	Pitch	Tooth Count	Price	Part Number	Source
Pinion	Brass	48	15	$ 14.99	RSA48-FMG-15	www.servocity.com
Pinion	Delrin	48	18	$ 3.08	RSA48-2FS-18	www.servocity.com
Driven	Aluminum	48	80	$ 16.58	F48A76-80	www.wmbirg.com
Driven	Brass	48	60	$ 18.10	GBS-48060-10	www.smallparts.com
Driven	Steel	48	80	$ 23.25	S10A6Z-048H080	www.sdp-si.com
Driven	Delrin	48	72	$ 2.40	SPBD48-24-72	www.servocity.com
Driven	Acetal	48	60	$ 3.45	GDS-48060-01	www.smallparts.com
Pinion	Brass	32	16	$ 14.95	RSA32-FMG-16	www.servocity.com
Driven	Steel	32	60	$ 27.36	S10A6Z-032H040	www.sdp-si.com
Driven	Delrin	32	60	$ 3.20	SPBD32-34-40	www.servocity.com
Driven	**Steel**	**32**	**60**	**$5.10**	**A 1C29-Y32060**	**www.sdp-si.com**

It was found that all off-the-shelf metal gears were prohibitively expensive for the budget of this project. Therefore it was initially decided that plastic gears would be the most cost effective gearing option. In order to determine if this was possible a number of calculations were performed to determine the tooth stress that would be acting on the gears, and what materials would be reasonable given that stress. These calculations can be found in appendix B. The results of these calculations are shown in Table 5. As can be seen the only two acceptable safety factors are associated with the 32 pitch gears made of either 2024 aluminum or steel. Plastic gears are not strong enough for this application and so the initial plan had to be rethought. One of the important specifications of this project is to ensure that it is rugged and will not fail prematurely. Therefore the safety factor of 3.272 is preferable. The cheapest off-the-shelf option is from www.sdp-si.com for a price of $27.36 per gear. This is nearly three times the cost of each servo. In order to reduce the cost gear stock was found. Gear stock is a shaft with gear teeth already cut along the length of it. All that was required was to use a lathe in order to cut the gears off the gear stock to obtain the desired width. This resulted in a total cost of $5.10 per gear, about

20% the cost of other options. For the pinion gear only one option was available. The servo that the pinion gear attached to has a spline for its output shaft. In order to attach the pinion gear needed to have a mating female spline. This requirement severely limited the number of available options. The only gear meeting this requirement was from www.servocity.com, it is a brass 16 tooth gear.

Table 5: Gear Safety Factors

Pitch	Material	Safety Factor
48	Delrin	0.116
48	2024 T6 Aluminum	0.291
48	Steel	0.485
32	Delrin	0.785
32	2024 T6 Aluminum	1.963
32	Steel	3.272

Batteries

Battery selection was difficult due to the space restrictions imposed by the nature of the cylindrical shape of the segments. The initial goal was to find a 6 volt battery with at least 500mAh of capacity. Nickel cadmium and nickel metal hydride battery chemistries were preferred for their cost, reliability, ease of use, and safety. However cells of these types are typically rated for 1.2 volts. This means that to achieve 6 volts, 5 cells are necessary. After a thorough search it was found that meeting all three parameters (capacity, voltage, and size constraint) would not be possible using these technologies. Other technologies were then explored, primarily lithium polymer (LiPoly) batteries. LiPoly batteries are known for their high power-to-weight and size ratio. They typically have a rectangular form factor and each cell is rated at 3.7 volts. Therefore only two cells are necessary which results in a combined voltage of 7.4 volts. This is higher than the goal of 6 volts, however this increase in voltage will allow the servo to achieve 20% more torque. Servos have been used extensively at slightly higher voltages then rated for so this 1.4 volt increase should have no adverse affects on the daily use of the servo. The battery chosen was the cheapest LiPoly battery available that met all of the requirements. The battery was the ThunderPower 730mAH LiPoly Double Cell 2S 7.4V Prolite V2

Series purchased from www.robotmarketplace.com. It is rated for 14.6 amps continuous and 29.2 amps burst. Its size is 55x33x9.5mm and it has a weight of 1.2 oz. This battery can be seen in figure 17.

Figure 17: ThunderPower 730mAh LiPoly Battery

Circuit Board

The circuit board was designed to include the following features; voltage sense, position sense, current sense, address selection, voltage regulation, and a power off switch. Each was first simulated in Multisim, and then tested using proto-boards to verify design and component attributes. These boards are shown in figure 18.

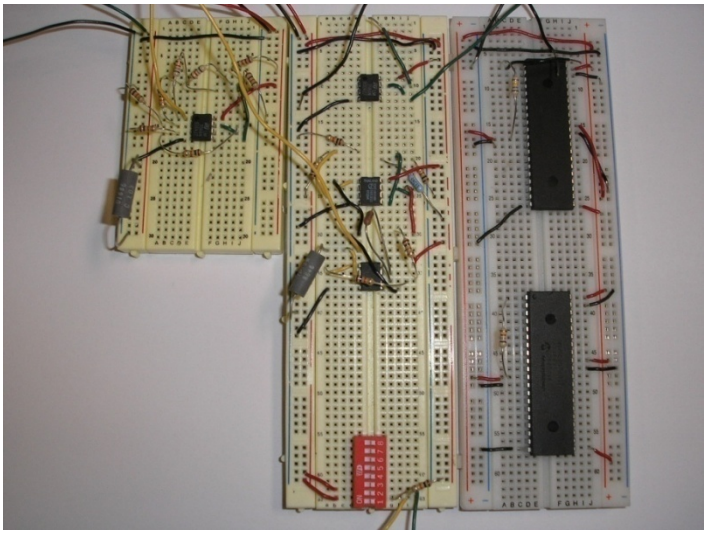


Figure 18: Proto-Board Testing

In order to implement a voltage sense circuit, the battery voltage needed to be converted to a value from 0 to 5V. Since the LiPoly batteries could not exceed a charge of 9V, then the battery voltage was taken and divided by a factor of two using a voltage divider. This would ensure that the battery voltage would be less than 5V but also give a wide range of values from 0 to 10V to be inputted as the voltage supply. To protect the microprocessor and increase the accuracy of the voltage divider a buffer was added between the divider and the ADC input. The circuit is shown in figure 19.

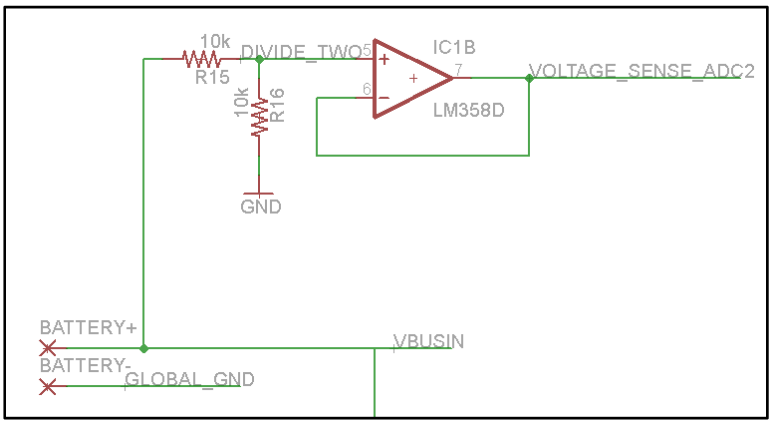


Figure 19: Voltage Sense

Because servos are being used, position feedback of each was potentially available. Each servo consists of a motor with position feedback and a control loop that will transform a pulse into a position. Since the servo already has a potentiometer on the output shaft, it can be tapped in order to gain position feedback. The location of the potentiometer output voltage is displayed in figure 20.

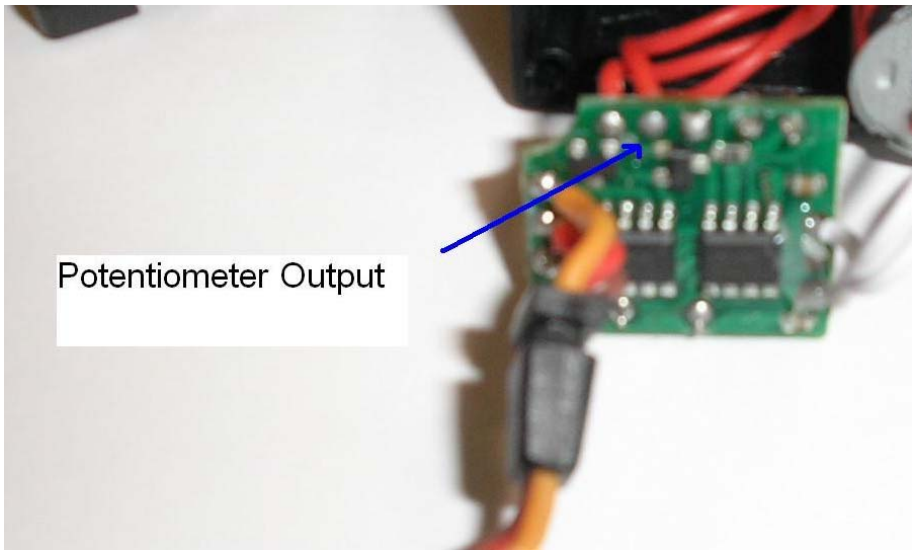


Figure 20: Position Feedback Location

The actual location of the position feedback is on the small circuit board inside the servo. The potentiometer takes two inputs, power and ground, and has one output, position. Its output is located on the middle pin and its value ranges from 0 to 3.3V for a full 180 degrees rotation. To collect this feedback a line was soldered to the middle wire and then put through a buffer before being sent to the microprocessor's ADC. The position sense line is filtered using a 100 Hz low-pass filter. This was done to reduce the input noise at the sampling frequency. See figure 21 for the feedback circuit.

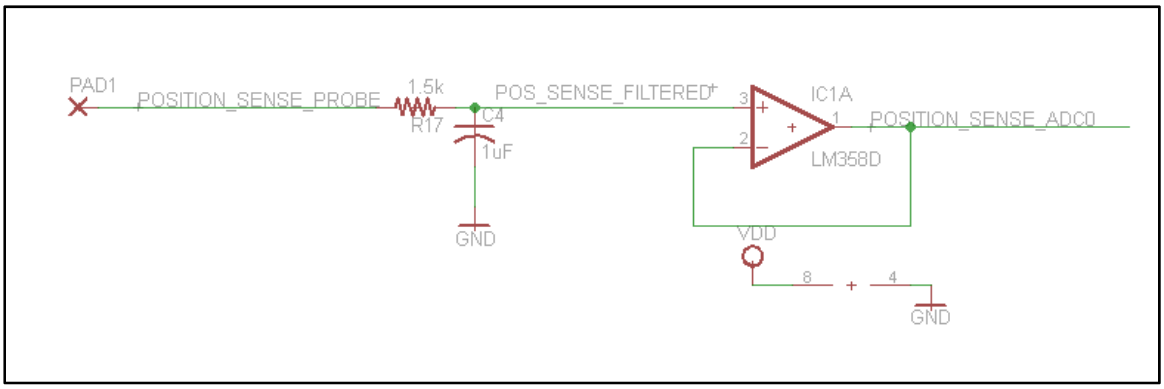


Figure 21: Position Feedback Circuit

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 servo and the resistor into a voltage

drop that can be measure. However, when the servo turns the other way the current can be seen as negative. The ADC used cannot handle negative voltages. In order to account for this, the current sense circuit consisted of the sense resistor output going into a non-inverting amplifier. This allows the current to be represented by a positive voltage always. 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 100Hz low-pass filter to reduce noise around the sampling frequency. The current sense circuit is demonstrated in figure 22.

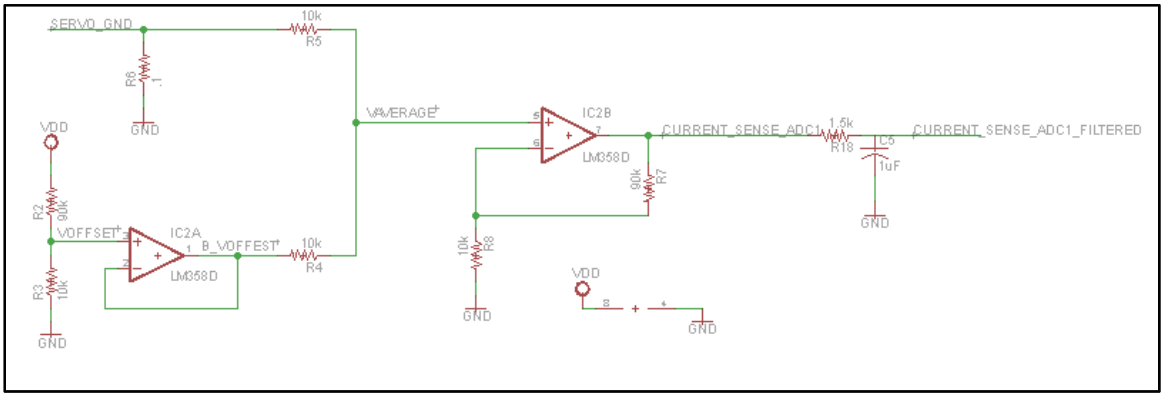


Figure 22: Current Sense Circuit

Address selection was implemented in a relatively simple way. The microprocessor has several general purpose input-output pins. A small dip switch was used to either set the pin high or to disconnect them from Vdd. When disconnected a pull-down resistor was used to ensure that the pins do not float and are driven to ground. See figure 23.

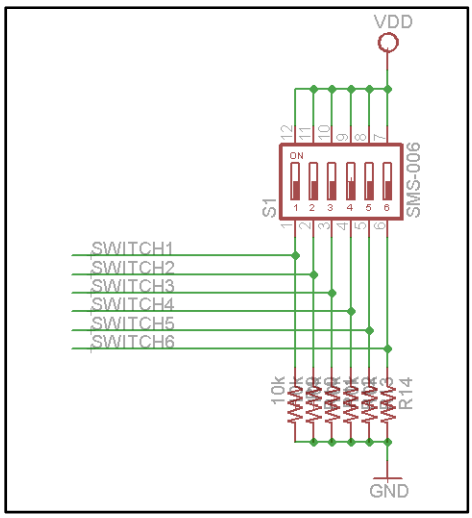


Figure 23: Address Select

Since the battery input was +7.4 Volts and all of the logic ran on 5 volts, a voltage regulator was needed. The regulator chosen was Linear Technologies' LT1763. The conversion was only from 7.4 to 5 volts so a linear regulator would work with little noise. The regulation circuit was built to the specifications that were given on the datasheet. However, an additional snubber capacitor was added in order to reduce the voltage spikes cause by the servo. The regulator and corresponding components are shown in figure 24.

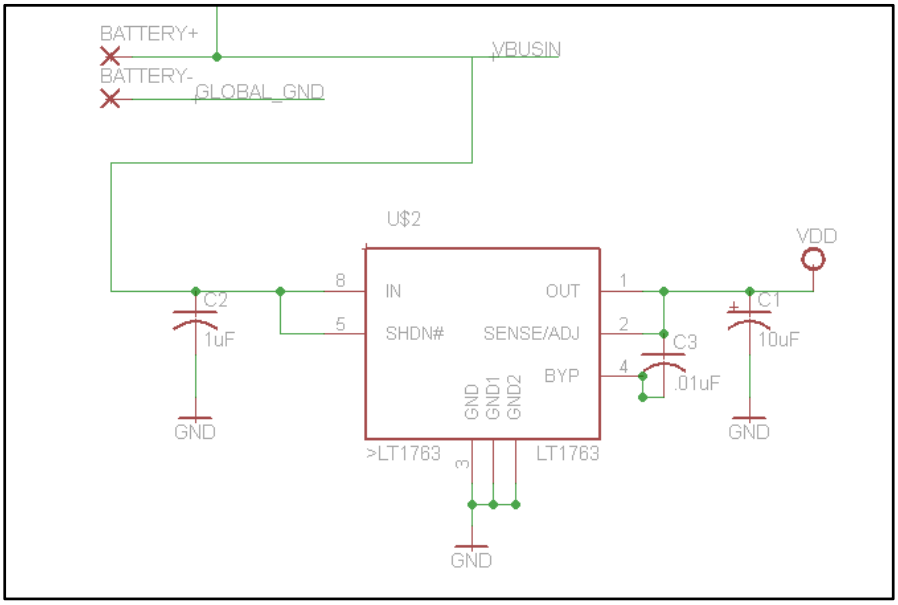


Figure 24: Voltage Regulator Circuit

To conserve battery life, a power on and off switch was designed into the boards. This was done using a MOSFET transistor. The transistor acted as a switch to disconnect

the logic and servo from the main power supply. The power on signal starts in the head and travels down the snake turning on each joint using only one switch in the beginning. See figure 25 for the schematic.

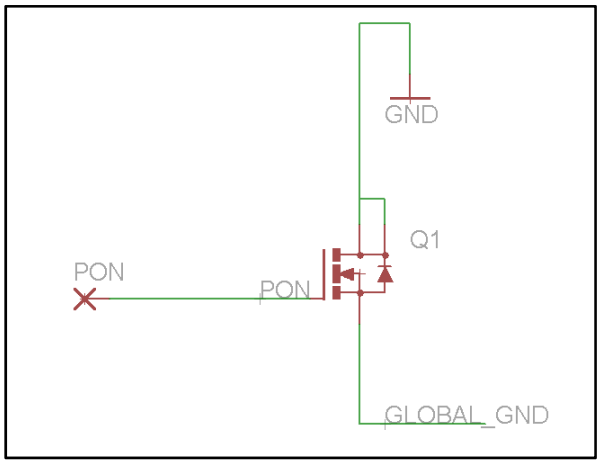


Figure 25: Power Off Signal

Once each component was designed and tested then the PCB board layout was designed. Basic design layout conventions were followed. Signal traces have a width of .012 inches while the power supplying the regulator and servo are .025 inches to account for the increase in current. As a rule of thumb the top-layer traces are horizontal, while the bottom layer traces are vertical. Also in addition to the circuits provided above, the PCB also breaks out the necessary lines for servo control, LED control, serial communication, and I2C communication. Figure 26 shows the PCB layout.

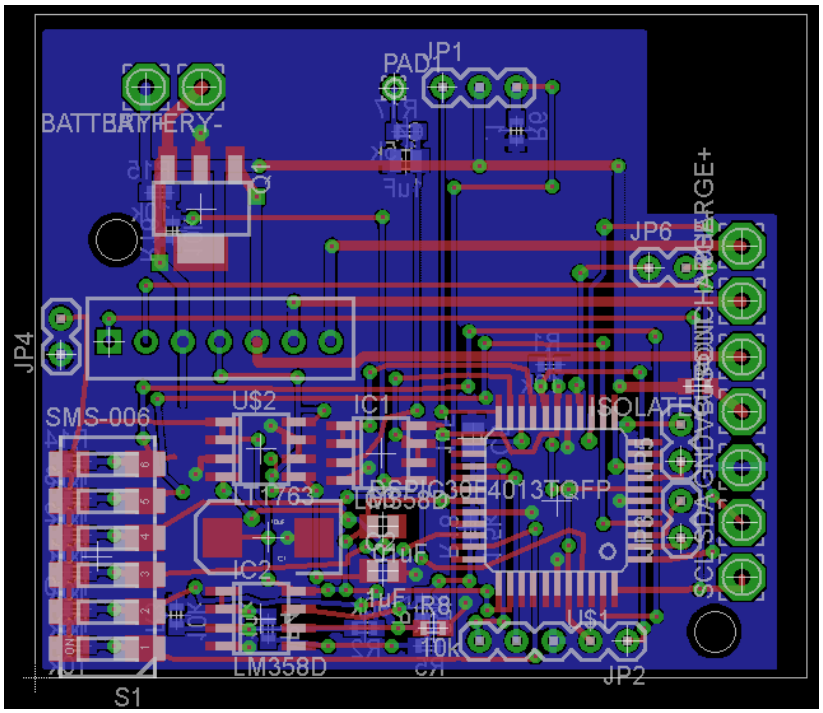


Figure 26: PCB Layout

The microprocessor for each link was chosen to be the dsPIC30F4013. The advanced capabilities and the ability to store data between power cycles make the dsPIC30F4013 the best choice for this situation. Each link will communicate using I2C. If serial was used then much time would be spent trying to create the functionality of I2C, while using serial. The master processor will communicate to the user through Bluetooth. The frequency hopping and the ease of use of Bluetooth make it the clear choice.

Vertebrae Design

The component final selection allowed for the frame or body design of each vertebra to be completed. This occurred in parallel with the product selection as both processes were closely tied together. In order to best mimic a biological snake a cylindrical shape was chosen. One of the main objectives in designing each vertebra was that they had to connect and disconnect from each other easily such that the snake would be scalable. After a few brainstorming sessions it was decided to have the driven gear be directly attached to the back of each vertebra and have a dead (non-rotating) shaft on the front of each vertebra. The shaft would go connect to a hitch on the front of each vertebra. The hitch would consist of an extrusion at the top and bottom of the cylinder. At

the back of each vertebra there would be a bearing block containing a bushing. This bearing block would hold the shaft and provide much of the strength between segments. In addition to the bearing block the driven gear would be attached to the rear of the vertebrae. This gear would have a bushing at its center point and provide additional support for the connection between the segments. The front of each vertebra would then have a window that would allow the driven gear of the next vertebra enter into the vertebra and mesh with the pinion gear on the servo. The initial concept for this idea can be found in figure 27.

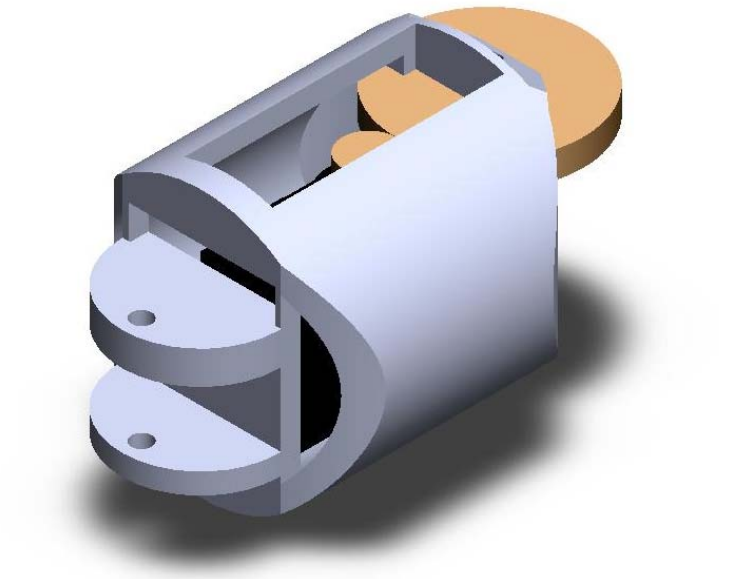


Figure 27: Initial Vertebra Design Concept

The initial design was very rudimentary but played a large role in acting as a starting point from which to develop the final design. The next design iteration can be seen in figure 28. As shown the front hitches have been refined. A hole has been added to allow wires to pass between segments and initial compartments for the battery and circuit board have been laid out. This particular picture shows the full gear attached to where the dead shaft will be located, and not attached at it the rear. Notice that at the rear of the vertebra a window to allow clearance for the driven gear is present, this was eliminated the next design iteration.

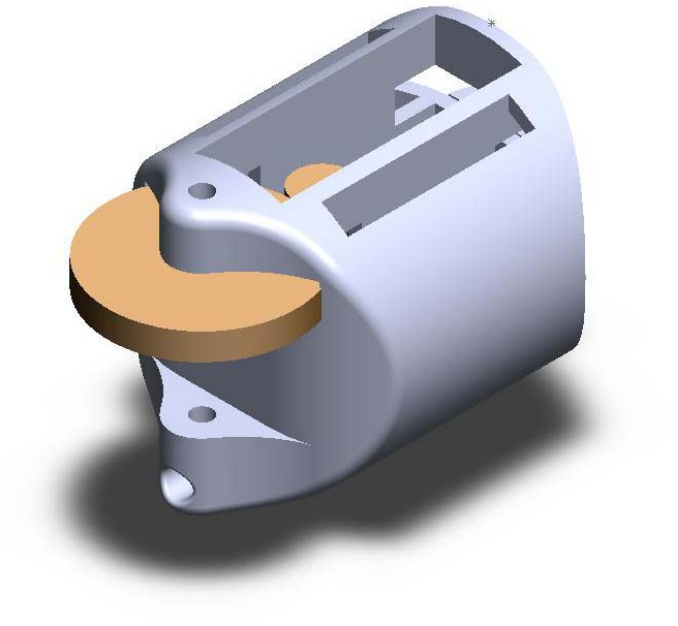


Figure 28: Second Design Iteration

The third design iteration resulted in the final design. This can be seen in figure 29, 30, and 31. This design incorporates many features critical to the assembly and performance of the snake. The largest of which is that the gear has been modified such that only the profile of the gear that is necessary is present. It attaches using two flat head 100 deg 8-32 machine screws. These are inserted from the bottom of the vertebrae through a countersunk shelf, through the gear, and into a substantial extrusion. While this design does put the screws in shear, it does so only partially. Because the gear will be seeing a moment about its point of rotation at the center hole, it will want to rotate. Therefore when the gear tries to turn it will be pushing into the vertebrae on one side, and pulling on the other. The tolerances are such that all of the pushing force will be transmitted directly into the body of the vertebrae, and pulling will be absorbed by the frictional forces from the two shelves and the screw in shear. Other features include additional mounting holes on each of the sides of the snake that allow attachments to be added at a later date. There is a battery compartment accessible from the top of the snake. The dead shaft is a press fit through both hitches. Shown in the side view of figure 31 there are clearance holes to allow for the assembly of the circuit board and a window to allow for a screwdriver to be used to loosen the screw terminals to connect the pigtail

wire that runs between each segment. An exploded view of a vertebra can be found in figure 33.

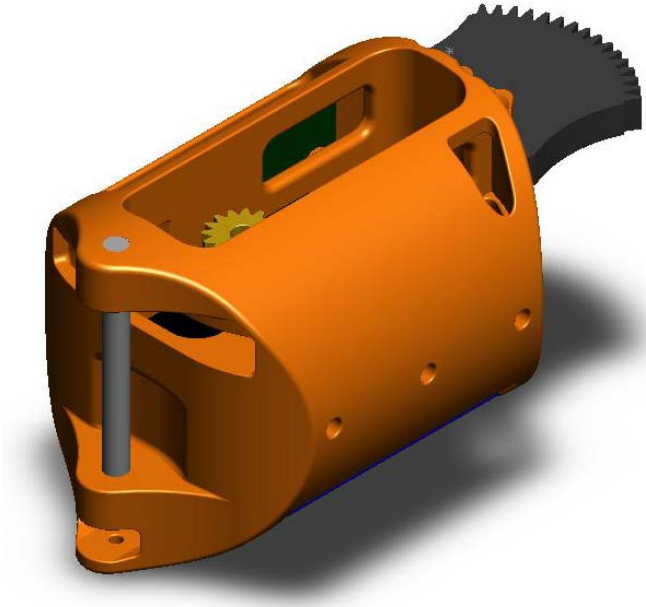


Figure 29: Final Design Front Isometric

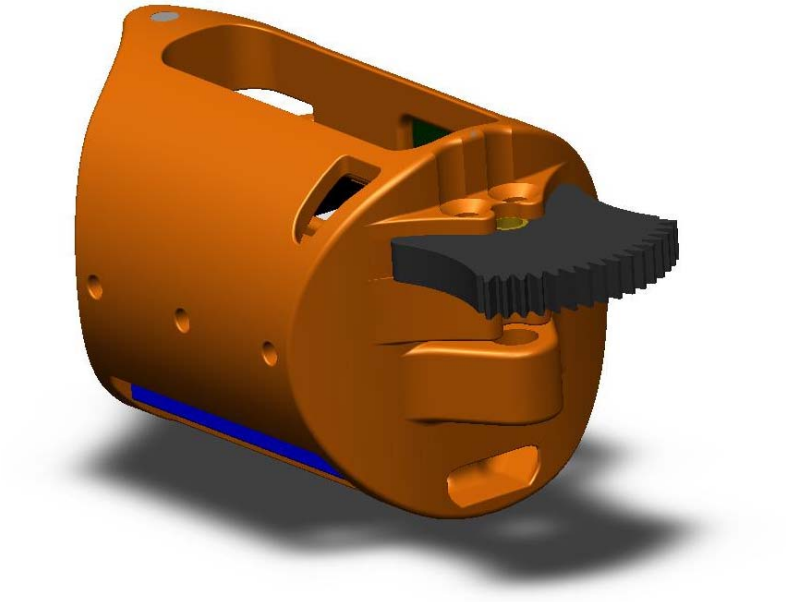


Figure 30: Final Design Back Isometric

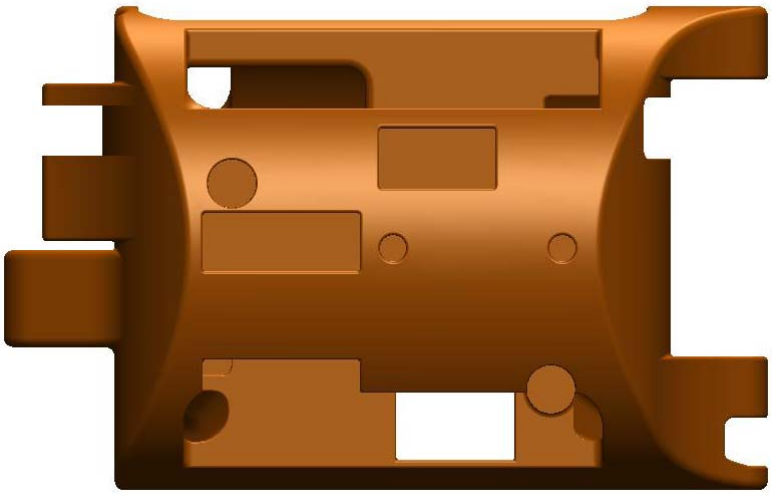


Figure 31: Final Design Side

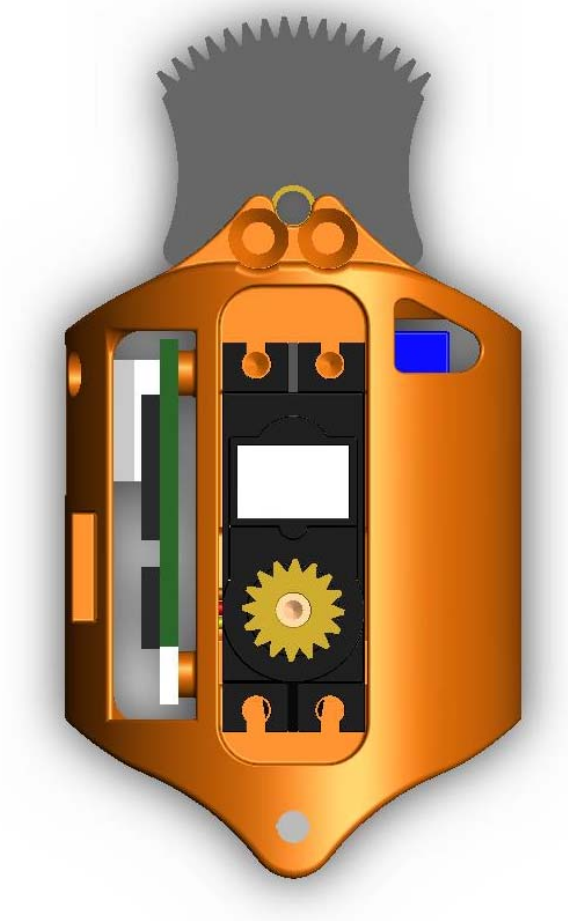


Figure 32: Final Design Bottom

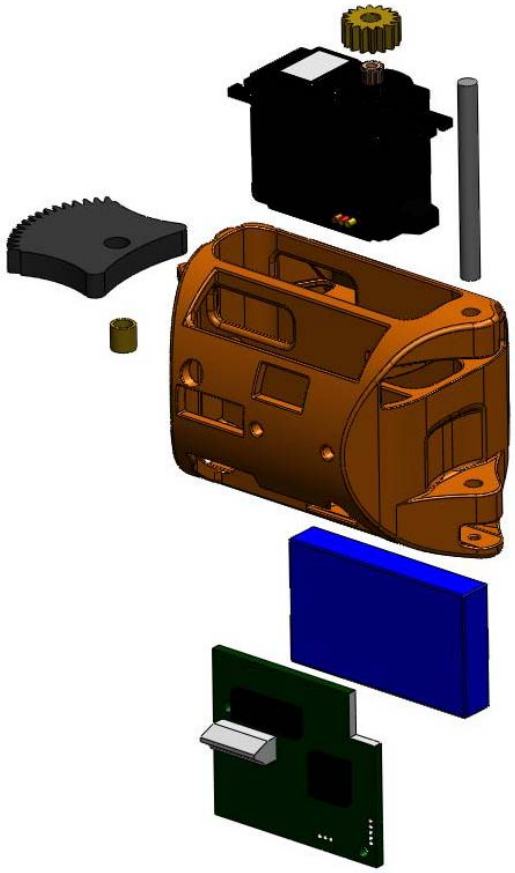


Figure 33: Vertebrae Assembly Exploded View

Head Design

The head of the snake involved a number of unique challenges. The head had to be shaped similar to a shovel-nosed snake yet contain the accelerometer, Bluetooth modem, and controller, on/off switch, and have an easy way to gain access to the internals. In order to meet all of these requirements sacrifices had to be made in the overall shape of the head. A snake's head comes to a point rather quickly, but this severely limits the internal space available. Therefore the head was CADed to allow for easy adjustments to the basic dimensions to have the most efficient use of space. The head consisted of two main portions, the bottom base and a lid. A majority of the head would not be covered with skin and therefore needed to remain sealed. In order to accomplish this, the lid was designed so that it overlapped with a lip on the head base. This lip also served the purpose of increasing the strength of the head. Four screws were

used to attach the lid to the base, two in the front and two in the rear. This low screw count allows for fast access to the internals of the head.

The head required a battery power source for the electronics. Because there wasn't a servo in the head a smaller battery could be used. The battery chosen was a smaller version of the vertebrae battery, the ThunderPower 250mAH LiPoly Double Cell 2S 7.4 V Prolite V2 Series. A small compartment was designed in the rear bottom portion of the head to hold the battery. A cover was designed to be placed over the battery and secured with two 4-40 screws. On this cover was a threaded boss onto which the master controller attached. The accelerometer board did not have any mounting points so a unique solution had to be developed. A slotted bracket was designed in the front bottom of the head that the board slid into, providing an easy assembly method and secure mounting. Mounting the Bluetooth modem was accomplished using the same technique, but in the top rear of the head. Two brackets and holes in the lid allow for the mounting of LEDs to serve as status lights so that the current state of the snake is obvious. The attachment method for the gear at the rear of the head is the same as all of the other segments. A feature added for future work is a main power input placed in the bottom of the head. Future projects may add the capability for the snake to charge itself, and this power jack will be able to provide power to the entire snake. An exploded view of the head can be seen in figure 34.

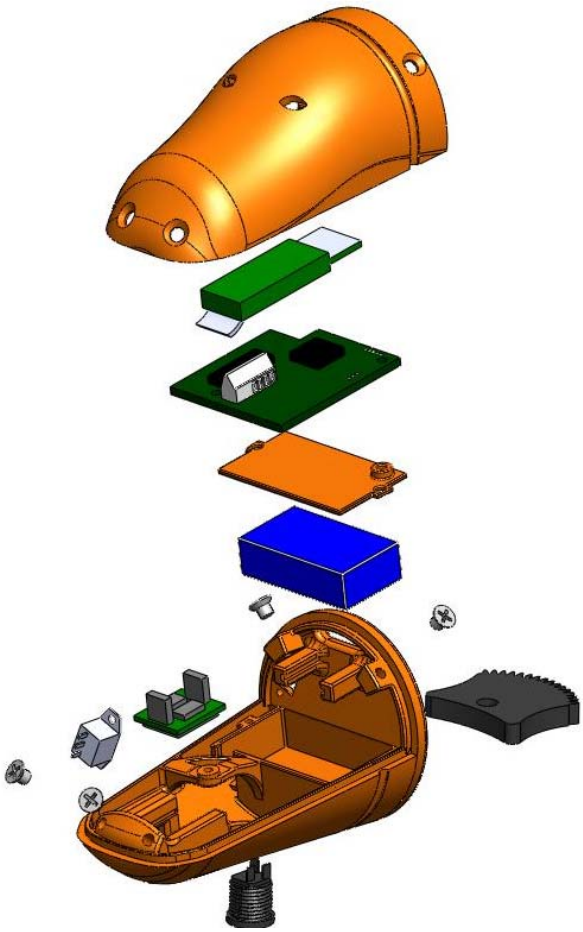


Figure 34: Head Assembly Exploded View

Skin

The skin was one of the more unique challenges of this project. When a snake forms a curve from a straight line the inner circumference becomes smaller than the outer circumference. However when the snake then curves the opposite way that previously smaller circumference becomes longer. This means if a standard type of material is used such as cotton cloth then the skin will either bunch up or be pulled to tightly and restrict the snake's movement. To avoid this the snake's skin needs to be flexible and self correcting to this problem. After an extensive search expandable braided sleeving was chosen as the ideal solution. This is a woven sleeve comparable to the Chinese finger-trap toy, however it is designed for protecting wires. It is unique in that the strands that make up the braid weave in a helical pattern. This in combination with the ability for the sleeve to move relative to itself means that it is self correcting when the snake curves.

When one side gets shorter than the opposite the extra material simply slides to the opposite side. This sleeve acts as a support to bridge the gaps between the segments but does not have a fine enough weave to keep out the granular media and therefore seal the snake. To seal the snake a spandex sleeve was used. The selected fabric was a 6 Oz 80% nylon 20% spandex 4-way stretch fabric. This was purchased and then sewn together into a tube the appropriate length. The seam was then glued with fabric glue in order to increase its strength and longevity. This skin composite skin combination can be seen in figure 35.

Figure 35: Composite Snake Skin

Software

In Appendix F the MATLAB code used to control the snake is displayed. One of the requirements for this research platform was to be able to develop an arbitrary traveling waveform and be able to perform that motion using the snake. In order to do this, the MATLAB symbolic toolbox was used. However the symbolic toolbox is only supported for 32-bit MATLAB.

Using this toolbox a program was created where the user can input the length of each link, the waveform equation, and the desired resolution. When MATLAB is run it displays the waveform generated along with the calculated snake joint angles. These

angles are then used to program the snake to perform the desired motion. MATLAB's graphics output is displayed in figure 36.

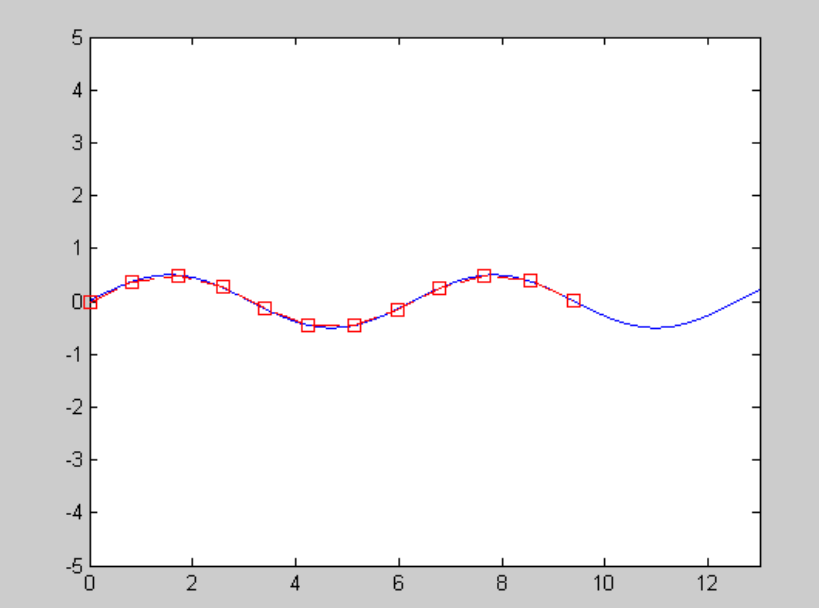


Figure 36: MATLAB Snake Waveform

The method of waveform following used is this project is simple. Since the scope of our projects was to allow for waveform following by the snake we needed a simple method for following it. While is method is quick and easy in implementation there are several drawbacks to using such a system. The first joint is placed at the beginning of the waveform. From that joint a circle with the radius equal to the link length is calculated. The intersection of the right side of that circle and the wave is the location of the second joint. This is repeated for each joint until there are no more links.

One of the major drawbacks from using this method is that this model does not take into account the non-linearity of the snake joints. As a result, certain waveform patterns cause it to create large amounts of error along with discontinuities of speed as each link goes around the apex of the wave. Figure 37 shows once such waveform.

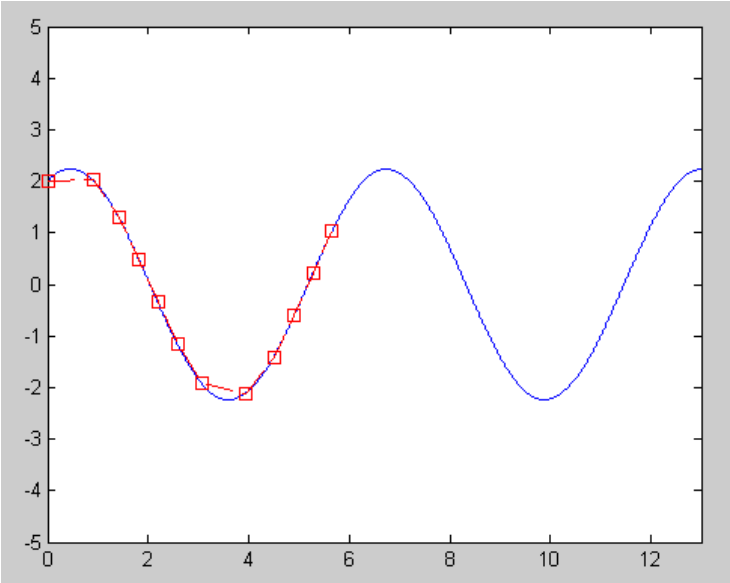


Figure 37: Poor Waveform

The error of a particular fit can be described as the integral of the square of difference between the waveform being followed and the wave created by the joint angles and link lengths of the snake. Since the scope of the project is the design and construction of the snake, more analysis of the waveform following and its implications are left for future work.

Testing Bed

In order to perform the desired experimental studies a test bed was necessary. A 3x1.2x.6 meter “tub” was built using plywood and 2x4s as shown in figure 38. This bed was then filled with 450 kg of granular media (polycrystal styrene). These plastic pellets are small cylinders 3 mm in diameter and roughly 1-4mm in length. They have density of 1 g/cm^3 and were kindly provided by Ineos Nova LLC. The size and makeup of this tub should allow for extensive testing to be performed for both straight line locomotion and turning.

Figure 38: Testing Tub

Validation

Finite Element Analysis

To ensure that the snake would not break during normal use a finite element analysis was performed on the vertebrae unibody. This validated that the design would indeed be strong enough for handling and normal operation. The primary analysis performed was to ensure two things. That the end segment could be held with the rest of the snake hanging orientated vertical and that the reaction forces between the segments could not break the snake in half. The analysis was performed using Cosmos Works. Restraints were set at the rear of the vertebra representing where the screws attach to the vertebrae and at the pillow block where the shaft goes. Linear forces were then placed on the two hitches. These were in line with the length of the vertebrae and their combined force was set to 68.75 N. The force exerted by the weight of the snake is 30.5 N. The reaction force between the two gears was calculated as follows:

$W_r = \frac{2*T_p}{d_p} * \tan \emptyset$

Where,

$T_p = Pinion\ Torque$

$d_p = Pitch\ Diameter$

$\emptyset = Pressure\ Angle$

$W_r = \frac{2*1.2Nm}{.0127m} * \tan 20 = 68.78\ N$

Therefore only the reaction force needs to be analyzed. For material properties the lowest documented yield strength for abs plastic was used and then multiplied by .6 to account for the degraded strength due to the printing process. The software then analyzed the safety factor as a function of location and plotted it as shown in figure 39. The lowest safety factor was calculated to be 3.1 and is shown by the red points.

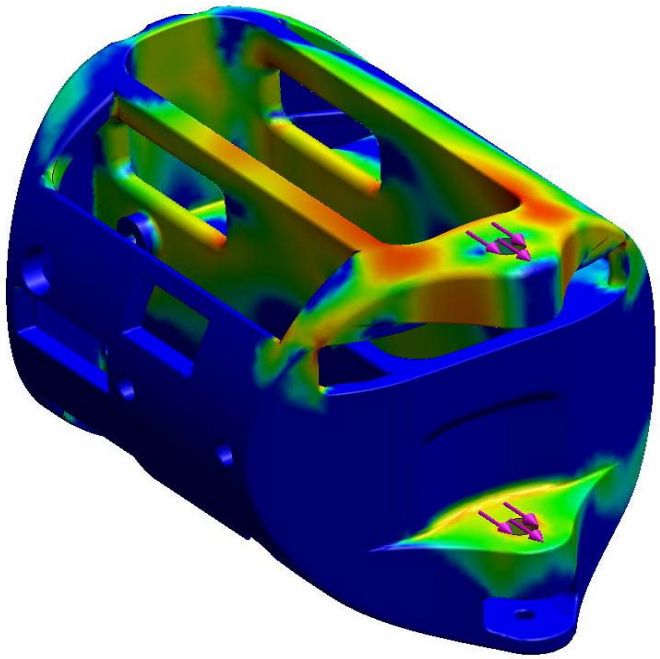


Figure 39: FEA Analysis Safety Factor Plot

Movement Testing

In order to validate that the snake could indeed move through the granular media as required tests were performed throughout the course of the project. Initially two segments were built in order to validate that all parts would fit as expected. These were then placed in a zip block bag to act as a skin and the joints were actuated using standard hobby radio gear. The two joints were able to move without issue at a depth of .5 meters, the deepest checked. As the snake neared completion testing was done with 6 segments at a depth up to 15 cm with successful results.

Conclusion

The project succeeded in producing a self-contained scalable robotic snake that is able to move inside granular media. The snake consisted of eleven powered joints, in addition to the head and tail segments. Testing was limited but functionality was shown at depths of at least .15 meters. A user is able to use the custom MATLAB program to create a lookup table and then wirelessly transfer that table to the snake via HyperTerminal. Then using a series of commands in HyperTerminal the snake can be commanded to execute the desired traveling waveform step by step or continuously. This project incorporated a number of challenges with the primary one being limited availability of time. With minimal additional work the snake will be fully capable of acting as a research platform for future studies of swimming in granular media. Images of the final snake robot can be seen in figures 40, 41, 42 and 43.

Figure 40: Completed Snake

Figure 41: Completed Snake w/ Skin

Figure 42: Five Identical Links Connected

Figure 43: The Snake Robot in the Testing Bed and a Real Snake in the Desert

Recommendations

Recommendations for future work on the robotic snake consist of waveform evaluation, and both a hardware and software updates. The scope of this project was to create a platform for research. In its current state, the robot could be used for research. However, several improvements to the snake can increase both its effectiveness and usability.

For hardware, an onboard charging circuit will greatly improve usability and will allow for a much smoother transition between testing and charging. There is space on the current board design for such a circuit to be implemented. In order to allow such a design, there must also be another layer of isolation between the battery and circuit board that will prevent that charging from affecting the main logic of each controller.

In addition to the charging circuit, we recommend that time be taken to look at the interface between the battery and the circuit board and look into ways to both increase the stability of the power line while under high current draw from the servo, and to increase the protection for the board's components. With these additions the snake will become much more robust.

For software, a well defined interface between the user and the snake should be created. Currently the software is such that it can be used for testing, but changing the waveform requires wired manual transfer of the individual link's lookup tables. A file parser should be writing so that the user may download the position file through a program like HyperTerminal.

Although that file parser will increase usability one of the greatest changes would be to give the user access to the onboard EEPROM. This will allow for the user to collect the data from the experiments along with storing different waveforms into the snake even between power cycles.

With these changes the snake will be both far more robust and user friendly. Implementation of these recommendations will greatly decrease the time between tests along with increasing the overall effectiveness of the snake.

Works Cited

Bauchot, R. (2006). *Snakes: A Natural History.* New York, New York: Sterling Publishing Co., INC.

Bzdega, M., Robertson, B. D., Soller, R., Huber, G., & Koehler, S. A. (June 24, 2008). *Swimming in granular media.*

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

Flickr. (n.d.). Retrieved 04 30, 2009, from http://farm2.static.flickr.com/1410/1069642779_c4d52d0aa2_o.jpg

Gray, J. (1946). The Mechanism of Locomotion in Snakes. *Journal of Experimental Biology , 23* (2), 101-120.

Gray, J., & Lissmann, H. W. (1949). *The Kinetics of locomotion of the Grass-Snake.*

Grimm, Todd, T.A. Grimm & Associates. (2003). Fused Deposition Modelling: A technology Evaluation. *Time Compression , II* (2).

Howstuffworks. (n.d.). Retrieved 04 30, 2009, from http://static.howstuffworks.com/gif/snake-motion.gif

Hu, D., Nirody, J., Scott, T., & Shelley, M. (April 5, 2008). *Snakes on a plane: the mechanics of slithering.*

Koehler, S. (March 1, 2007). *Research Proposal: Swimming In Granular Media.*

Saito, M., Fukaya, M., & Iwasaki, T. (N.A.). *Modeling, Analysis, and Synthesis of Serpentine locomotion with a Multilink Robotic Snake.*

WorldBook. (n.d.). Retrieved 04 30, 2009, from http://www.worldbook.com/wb/images/content_spotlight/reptiles/internalorgans.gif

Appendices

Appendix A: Torque Requirement Calculations

$Length_{pvc} := 2.75in = 0.07\,m$

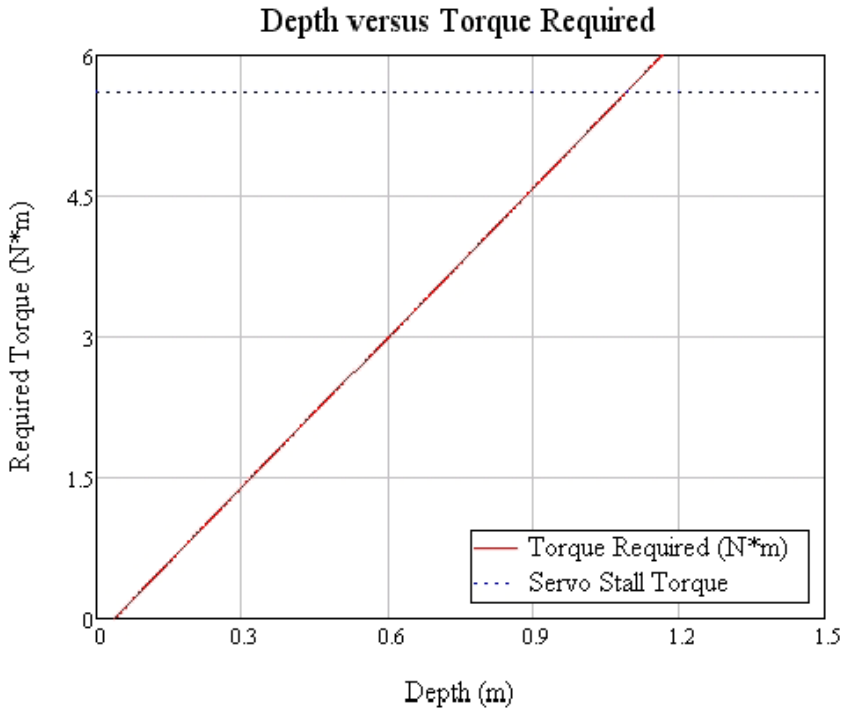
$Length_{bar} := 4in = 0.102\,m$

$Pivot\_Length_{segment} := 1in = 0.025\,m$

$Servo_{stalltorque} := 138oz \cdot g \cdot in \cdot \left(\frac{7.4}{6}\right) \cdot 3.5 \cdot \left(\frac{1}{.75}\right) = 5.609 \cdot N \cdot m$

$Torque(z) := \left[\frac{(13.976 \cdot z - .5198) \cdot N \cdot m}{\left[\left(Length_{bar} - Length_{pvc}\right) + \left(.5 \cdot Length_{pvc}\right)\right]}\right] \cdot Pivot\_Length_{segment}$

$Torque(.191) = 0.819 \cdot N \cdot m$



$Dia := 2.5in = 0.064\,m$

$Radius_{pulley} := .55in = 0.014\,m$

$Density_{plastic} := 1\,\frac{gm}{cm^3}$

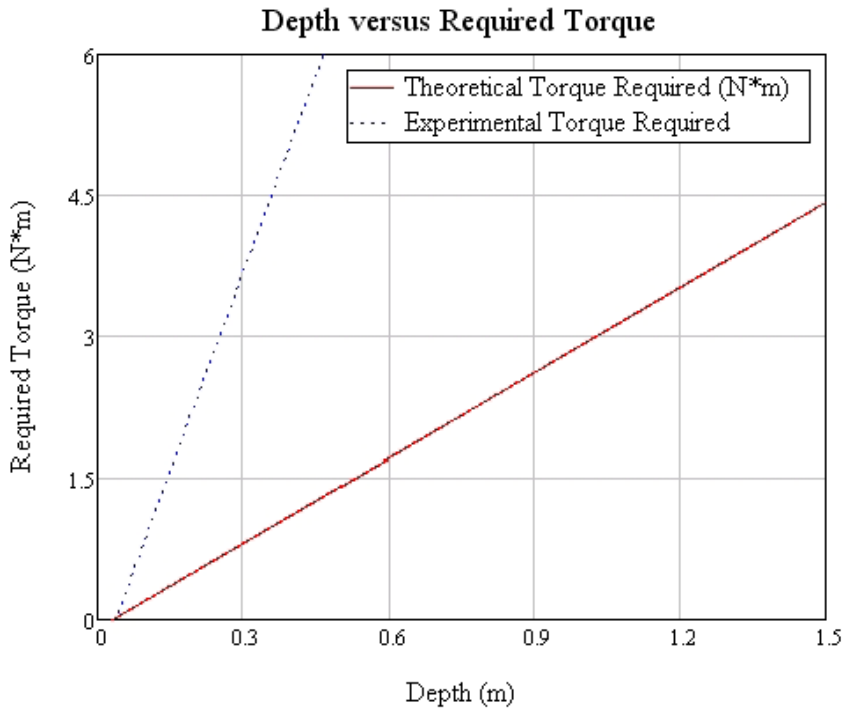
$x_1 := 1in = 0.025\,m \qquad z_2(z) := z$

$x_2 := 4in = 0.102\,m \qquad z_1(z) := z_2(z) - Dia$

$$Torque_{th}(z) := \frac{Density_{plastic} \cdot g \cdot \left(x_2^{\,2} - x_1^{\,2}\right)}{4} \cdot \left(z_2(z \cdot m)^2 - z_1(z \cdot m)^2\right)$$

$Torque_{ex}(z) := (13.976 \cdot z - .5198) \cdot N \cdot m$

$$\frac{Torque_{ex}(.2)}{Torque_{th}(.2)} = 4.488$$



Appendix B: Gear Requirement Calculations

General Parameters

Pressure Angle

$\phi := 20deg$

Torque at Pinion

$T_p := 138 \cdot \left(\frac{7.4}{6}\right) oz \cdot in$

Width

$F := .25in$

Application Factor

$K_a := 1.25$

Load Distribution Factor

$K_m := 1.2$

Bending Geometry Factor

$J_p := .24$

$J_g := .28$

Dynamic Factor

$K_v := .99$

Size Factor

$K_s := 1$

Rim Thickness Factor

$K_B := 1$

Idler Factor

$K_I := 1$

32 diametrical pitch

Pinion Pitch Dia $D_{pp32} := .5in$

Gear Pitch Dia $D_{pg32} := 1.875in$

Dia Pitch $P_{d32} := 32in$

48 diametrical pitch

Pinion Pitch Dia $D_{pp} := .4167in$

Gear Pitch Dia $D_{pg} := 1.5in$

Dia Pitch $P_d := 48in$

Calculations:

Addendum Gear

$$a_p(P_d) := \frac{1in^2}{P_d}$$

Pitch radii

$$r_p(D_p) := .5 \cdot D_p$$

Center Distance

$$C(D_{pp}, D_{pg}) := r_p(D_{pp}) + r_p(D_{pg})$$

Length of Action

$$Z(\phi, D_{pp}, D_{pg}, P_d) := \sqrt{\left(r_p(D_{pp}) + a_p(P_d)\right)^2 - \left(r_p(D_{pp})\cos(\phi)\right)^2} + \sqrt{\left(r_p(D_{pg}) + a_p(P_d)\right)^2 - \left(r_p(D_{pg})\cdot\cos(\phi)\right)^2} - C(D_{pp}, D_{pg})\cdot\sin(\phi)$$

Contact Ratio

$$m_p(D_{pp}, \phi, D_{pg}, P_d) := \frac{Z(\phi, D_{pp}, D_{pg}, P_d)}{\frac{3.14in^2}{P_d}\cdot\cos(\phi)}$$

Tangential Force

$$W_t(T_p, D_p) := \frac{T_p}{r_p(D_p)}$$

Radial Force

$$W_r(T_p, P_d, \phi) := W_t(T_p, P_d)\cdot\tan(\phi)$$

Total Force

$$W(T_p, P_d, \phi) := \frac{W_t(T_p, P_d)}{\cos(\phi)}$$

Bending Stress

$$\sigma_b(T_p, D_p, P_d, F, J_f, K_a, K_m, K_v, K_s, K_B, K_I) := \frac{W_t(T_p, D_p)\cdot\frac{P_d}{in}}{F\cdot J_f}\cdot\frac{K_a\cdot K_m}{K_v}\cdot K_s\cdot K_B\cdot K_I$$

48 Pitch

Contact Ratio: Should be between 1 and 2

$$m_p(D_{pp}, \phi, D_{pg}, P_d) = 1.685$$

Bending Stress

$$\sigma_b(T_p, D_{pp}, P_d, F, J_p, K_a, K_m, K_v, K_s, K_B, K_I) = 6.189 \times 10^4 \cdot \frac{lb}{in}$$

Safety Factor for Delrin

Fatique bending stress of Delrin at 1 million cycles 7200PSI

$$\frac{7200 \frac{lb}{in}}{\sigma_b(T_p, D_{pp}, P_d, F, J_p, K_a, K_m, K_v, K_s, K_B, K_I)} = 0.116$$

Safety Factor for Steel

Fatique strength of steel 30000PSI

$$\frac{30000 \frac{lb}{in}}{\sigma_b(T_p, D_{pp}, P_d, F, J_p, K_a, K_m, K_v, K_s, K_B, K_I)} = 0.485$$

Safety Factor for 2024- T6 Aluminum

Fatique strength of 2024 T6 Aluminum 18000PSI

$$\frac{18000 \frac{lb}{in}}{\sigma_b(T_p, D_{pp}, P_d, F, J_p, K_a, K_m, K_v, K_s, K_B, K_I)} = 0.291$$

32 Pitch

Contact Ratio: Should be between 1 and 2

$$m_p\left(D_{pp32}, \phi, D_{pg32}, P_d\right) = 1.722$$

Bending Stress

$$\sigma_b\left(T_p, D_{pg32}, P_{d32}, F, J_p, K_a, K_m, K_v, K_s, K_B, K_I\right) = 9.169 \times 10^3 \cdot \frac{lb}{in}$$

Safety Factor for Delrin

Fatique bending stress of Delrin at 1 million cycles 7200PSI

$$\frac{7200 \frac{lb}{in}}{\sigma_b\left(T_p, D_{pg32}, P_{d32}, F, J_p, K_a, K_m, K_v, K_s, K_B, K_I\right)} = 0.785$$

Safety Factor for Steel

Fatique strength of steel 30000PSI

$$\frac{30000 \frac{lb}{in}}{\sigma_b\left(T_p, D_{pg32}, P_{d32}, F, J_p, K_a, K_m, K_v, K_s, K_B, K_I\right)} = 3.272$$

Safety Factor for 2024 T6 Aluminum

Fatique strength of 2024 T6 Aluminum 18000PSI

$$\frac{18000 \frac{lb}{in}}{\sigma_b\left(T_p, D_{pg32}, P_{d32}, F, J_p, K_a, K_m, K_v, K_s, K_B, K_I\right)} = 1.963$$

Appendix C: Operator Instructions

Hardware Needed:

To wirelessly communicate with the robotic snake Bluetooth capability is necessary. This can be achieved either through a USP Bluetooth Dongle or built-in Bluetooth module.

To program the processors a PIC programmer is needed. Either the MPLAB ICD 2 or a PIC kit can be used.

Software Needed:

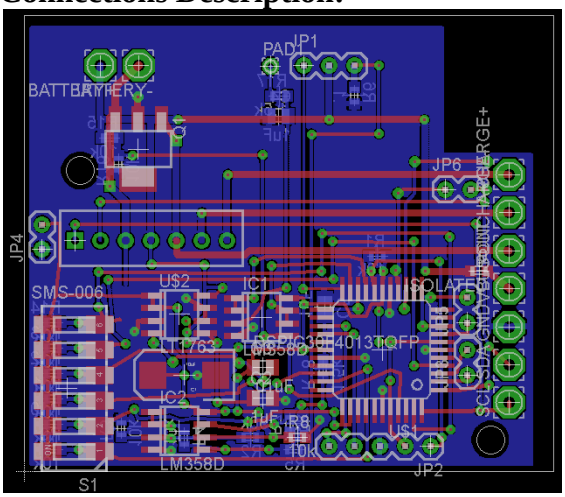
In order to program the processors MPLAB C30 is recommended. This can be acquired directly through the Microchip site and has a free student version for use.

To communicate via Bluetooth BlueSoleil 6.4.249.0 or higher is suggested. It can be acquired through http://www.bluesoleil.com/. It is free for 30 days, after that there is a one time fee of 30 dollars.

For user commands HyperTerminal 7.0 is suggested and is freely available online.

To generate waveform 32-bit MATLAB must be used. Of the remote servers offered at WPI only hutt.ece.wpi.edu is a 32-bit machine with 32-bit MATLAB.

Connections Description:



The following information assumes that the board is orientated such that the cut out is in the top right hand corner as shown above.

(From Left to Right, Top to Bottom)

Battery: Dual Through-Hole Horizontal top left corner. Pins are BATTERY+, BATTERY- .
The Servo Connecter: Only 3 Pin R/A header. Pins are Signal, Power, Ground

5v Power: Dual Through-Hole Vertical far left middle. Pins are 5v, Local GND

Terminal Block 7 Pos: 7 Through-Holes Horizontal left middle. Pins are SCL, SDA, BATTERY-, VBUS, PON, CHARGE-, CHARGE+

PIGTAIL: 7 Through-Holes Vertical Far Right Middle. Pins are CHARGE+, CHARGE-, PON, VBUS, BATTERY-, SDA, SCL

Serial connecter: 2 Pin vertical middle bottom right. Pins are U2RX, U2TX

Programming connector: 5 Pin R/A header Pins are PGC, PGD, GND, VDD, VPP

Basic Command Overview:
(replace X with number of choice)

PXX: Send current Joint to angle XX (from -26 to 26)

C: Run entire waveform table

W: Move to next step in waveform table

M: Communicate each link to move (current one does not)

AX: Get and output ADC value of ADC X (POS A0, Current A1, Voltage A2)

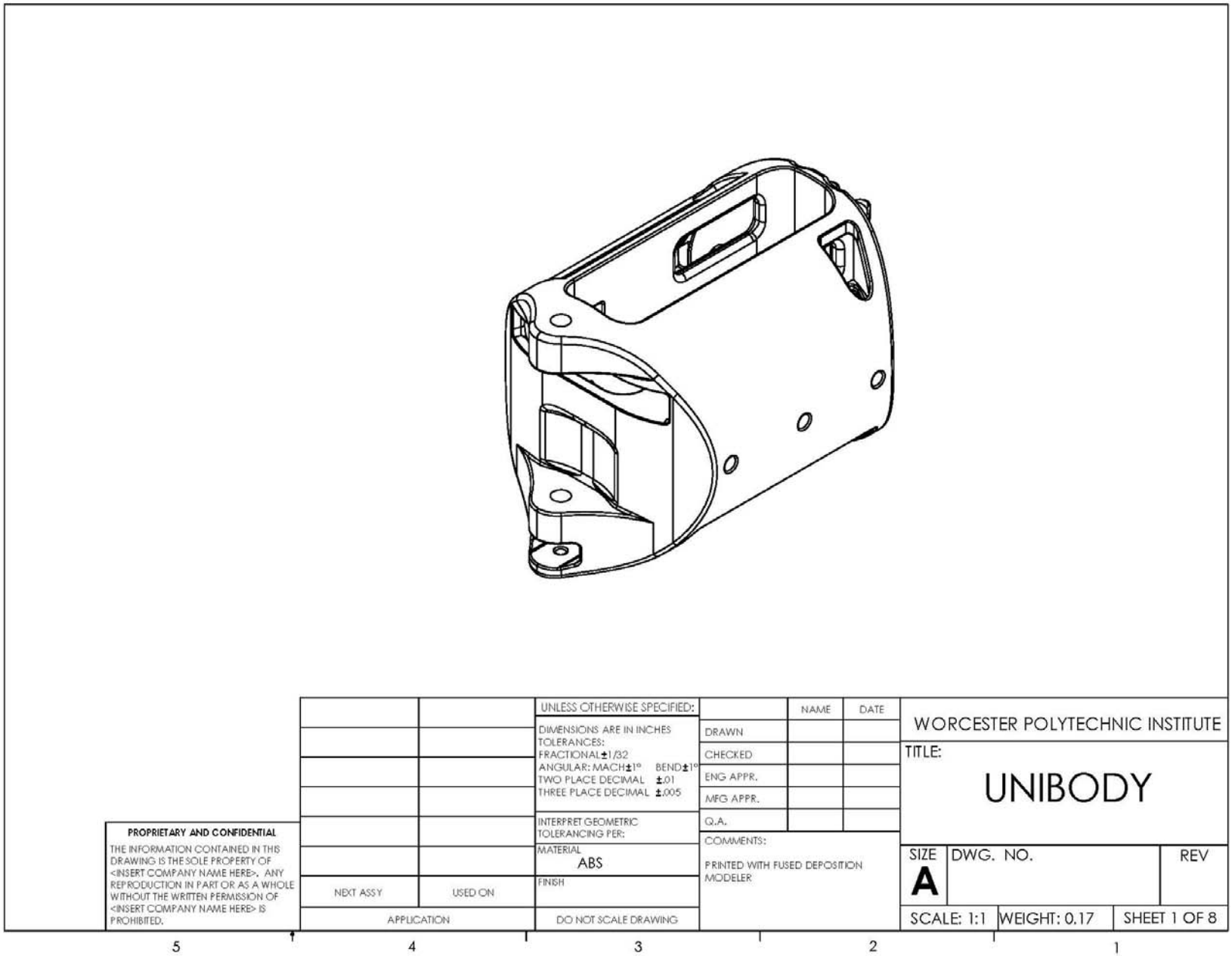
dsPIC Start Up:
To ensure the dsPIC is working it sends a U to the serial line when it is reset or connected.

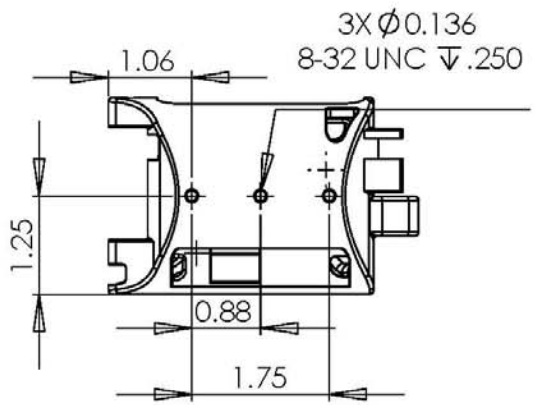
Standard Startup:
Connect batteries, Power On, Connect via HyperTerminal, Check connection (i.e. U or Echo of serial inputs), if connected begin.

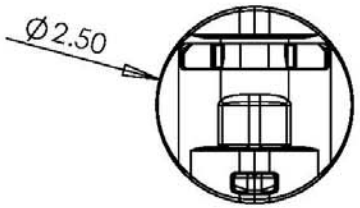
If that fails most common problems are Bluetooth connection (green light means connected , red means no), Low-Battery Ideal (7v+), and Power Switch.

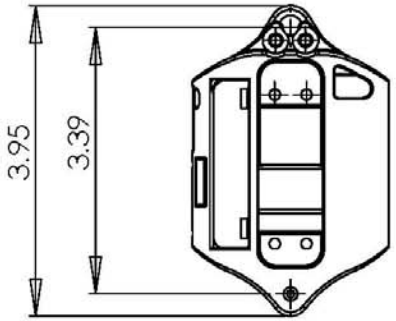
When in doubt Reset everything and check for serial response (‘U’). If that fails try to reprogram and connect.

Appendix D: Part Drawings









		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES 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			

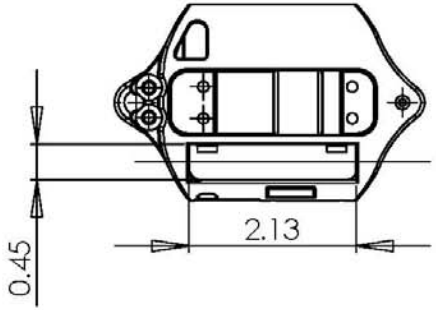
WORCESTER POLYTECHNIC INSTITUTE

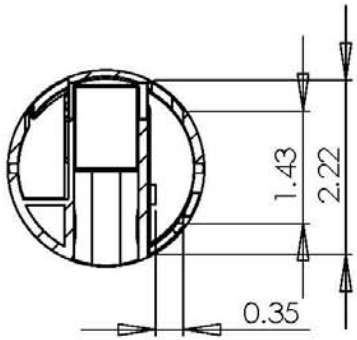
TITLE:

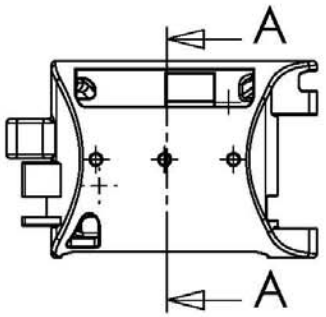
UNIBODY

SIZE	DWG. NO.	REV
A		
SCALE: 1:2	WEIGHT: 0.17	SHEET 2 OF 8

5 4 3 2 1







SECTION A-A

		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES 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

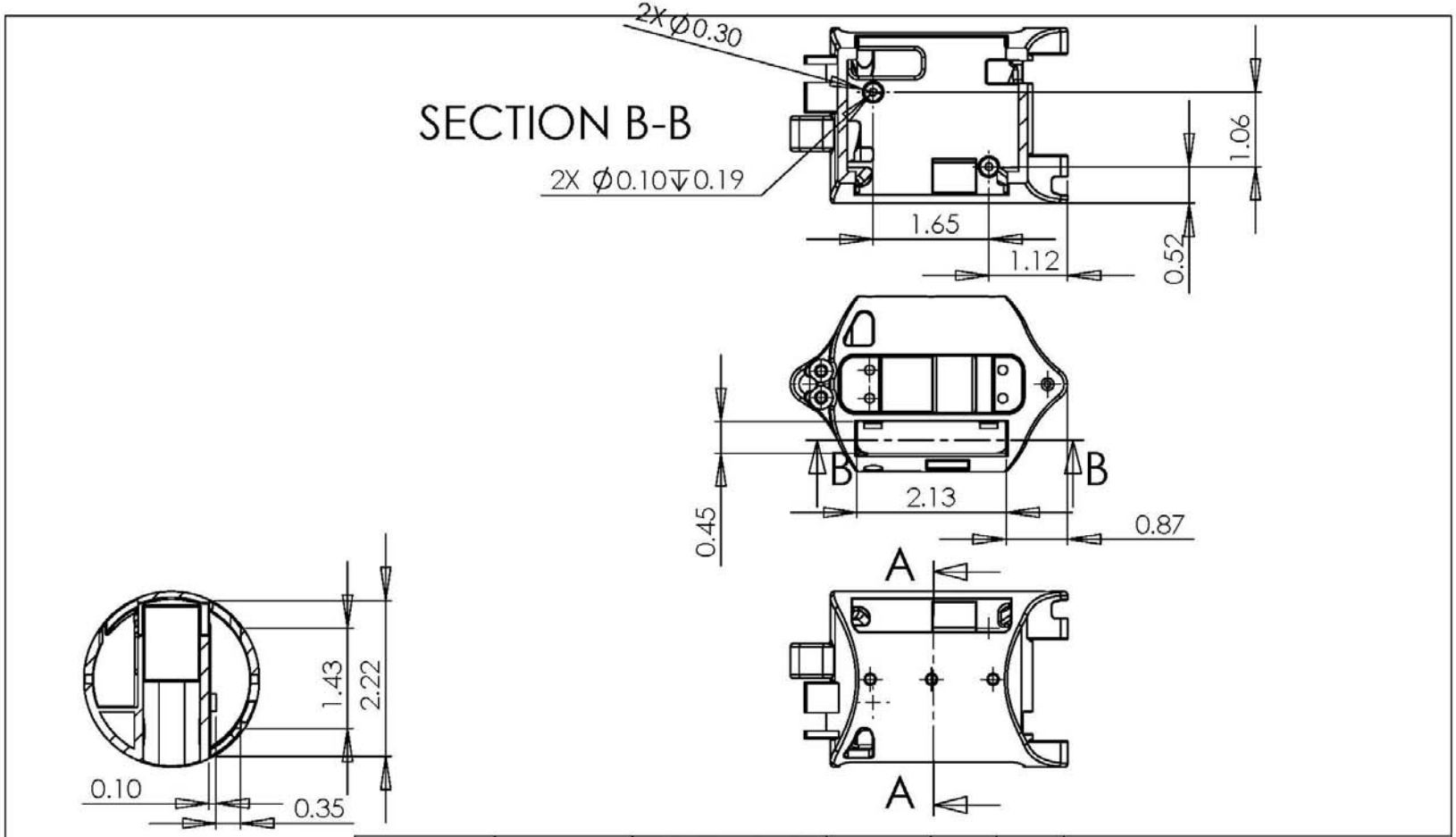
TITLE:

UNIBODY

SIZE	DWG. NO.	REV
A		

SCALE: 1:2	WEIGHT: 0.17	SHEET 3 OF 8

5 4 3 2 1



		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES 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:		
		MATERIAL ABS	PRINTED WITH FUSED DEPOSITION MODELER		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

WORCESTER POLYTECHNIC INSTITUTE

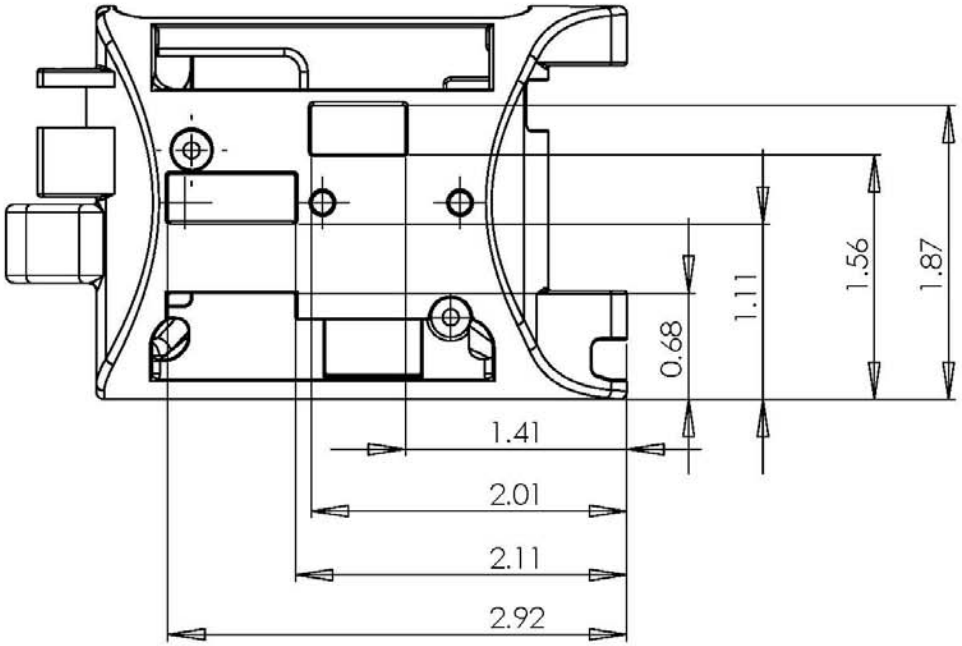
TITLE:

UNIBODY

SIZE	DWG. NO.	REV
A		

SCALE: 1:2	WEIGHT: 0.17	SHEET 4 OF 8

5 4 3 2 1



		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES 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 ABS			
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

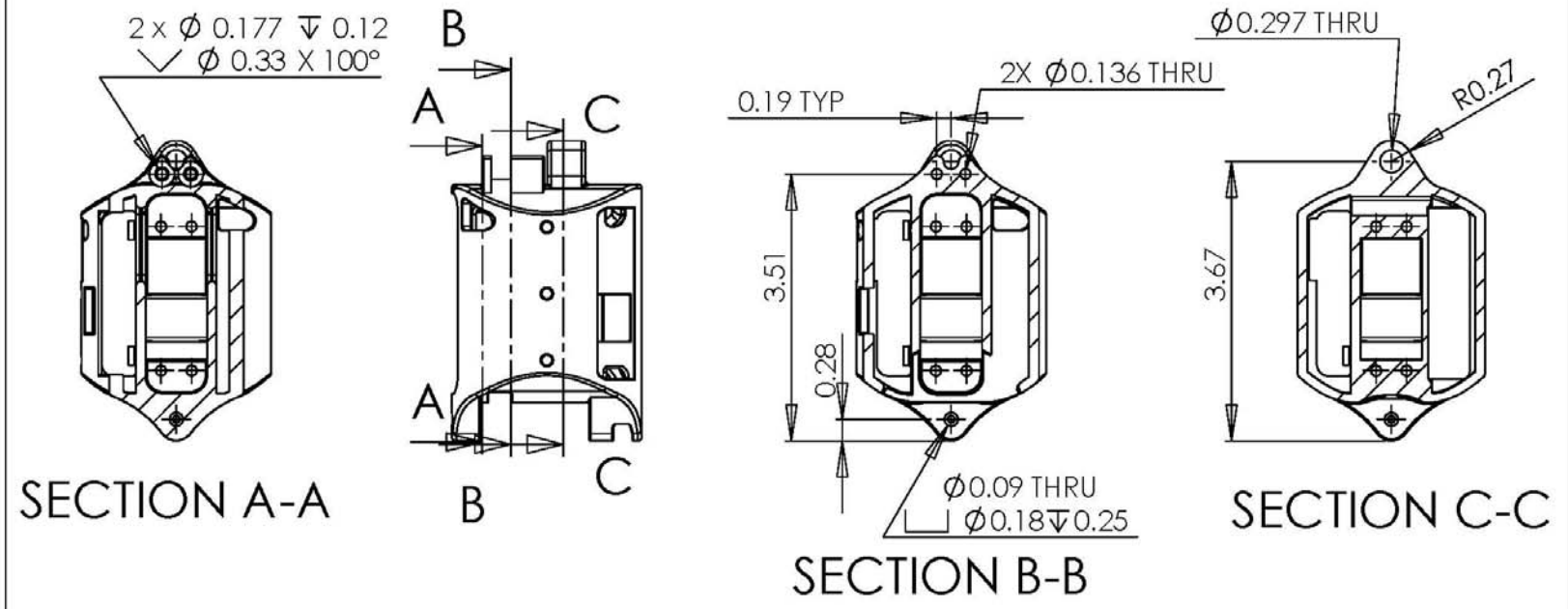
WORCESTER POLYTECHNIC INSTITUTE

TITLE:

UNIBODY

SIZE	DWG. NO.	REV
A		
SCALE: 1:1	WEIGHT: 0.17	SHEET 5 OF 8

5 4 3 2 1



		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES 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			

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.

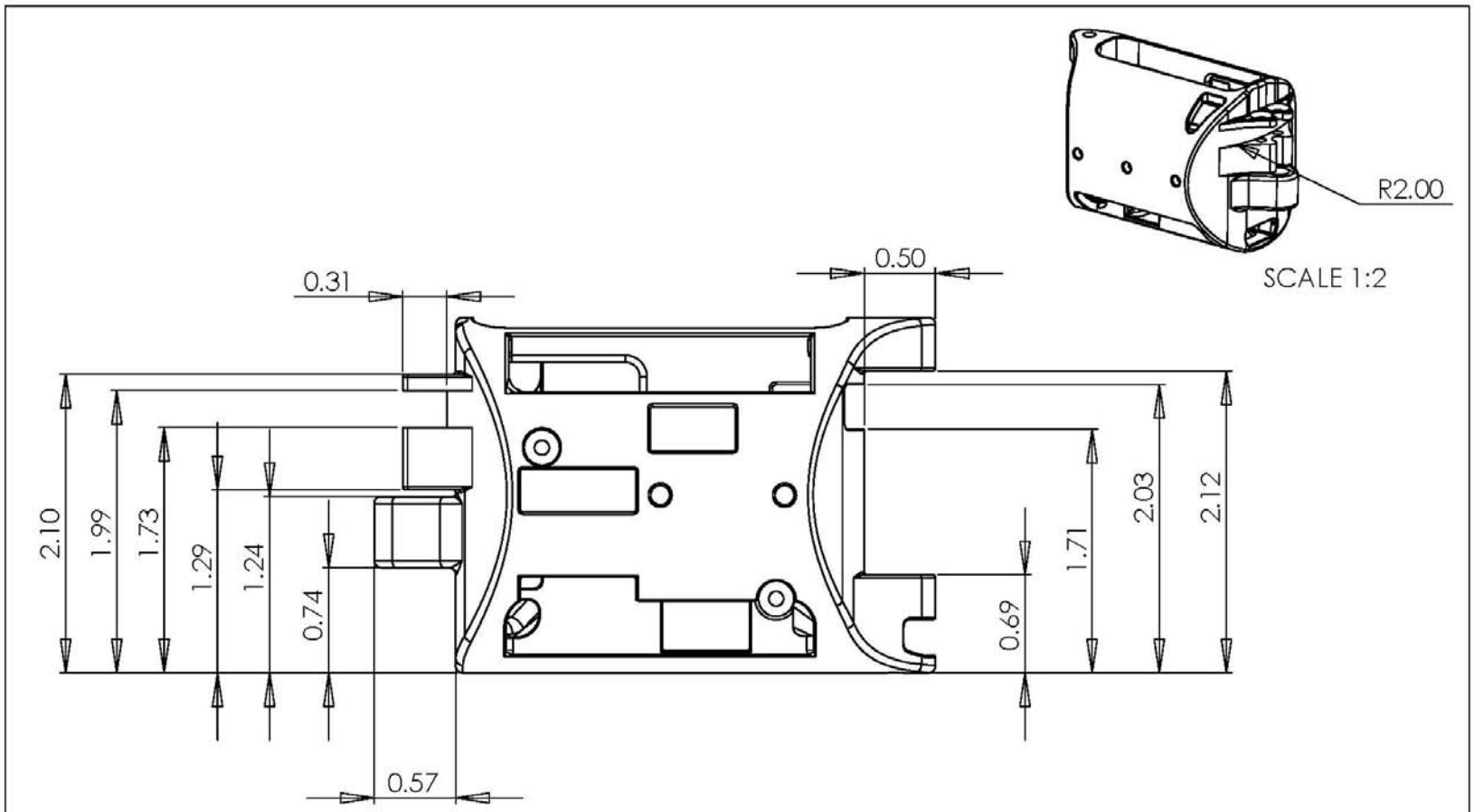
WORCESTER POLYTECHNIC INSTITUTE

TITLE:

UNIBODY

SIZE	DWG. NO.	REV
A		
SCALE: 1:2	WEIGHT: 0.17	SHEET 6 OF 8

5 4 3 2 1



		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES 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

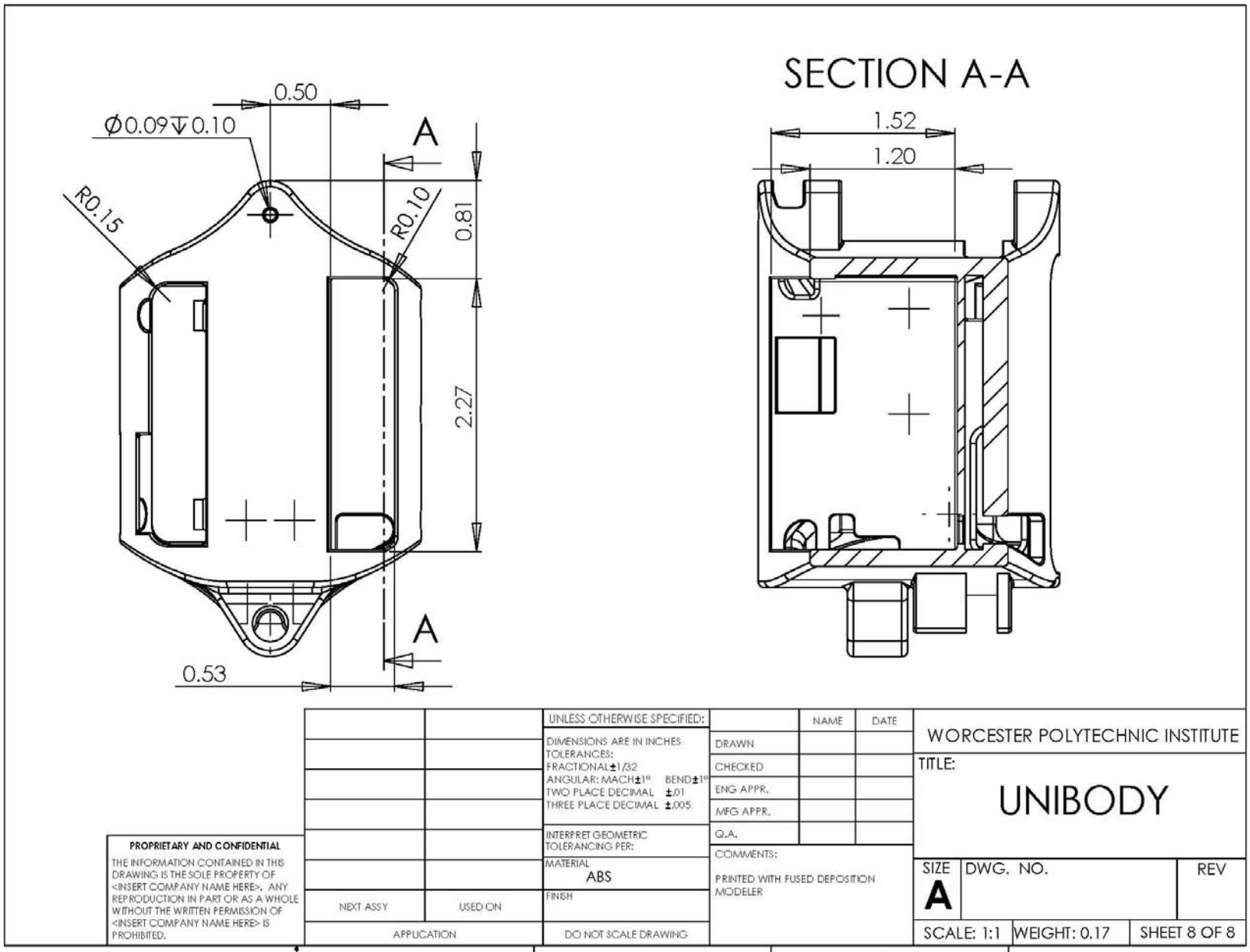
TITLE:

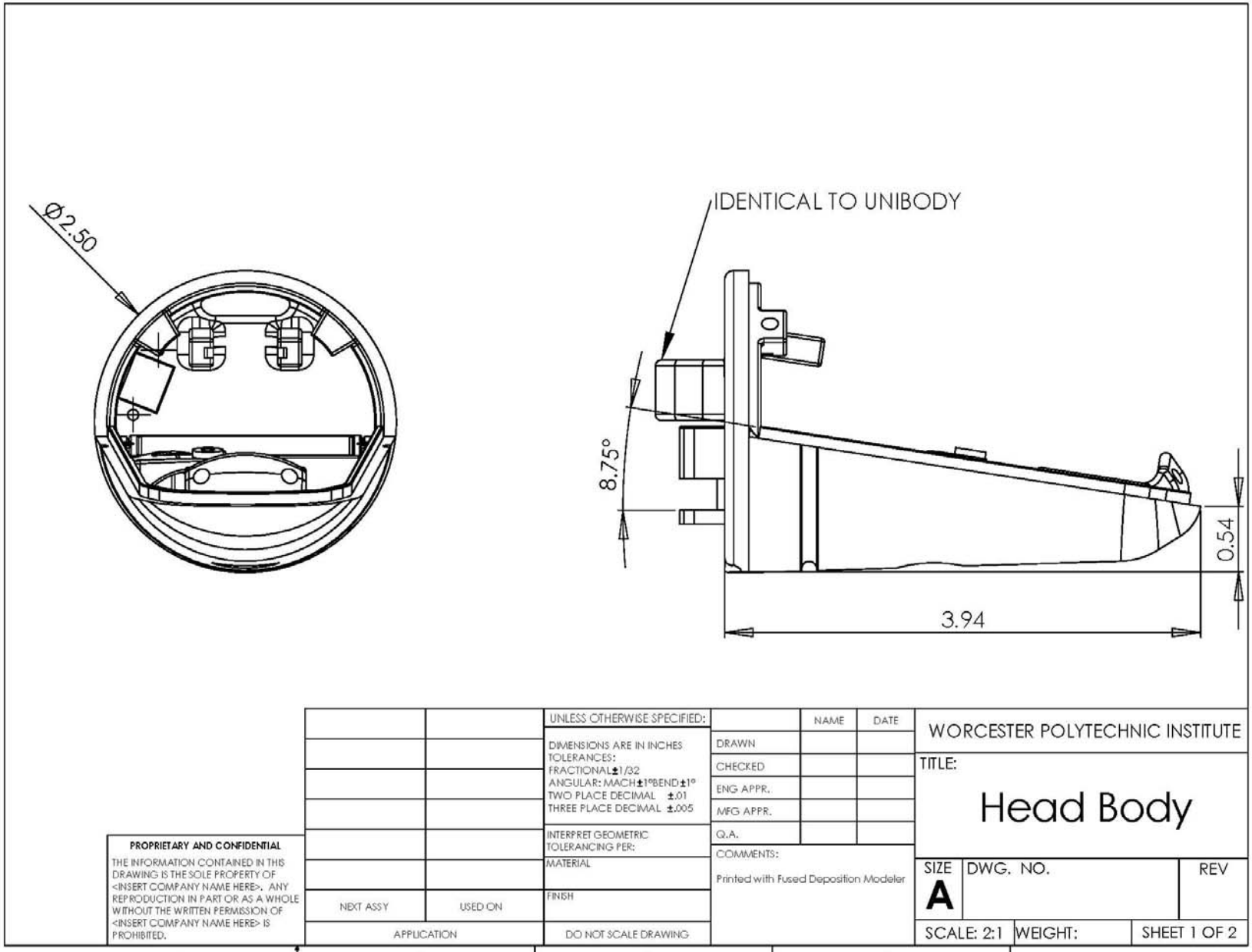
UNIBODY

SIZE	DWG. NO.	REV
A		

SCALE: 1:1	WEIGHT: 0.17	SHEET 7 OF 8

5 4 3 2 1


SECTION A-A
0.50
Ø0.09Ⓜ0.10
A
R0.15
R0.10
0.81
2.27
A
0.53
1.52
1.20
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES:
FRACTIONAL ±1/32
ANGULAR: MACH ±1° BEND ±1°
TWO PLACE DECIMAL ±.01
THREE PLACE DECIMAL ±.005
INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
ABS
FINISH
DO NOT SCALE DRAWING
NAME
DATE
DRAWN
CHECKED
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
PRINTED WITH FUSED DEPOSITION MODELER
NEXT ASSY
USED ON
APPLICATION
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:
UNIBODY
SIZE
A
DWG. NO.
REV
SCALE: 1:1
WEIGHT: 0.17
SHEET 8 OF 8
5
4
3
2
1



		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES 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	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.

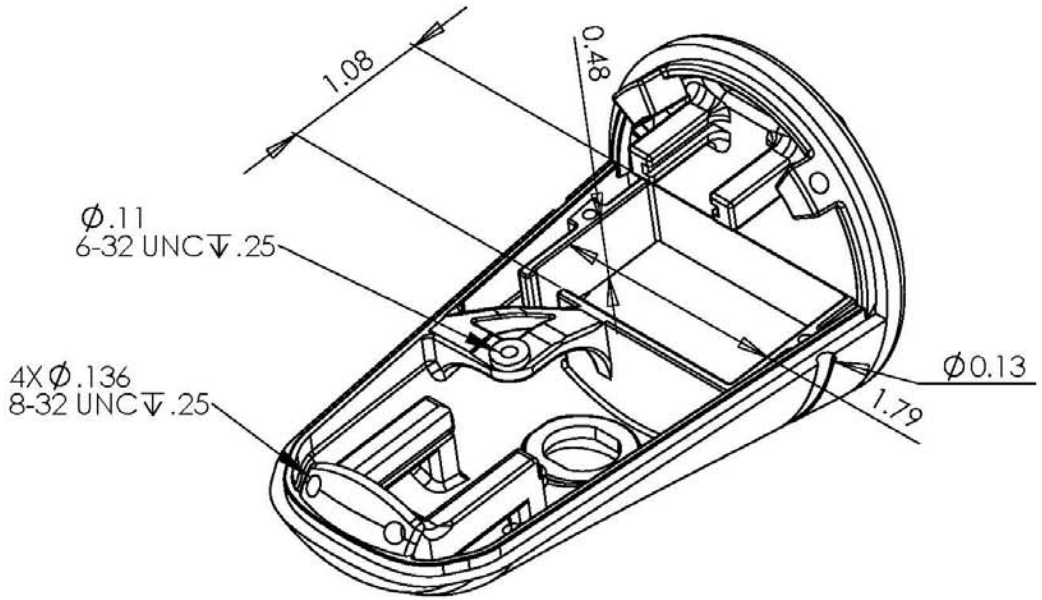
WORCESTER POLYTECHNIC INSTITUTE

TITLE:

Head Body

SIZE	DWG. NO.	REV
A		
SCALE: 2:1	WEIGHT:	SHEET 1 OF 2

5 4 3 2 1



		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES 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	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.

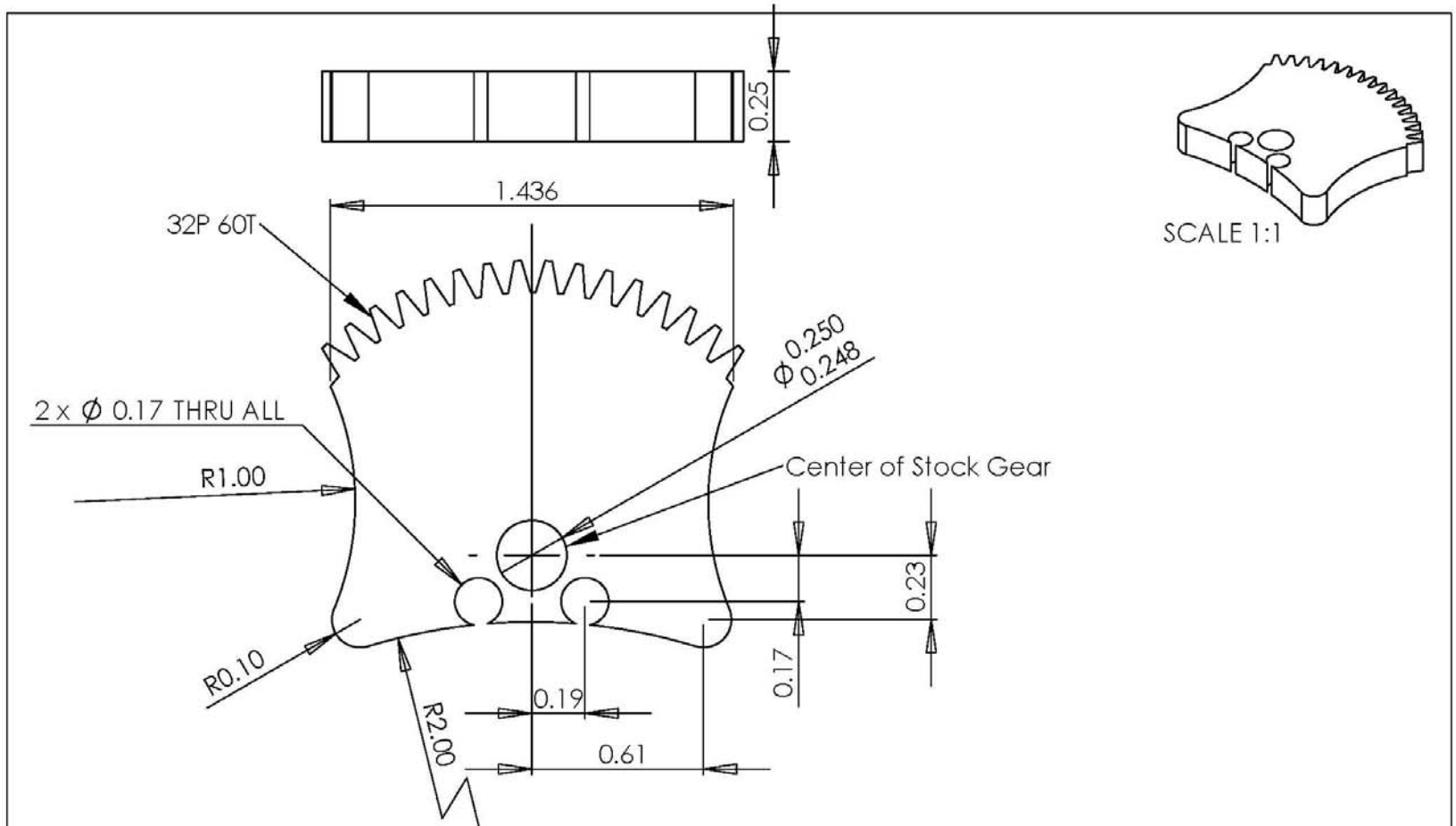
WORCESTER POLYTECHNIC INSTITUTE

TITLE:

Head Body

SIZE	DWG. NO.	REV
A		
SCALE: 2:1	WEIGHT:	SHEET 2 OF 2

5 4 3 2 1



		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES 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 Alloy Steel	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

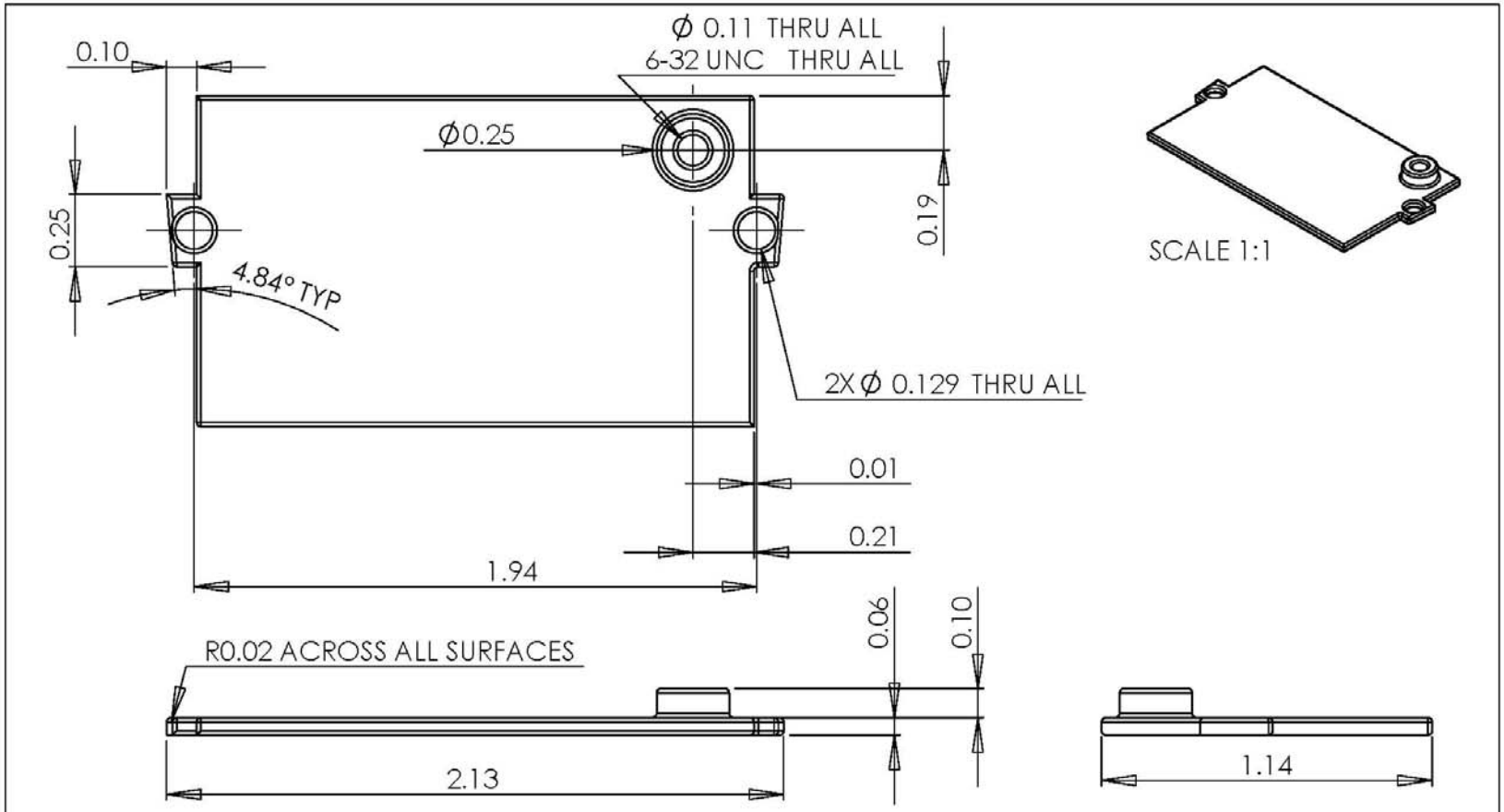
WORCESTER POLYTECHNIC INSTITUTE

TITLE:

Big Gear

SIZE	DWG. NO.	REV
A		
SCALE: 2:1	WEIGHT: 0.10	SHEET 1 OF 1

5 4 3 2 1



		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES 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	COMMENTS: Printed with Fused Deposition Modeler		
		FINISH			
NEXT ASSY	USED ON				
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:

Head Battery Lid

SIZE	DWG. NO.	REV
A		
SCALE: 2:1	WEIGHT:	SHEET 1 OF 1

5 4 3 2 1

Appendix E: Bill of Materials

Bill of Materials

Category	Part	Part #	Quantity	Cost per Unit	Total Cost	Source	Lead Time	Order	Receive	Status
Mechanical										
	Segment	N/A	15	$ -	$ -	WPI	1 Week	12	12	C
	Hektronix HXT12K Servo	N/A	12	$ 12.06	$ 144.72	www.unitedhobbies.com	2 Week	12	12	C
	Bronze Bearing	6391K122	14	$ 0.43	$ 6.02	McMaster	2 Days	14	14	C
	3/16" Dia 1' Shaft	88565K36	3	$ 7.14	$ 21.42	McMaster	2 Days	2	2	C
	8-32 SS 1/4" Cap Screw 100pk	92196A190	1	$ 6.39	$ 6.39	McMaster	2 Days	1	1	C
	Nylon Bearing 5pk	6389K353	3	$ 2.70	$ 8.10	McMaster	2 Days	3	3	C
	8-32 SS Flat Head 50pk	93085A197	1	$ 8.52	$ 8.52	McMaster	2 Days	1	1	C
	.02" THK Washer 25pk	93574A438	1	$ 7.81	$ 7.81	McMaster	2 Days	1	1	C
	.024-.038 THK Washer 100pk	92141A009	1	$ 1.82	$ 1.82	McMaster	2 Days	1	1	C
	100 Flat Head 8-32 1/4" screw	90471A250	1	$ 7.55	$ 7.55	McMaster	2 Days	1	1	C
	100 Pan Head 6-32 3/16" screws	90272A143	1	$ 1.58	$ 1.58	McMaster	2 Days	1	1	C
	100 Pan Head 4-40 3/16" Screws	90272A105	1	$ 1.35	$ 1.35	McMaster	2 Days	1	1	C
	Expandable Sleeving 15'	9284K393	1	$ 14.53	$ 14.53	McMaster	2 Days	1	1	C

	Spandex Sleeving	4896 & 4225	1	$ 32.20	$ 32.20	www.spandexworld.com	2 Days	1	1	C
	Servo Gear 16T 32P	RSA32-FMG-16	12	$ 14.95	$ 179.40	www.servocity.com	3 Days	12	12	C
	60T Gear	A 1C29-Y32060	1	83.88	$ 83.88	www.sdp-si.com	3 Days	1	1	C
Electrical										
	730mAh LiPoly 7.4V	LP-TP730-2SJPL2	12	$ 22.99	$ 275.88	www.robotmarketplace.com	1.5 Weeks	12	12	C
	250mAh LiPoly 7.4V	LP-TP250-2SJPL2	1	$ 12.99	$ 12.99	www.robotmarketplace.com	1 Week	1	1	C
	10 ft Audio Cable, 22/4 Awg	71335K52	1	$ 1.21	$ 1.21	McMaster	2 Days	1	1	C
	Bluetooth Modem	WRL-00582	1	$ 64.95	$ 64.95	www.sparkfun.com	1 Week	1	1	C
	Battery Connectors	LXPHE0, LXPHE4	1	60.23	$ 60.23	http://www.towerhobbies.com/	1 Week	1	1	C
	Power MOSFET	ZXMN4A06GCT-ND	15	1.26	$ 18.90	www.digikey.com	1 Week	15	15	C
	5v Linear Regulator	57M6063	15	3.76	$ 56.40	http://www.newark.com	1 Week	15	15	C
	Capacitors	10 uF POL EIA7343	15	0.6	$ 9.00	http://digikey.com/	1 Week	15	15	C
		.1 uF C0805	15	0.2	$ 3.00	http://digikey.com/	1 Week	15	15	C
		.01uF C0805	15	0.2	$ 3.00	http://digikey.com/	1 Week	15	15	C
		1uF C0805	30	0.2	$ 6.00	http://digikey.com/	1 Week	30	30	C
	Resistors	10k R0603	200	0.06	$ 12.00	http://digikey.com/	1 Week	200	200	C
		90.9k R0603	30	0.1	$ 3.00	http://digikey.com/	1 Week	30	30	C
		1.5k R0603	30	0.1	$ 3.00	http://digikey.com/	1 Week	30	30	C
		.1 R0603 1/2W	15	0.2	$ 3.00	http://digikey.com/	1 Week	15	15	C
	Processor	DSPIC30F4013	15	7	$ 105.00	http://www.microchipdirect.com/	1 Week	15	15	C
	OP-AMP	LM358	30	0.5	$ 15.00	http://digikey.com/	1 Week	15	15	C
	DIP Switch	GH7321-ND M:90HBJ06PT	15	1.73	$ 25.95	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=GH7231-	1 Week	15	15	C

						ND				
	Terminal Block 7 Pos	277-1278-ND	15	4.5	$ 67.50	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=277-1278-ND	1 Week	15	15	C
	H 2POS .1 STR TIN	WM8072-ND	30	0.21	$ 6.30	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=WM8072-ND	1 Week	30	30	C
	H 3POS .1 R/A TIN	WM4101-ND	15	0.36	$ 5.40	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=WM4101-ND	1 Week	15	15	C
	H 5POS .1 R/A TIN	WM8099-ND	15	0.57	$ 8.55	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=WM8099-ND	1 Week	15	15	C
	PCB	Eagle Design	15	$ 30.00	$ 450.00	http://www.sunstone.com/?gclid=CJPZ7OPGmJoCFZpM5QodRjko8w	1 Week	15	15	C
Test Bed										
	Granular Material	N/A	1	$ -	$ -	Ineos Nova	1 week	1	1	C
				TOTAL:	$ 1,741.55					

Appendix F: MATLAB Code

Traveling_Snake2.m

```
%traveling_snake2

%Waveform(function, numberoflinks, linklength, periods, startpoint,
endpoint, resolution)
%Waveform(13, .9, 1, 0, 10, .01);

clf
clc
clear all

clear, syms X;
Y = (1/2)*sin(X)
%Y = 2*cos(X) + sin(X)
%Y = cos(X^2)

counter = 1;
interations = 10;
linklength = .9;
numberoflinks = 13;
tablethetar = zeros(numberoflinks - 1, interations / .05);
tablethetad = tablethetar;
for i = interations: -.05: 0
    %[thetar thetad] = Waveform2(Y, 13, .9, 1, 0 + i, 10 + i, .01);

    [thetar thetad] = Waveform2(Y, numberoflinks, .9, 1.5, 0 + i, 13 +
i, .001);
    %[thetar thetad] = Waveform2(Y, numberoflinks, 1.8, 1.5, 0 + i, 25
+ i, .001);
    %[thetar thetad] = Waveform(Y, numberoflinks, 1.8, 1.5, 0 + i, 25 +
i, .01);

    tablethetar(:, counter) = thetar;
```

```
    tablethetad(:, counter) = thetad;
    counter = counter + 1;
end


tablethetar;
tablethetad
plotresults(linklength, tablethetar);
```

Waveform2.m

```
function [thetar thetad] = Waveform2(Y, numberoflinks ,linklength
,wavelength ,startpoint ,endpoint, resolution)


pointsperwavelength = ((endpoint-startpoint) / resolution) * (2/3);
conversion = wavelength / pointsperwavelength;



X = linspace(startpoint,endpoint,(endpoint-startpoint)/resolution);



y = eval(Y);
x = linspace(0,endpoint - startpoint,(endpoint-startpoint)/resolution);



A = ones(numberoflinks - 1,1);
B = zeros(numberoflinks - 1,1);
thetar = zeros(numberoflinks - 1,1);
theta = thetar;
for i = 1: 1: numberoflinks - 2
    if(i == 1)
        for j = 2: 1: (endpoint-startpoint)/resolution - 1
            if (((y(1,j) - y(1,1))^2 + (x(1,j) - x(1,1))^2) >=
linklength^2)
                    nextx = j;
                break;
            end
        end
```

```
    else
        for j = A(i): 1: (endpoint-startpoint)/resolution - 1
            if(j > 1)
                if (((y(1,j) - y(1,A(i)))^2 + (x(1,j) - x(1,A(i)))^2)
>= linklength^2)
                    nextx = j;
                    break;
                end
            else
               if ((y(1,1)^2 + (x(1,1))^2) >= linklength^2)
                    nextx = j;
                    break;
               end
            end
        end
    end
    nextx;
    A(i + 1, 1) = nextx;
    B(i + 1, 1) = y(1,nextx + 1);
end

B(1,1) = y(1,1);

figure(1)
plot(x,y);
hold on
A(:,1);
C(:,1) = x(1, A(:,1));
B(:,1);
plot(C(:,1),B(:,1),'--rs','color', [1 0 0]);
axis([0 (endpoint-startpoint) -5 5]);
hold off

for i = 1: numberoflinks - 2
  theta(i, 1) = atan((B(i + 1, 1) - B(i, 1))/(C(i + 1, 1) - C(i, 1)));
  if(i ~= 1)
      thetar(i, 1) = theta(i , 1) - theta(i-1, 1);
```

```
  else
      thetar(i, 1) = theta(i);
  end
end
thetad = thetar * 57.3;
```

PlotResults.m

```
function [] = plotresults(linklength, thetar);

theta = zeros(size(thetar,1),size(thetar,2));
pointsx = theta;
pointsy = pointsx;
for i = 1: size(thetar,2)
    for j = 1: size(thetar, 1)
        if(j ~= 1)
            theta(j, i) = thetar(j, i) + theta(j - 1,i);
        else
            theta(j, i) = thetar(j , i);
        end
    end
end

theta;

for i = 1: size(theta,2)
    for j = 2: size(theta, 1)
        pointsx(j, i) = linklength * cos(theta(j - 1,i)) + (pointsx(j -
1,i));
        pointsy(j, i) = linklength * sin(theta(j - 1,i)) + (pointsy(j -
1,i));
    end
end

figure(2)
for i = 1: size(theta, 2)
    plot(pointsx(:,i),pointsy(:,i),'--rs','color', [1 0 0]);
    axis([0 13 -5 5]);
```

```
    drawnow
end
```

Orientation.m

```
function [alpha beta gamma] = Orientation(Fx, Fy, Fz)


Fr = (Fx^2 + Fy^2 + Fz^2)^1/2;
Ufx = Fx/Fr;
Ufy = Fy/Fr;
Ufz = Fz/Fr;


alpha = acos(Fx);
beta = acos(Fy);
gamma = acos(Fz);
```

Appendix G: Onboard Software Code

Main.c

```
/*
 *@author Neal Humphrey
 *Main.c
 */

#define THIS_IS_STACK_APPLICATION
#include "WPIO.h"

// C30 and C32 Exception Handlers

// If your code gets here, you either tried to read or write

// a NULL pointer, or your application overflowed the stack

// by having too many local variables or parameters declared.

char string_buffer[16];
char number[10];
double* data_table;
char pos = 0;
char neg = 0;
char dataflag = 0;
char pos_val = 0;
long adc0_val = 0;
long adc1_val = 0;
long adc2_val = 0;
char data_flag = 0;
unsigned int data_elements = 0;
unsigned int buf_index = 0;
unsigned int table_index = 0;

//test
//unsigned int inputSignal[16];
volatile unsigned int * adcPtr;
```

```
unsigned int* iPtr;
unsigned int inputSignal[16];

long long waveform_index = 0;
double step_table[] = {-14.42160824,-15.56138443,-16.63714973,-17.67178029,-18.64946752,-
19.55963753,-20.42103069,-21.22182343,-21.95479251,-22.62692801,-23.23138724,-
23.78250945,-24.26631091,-24.69085014,-25.05025482,-25.35080436,-25.592543,-
25.77228381,-25.89482087,-25.95770475,-25.96419331,-25.91428895,-25.80773373,-
25.64589103,-25.42868137,-25.15753344,-24.83206941,-24.45501795,-24.02441543,-
23.54079341,-23.00407071,-22.41925353,-21.77778917,-21.08923873,-20.35026292,-
19.56821795,-18.7302271,-17.84433195,-16.91880348,-15.93972359,-14.91638372,-
13.85068684,-12.74554072,-11.60122577,-10.42268004,-9.200169388,-7.948310381,-
6.658900588,-5.358472879,-4.014726802,-2.656922734,-
1.289827267,0.094593125,1.480585882,2.874171617,4.257283021,5.63875909,6.998671382,8.
348660457,9.667383974,10.96456462,12.22256207,13.43686101,14.61870332,15.73824294,16
.81736682,17.84185135,18.80918055,19.71017541,20.56145188,21.34395456,22.06663188,22.
72848747,23.33058841,23.86490897,24.33927975,24.75364425,25.10770519,25.39418832,25.
62623452,25.79657365,25.90933944,25.96273956,25.95974623,25.90035833,25.78442283,25.
61316842,25.38754418,25.10763969,24.77341498,24.38484211,23.94517399,23.45232977,22.
91078894,22.31251639,21.667098,20.97050885,20.22323414,19.4259058,18.58706951,17.692
77276,16.7514088,15.77375881,14.743324,13.67087356,12.55863362,11.40911647,10.213412
07,8.985516792,7.728890933,6.44726621,5.1315021,3.785021114,2.438877672,1.057072138,-
0.328606209,-1.713079768,-3.105259332,-4.486116187,-5.864498203,-7.235181297,-
8.579349317,-9.892427762,-11.18320245,-12.43413406,-13.64161094,-14.80241911,-
15.92526069,-16.99526873,-18.01029822,-18.95847321,-19.85901003,-20.70014182,-
21.47245222,-22.18483459,-22.83679847,-23.42800313,-23.95197129,-24.41597384,-
24.81996429,-25.15938695,-25.4399674,-25.65832863,-25.81907908,-25.92226307,-
25.96617999,-25.95370473,-25.88483267,-25.75950353,-25.57884828,-25.34391144,-
25.05468619,-24.71113385,-24.31601285,-23.86767907,-23.36618221,-22.81162712,-
22.20923844,-21.55537189,-20.85040662,-20.09485057,-19.28936201,-18.4347715,-
17.54048388,-16.59151532,-15.59717443,-14.5691305,-13.48998744,-12.37145053,-
11.20472379,-10.01494487,-8.782166263,-7.521249509,-6.223093875,-4.90400872,-
3.568347362,-2.207018041,-
0.824207407,0.561623833,1.945396265,3.33602702,4.728421533,6.103481461,7.456274324,8.
795623655,10.11642875,11.3877074,12.63179449,13.84416259,14.99670537,16.11075991,17.
17159014,18.16661178,19.11559589,20.00613612,20.82815154,21.59919442,22.30126705,22.
9428517,23.5169016,24.03107661
```

```
};

/*globals*/
	BYTE INpacketArray[248];
	BYTE NOWpacketArray[248];
	int servoPos[NUM_SERVOS];
	BOOL processing;
	long i;
	BOOL running;
	char idxDat[239];
	int idxDatLen;
	int packetSize;
	int rxPacketIndex;
	BOOL packetReady;
	long range_0=0;
	long range_1=0;
	long sonic_1;
	long sonic_0;
	char sensors[8];
/*globals*/

#define MAX_ANGLE 57
#define US_PER_DEGREE 11.666666666
#define USER_ANGLE_TO_DEGREE 3.157894736
#define DRIVE_SERVO_NUM 3
#define MAX_ADC_VALUE 4096
#define MAX_ADC_mVOLTAGE 5000
#define ADC_mVOLTAGE_PER_VALUE 1.220703125

#if defined(__C30__)
	void _ISR __attribute__((__no_auto_psv__)) _AddressError(void)
	{
	   Nop();
		Nop();
	}
	void _ISR __attribute__((__no_auto_psv__)) _StackError(void)
	{
```

```
            Nop();
                Nop();
        }
#endif

void __attribute__((interrupt,auto_psv)) _U2RXInterrupt(void) {
                BYTE read;
                while(!DataRdyUART2()){
                }
                read = ReadUART2();

                if(read  != 'M' && read != 'U'){
                        WriteUART2((unsigned int)read);
                }



//              UpdateRangeFinder();

        switch (pos){
                        case 0:
                                if(read == 'P'){
                                        pos = 1;
                                        neg = 0;

                                        #if defined(SERVO_DEBUG)
                                                WriteUART2((unsigned int)read);
                                        #endif
                                }
                                if(read == 'W'){
                                        waveForm(step_table, waveform_index);
                                        if(waveform_index < TABLE_SIZE)
                                                waveform_index = waveform_index + 1;
                                        else
                                                waveform_index = 0;

                                        //WriteUART2((unsigned int) 10);
```

```
                //WriteUART2((unsigned int) 13);
        }
        if(read == 'C'){
                int i = 0;
                waveForm(step_table, 0);
                DelayMs(750);
                for(i = 0; i < TABLE_SIZE; i++){
                        waveForm(step_table, i);
                        WriteUART2((unsigned int) 'W');
                        //DelayMs(300);
                }

                WriteUART2((unsigned int) 10);
                WriteUART2((unsigned int) 13);
        }
        if(read == 'A'){
                pos = 5;
                //UpdateRangeFinder();
        }
        if(read == 'D'){
                pos = 6;
        }

        if(read == 'M'){
                for(i = 0; i < TABLE_SIZE; i++){
                        WriteUART2((unsigned int) 'W');
                        DelayMs(25);
                }
        }

        break;

case 1:
        if(read == '-'){
                neg = 1;

                #if defined(SERVO_DEBUG)
```

```
                    WriteUART2((unsigned int)read);
                #endif
            }
            else {
                pos_val = ((int)read - 48)  * 10;
                pos = 2;

                #if defined(SERVO_DEBUG)
                    WriteUART2((unsigned int)read);
                #endif
}
            break;

        case 2:
            pos_val = pos_val + ((int) read - 48);
            pos = 0;

            #if defined(SERVO_DEBUG)
                WriteUART2((unsigned int)read);
                WriteUART2((unsigned int)pos_val);
            #endif

            //setServo(DRIVE_SERVO_NUM, pos_val);
            setAngle(DRIVE_SERVO_NUM, pos_val);

            WriteUART2((unsigned int) 10);
            WriteUART2((unsigned int) 13);
            break;

        case 5: // ADC
            if(read == '0'){
                adc0_val = GetADC(0);

                #if defined(ADC_DEBUG)
                    WriteUART2((unsigned int)read);
                #endif
```

```
                    WriteUART2((unsigned int)58);
                    WriteUART2((unsigned int)32);

                    itoa(adc0_val, &string_buffer[0]);
                    putsUART2((unsigned int*)&string_buffer[0]);

                    WriteUART2((unsigned int)58);
                    WriteUART2((unsigned int)32);

                    adc0_val = adc_Val_to_mVoltage((unsigned
int)adc0_val);

                    itoa((int)adc0_val, &string_buffer[0]);
                    putsUART2((unsigned int*)&string_buffer[0]);

                    WriteUART2((unsigned int) 10);
                    WriteUART2((unsigned int) 13);
                }
                if(read == '1'){
                    adc1_val = GetADC(1);

                    #if defined(ADC_DEBUG)
                        WriteUART2((unsigned int)read);
                    #endif

                    WriteUART2((unsigned int)58);
                    WriteUART2((unsigned int)32);

                    itoa(adc1_val, &string_buffer[0]);
                    putsUART2((unsigned int*)&string_buffer[0]);

                    WriteUART2((unsigned int)58);
                    WriteUART2((unsigned int)32);

                    adc1_val = adc_Val_to_mVoltage((unsigned
int)adc1_val);
```

```
                                        //test
                                        //adc1_val = 12345;

                                        itoa((int)adc1_val, &string_buffer[0]);
                                        putsUART2((unsigned int*)&string_buffer[0]);

                                        WriteUART2((unsigned int) 10);
                                        WriteUART2((unsigned int) 13);
                                }
                                        if(read == '2'){
                                        adc2_val = GetADC(2);

                                        #if defined(ADC_DEBUG)
                                                WriteUART2((unsigned int)read);
                                        #endif

                                        WriteUART2((unsigned int)58);
                                        WriteUART2((unsigned int)32);

                                        itoa((int)adc2_val, &string_buffer[0]);
                                        putsUART2((unsigned int*)&string_buffer[0]);

                                        WriteUART2((unsigned int)58);
                                        WriteUART2((unsigned int)32);

                                        adc2_val = adc_Val_to_mVoltage((unsigned
int)adc2_val);

                                        itoa(adc2_val, &string_buffer[0]);
                                        putsUART2((unsigned int*)&string_buffer[0]);

                                        WriteUART2((unsigned int) 10);
                                        WriteUART2((unsigned int) 13);
                                }
                                pos = 0;
                        break;
                }
```

```
		IFS1bits.U2RXIF = 0; //Clr UART_Rx interrupt flag
}


void __attribute__((interrupt,auto_psv)) _U1RXInterrupt(void) {
		IFS0bits.U1RXIF = 0; //Clr UART_Rx interrupt flag
}

void __attribute__((interrupt,auto_psv)) _T1Interrupt(void) {
		//TimeoutPacket();
		IFS0bits.T1IF = 0;
	/*
		putsUART2(itoa(GetADC(0), number, DECIMAL));
		putsUART2(",");
		putsUART2(itoa(GetADC(1), number, DECIMAL));
		putsUART2("\r\n");
	*/
		TMR1 = 0x0000;// reset timer;
}

void setAngle(unsigned char number, double value){
	if (neg == 1){
		value = (value * -1);
	}

	setServo(number, (value + (MAX_ANGLE/2)) * USER_ANGLE_TO_DEGREE);
}

void setServo(unsigned char number, double value){
	if(value < 0)
		value = 0;

	unsigned int timedelay =  (value * US_PER_DEGREE);

	if (timedelay > 2000)
		timedelay = 2000;
```

```
	#if defined(SERVO_DEBUG)
		WriteUART2((unsigned int) timedelay / 10);
	#endif

	switch (number){
		case 2:
		//	SERVO2_IO = 1;
		//	Delay10us(60);
		//	Delay10us(timedelay / 10);
		//	Delay1us(timedelay % 10);
		//	ServoOff(2);
		//	DelayMs(20);
			break;
		case 3:
			SERVO3_IO = 1;
			Delay10us(60);
			Delay10us(timedelay / 10);
			Delay1us(timedelay % 10);
			ServoOff(3);
			DelayMs(20);
			break;
		case 15:
			SERVO15_IO = 1;
			Delay10us(60);
			Delay10us(timedelay / 10);
			Delay1us(timedelay % 10);
			ServoOff(15);
			DelayMs(20);
			break;
	}
}

void waveForm(double * step_table, long step_index){
	setAngle(DRIVE_SERVO_NUM, step_table[step_index]);
}
```

```
double adc_Val_to_mVoltage(int adc_value){
        return adc_value * ADC_mVOLTAGE_PER_VALUE;
}

/*********************************************************************
* Function:         void itoa(unsigned int Value, char *Buffer)
*
* PreCondition: None
*
* Input:            Value: Unsigned integer to be converted
*                       Buffer: Pointer to a location to write the string
*
* Output:           *Buffer: Receives the resulting string
*
* Side Effects: None
*
* Overview:         The function converts an unsigned integer (16 bits)
*                       into a null terminated decimal string.
*
* Note:             None
********************************************************************/
void itoa(unsigned int Value, char *Buffer)
{
    unsigned char i;
    unsigned int Digit;
    unsigned int Divisor;
    enum {FALSE = 0, TRUE} Printed = FALSE;

    if(Value)
    {
        for(i = 0, Divisor = 10000; i < 5; i++)
        {
            Digit = Value/Divisor;
            if(Digit || Printed)
            {
                *Buffer++ = '0' + Digit;
                Value -= Digit*Divisor;
```

```
                        Printed = TRUE;
                    }
                    Divisor /= 10;
                }
        }
        else
        {
            *Buffer++ = '0';
        }

        *Buffer = '\0';
}




/*globals*/
        BYTE INpacketArray[248];
        BYTE NOWpacketArray[248];
        int servoPos[NUM_SERVOS];
        BOOL processing;
        long i;
        BOOL running;
        char idxDat[239];
        int idxDatLen;
        int packetSize;
        int rxPacketIndex;
        BOOL packetReady;
/*globals*/

#define RELEASE


int main(void){
        rxPacketIndex = 0;//set index to zero so first byte received goes into position 0 of buffer
        processing = FALSE; //processing blocking flag
        running = FALSE;//initialize to no output on all pins
```

```
	packetReady = FALSE;//the flag for processing packet. must be poller as it will be set
asyncronuslly
	InitializeBoard();
	packetSize = 0;
	strcpy(idxDat, "Generic Servo controller. 8 bits precision.\n");
	idxDatLen = strlen(idxDat);

DelayMs(500);

WriteUART2(85);

while (1){
}

/*
	while(1){
		if (packetReady){
			ProcessPacket();
			packetReady = FALSE;
		}
//servo control code
#if defined(RELEASE)
		if (running){
#endif
			allOn(0);
			//run minimal .75 ms pulse
			DelayPreServo();
			//loop 255 times and turn off all servos as thier set position is equal to the
loop counter
			half = NUM_SERVOS/2;
			for (y=0;y<256;y++){
				for (x=0; x < 8 ;x++){
					if (servoPos[x] == y){
						ServoOff(x);
					}//turn off if it is time to turn off
				}//check all servo positions
			//add the delay each loop
```

```
                        DelayIncServo();
                        }

//split the outputs into 3 sections
                        allOn(1);
                        DelayPreServo();
                        for (y=0;y<256;y++){
                                for (x=8; x < 17 ;x++){
                                        if (servoPos[x] == y){
                                                ServoOff(x);
                                        }//turn off if it is time to turn off
                                }//check all servo positions
                        //add the delay each loop
                        DelayIncServo();
                        }
//Last part, might be full of dummies
                        allOn(2);
                        DelayPreServo();
                        for (y=0;y<256;y++){
                                for (x=17; x < NUM_SERVOS ;x++){
                                        if (servoPos[x] == y){
                                                ServoOff(x);
                                        }//turn off if it is time to turn off
                                }//check all servo positions
                        //add the delay each loop
                        DelayIncServo();
                        }
//end servo pulses
                        for (y=0;y < NUM_SERVOS;y++){
                                ServoOff(y);
                        }
                        //add post servo pulse delay
                        DelayMs(23);
#if defined(RELEASE)
                }
#endif
//END servo control code
```

```
        }//while 1
*/

return(0);
}//Main

/*
 *Straight forward setting output pins to logic high in 3 blocks
 */
void allOn(int section){

        //1us delays added because of bizare output, pin going high for 50ns then going low. If other cause for this
        //is found then these can be removed, however the rest of the timeings will need to be adjusted accordingly.
        if (section == 0){
//      SERVO0_IO = 1;
//      Delay1us(1);
//      SERVO1_IO = 1;
//      Delay1us(1);
//      SERVO2_IO = 1;
//      Delay1us(1);
        SERVO3_IO = 1;
        Delay1us(1);
        SERVO4_IO = 1;
        Delay1us(1);
        SERVO5_IO = 1;
        Delay1us(1);
        SERVO6_IO = 1;
        Delay1us(1);
        SERVO7_IO = 1;
        Delay1us(1);
        }
        if (section == 1){
        SERVO8_IO = 1;
        Delay1us(1);
```

```
SERVO9_IO = 1;
Delay1us(1);
SERVO10_IO = 1;
Delay1us(1);
SERVO11_IO = 1;
Delay1us(1);
SERVO12_IO = 1;
Delay1us(1);
SERVO13_IO = 1;
Delay1us(1);
SERVO14_IO = 1;
Delay1us(1);
SERVO15_IO = 1;
Delay1us(1);
SERVO16_IO = 1;
Delay1us(1);
}
if (section == 2){
SERVO17_IO = 1;
Delay1us(1);
SERVO18_IO = 1;
Delay1us(1);
SERVO19_IO = 1;
Delay1us(1);
SERVO20_IO = 1;
Delay1us(1);
SERVO21_IO = 1;
Delay1us(1);
SERVO22_IO = 1;
Delay1us(1);
SERVO23_IO = 1;
Delay1us(1);
SERVO24_IO = 1;
Delay1us(1);
SERVO25_IO = 1;
Delay1us(1);
}
```

```
}

/*
 *Turn off pins one at a time FIXME and make the servo IO a struct so iteration over it can
happen.
 */
void ServoOff(int servo){
	switch (servo){
	case 0:
		//SERVO0_IO = 0;
		break;
	case 1:
		//SERVO1_IO = 0;
		break;
	case 2:
		//SERVO2_IO = 0;
		break;
	case 3:
		SERVO3_IO = 0;
		break;
	case 4:
		SERVO4_IO = 0;
		break;
	case 5:
		SERVO5_IO = 0;
		break;
	case 6:
		SERVO6_IO = 0;
		break;
	case 7:
		SERVO7_IO = 0;
		break;
	case 8:
		SERVO8_IO = 0;
		break;
	case 9:
		SERVO9_IO = 0;
```

```
            break;
case 10:
            SERVO10_IO = 0;
            break;
case 11:
            SERVO11_IO = 0;
            break;
case 12:
            SERVO12_IO = 0;
            break;
case 13:
            SERVO13_IO = 0;
            break;
case 14:
            SERVO14_IO = 0;
            break;
case 15:
            SERVO15_IO = 0;
            break;
case 16:
            SERVO16_IO = 0;
            break;
case 17:
            SERVO17_IO = 0;
            break;
case 18:
            SERVO18_IO = 0;
            break;
case 19:
            SERVO19_IO = 0;
            break;
case 20:
            SERVO20_IO = 0;
            break;
case 21:
            SERVO21_IO = 0;
            break;
```

```
		case 22:
			SERVO22_IO = 0;
			break;
		case 23:
			SERVO23_IO = 0;
			break;
		case 24:
			SERVO24_IO = 0;
			break;
		case 25:
			SERVO25_IO = 0;
			break;
		default:
			break;
		}
}

/*
 *This function initializes all the controle regesters and sets initial values of variables and
 *tristates.
 */
void InitializeBoard(void)
{
//		for (i=0;i<SENSOR_BYTES;i++){
//			sensors[i]=0;
//		}
		InitADC();
		#if defined(__dsPIC30F4011__)
		PWMCON1 = 0x0000;//disable pwm on all pins
		OVDCON = 0x0000;//disable pwm on all pins
		FLTACON = 0x0000;//disable pwm on all pins
		#endif
		#if defined(__dsPIC30F4013__)
		//
		#endif

		//Timer timeout for serial receive. Interrupts every 15ms.
```

```
T1CONbits.TON = 1; //timer on
T1CONbits.TSIDL = 1;
T1CONbits.TCKPS = 1;
PR1 = 0xffff;

T2CONbits.TON = 1; //timer on
T2CONbits.TSIDL = 1;
T2CONbits.TCKPS = 2;
PR2 = 0xffff;

//initialize the servo positions to center
int i;
for (i=0;i<NUM_SERVOS;i++){
servoPos[i]=INIT_VALUE;
}

// SERVO pins to outputs
//SERVO0_TRIS = 0;
//SERVO1_TRIS = 0;
//SERVO2_TRIS = 0;
SERVO3_TRIS = 0;
SERVO4_TRIS = 0;
SERVO5_TRIS = 0;
SERVO6_TRIS = 0;
SERVO7_TRIS = 0;
SERVO8_TRIS = 0;
SERVO9_TRIS = 0;
SERVO10_TRIS = 0;
//SERVO11_TRIS = 0;
//SERVO12_TRIS = 0;
SERVO13_TRIS = 0;
SERVO14_TRIS = 0;
SERVO15_TRIS = 0;
SERVO16_TRIS = 0;
SERVO17_TRIS = 0;
SERVO18_TRIS = 0;
SERVO19_TRIS = 0;
```

```
	SERVO20_TRIS = 0;
	SERVO21_TRIS = 0;
	SERVO22_TRIS = 0;
	SERVO23_TRIS = 0;
	//SERVO24_TRIS = 0;
	//SERVO25_TRIS = 0;

	T2CONbits.TON = 0; //timer on
	T2CONbits.TSIDL = 1;
	T2CONbits.TCKPS = 2;
	PR2=0xFFFF;

	SONIC0_TRIS = 0;
	SONIC1_TRIS = 0;
	SONIC_INT0_TRIS = 1;
	SONIC_INT1_TRIS = 1;

	POSITION_SENSE_TRIS = 1;
	CURRENT_SENSE_TRIS = 1;
	VOLTAGE_SENSE_TRIS = 1;

//serial port 2 setup
	UART2TX_TRIS = 0;
	UART2RX_TRIS = 1;
	U2MODE = 0x8000;			// Set UARTEN.  Note: this must be done before
setting UTXEN
//RX interupt enabled
	U2STA = 0x8400;			// UTXEN set
	#define CLOSEST_UBRG_VALUE2
((GetPeripheralClock()+8ul*BAUD_RATE2)/16/BAUD_RATE2-1)
	#define BAUD_ACTUAL2 (GetPeripheralClock()/16/(CLOSEST_UBRG_VALUE2+1))
	#define BAUD_ERROR2 ((BAUD_ACTUAL2 > BAUD_RATE2) ? BAUD_ACTUAL2-
BAUD_RATE2 : BAUD_RATE2-BAUD_ACTUAL2)
	#define BAUD_ERROR_PRECENT2
	((BAUD_ERROR2*100+BAUD_RATE2/2)/BAUD_RATE2)
	#if (BAUD_ERROR_PRECENT2 > 3)
		#warning UART frequency error is worse than 3%
```

```
        #elif (BAUD_ERROR_PRECENT2 > 2)
                #warning UART frequency error is worse than 2%
        #endif
        U2BRG = CLOSEST_UBRG_VALUE2;
//Enableing Interupts
        IFS1bits.U2RXIF = 0;
        IEC1bits.U2RXIE = 1; //Enable UART Rx receive interrupt
        //IPC6bits.U2RXIP = 3;
        IEC0bits.T1IE = 1;//endble timer 1 interupt.

#if defined(USE_UART1)
                //serial port 1 setup
                        UART1TX_TRIS = 0;
                        UART1RX_TRIS = 1;
                        U1MODE = 0x8000;                        // Set UARTEN.  Note: this must
be done before setting UTXEN
                //RX interupt enabled
                        U1STA = 0x8400;                 // UTXEN set
                        #define CLOSEST_UBRG_VALUE1
((GetPeripheralClock()+8ul*BAUD_RATE1)/16/BAUD_RATE1-1)
                        #define BAUD_ACTUAL1
(GetPeripheralClock()/16/(CLOSEST_UBRG_VALUE1+1))
                        #define BAUD_ERROR ((BAUD_ACTUAL1 > BAUD_RATE1) ?
BAUD_ACTUAL1-BAUD_RATE1 : BAUD_RATE1-BAUD_ACTUAL1)
                        #define BAUD_ERROR_PRECENT11
        ((BAUD_ERROR1*100+BAUD_RATE1/2)/BAUD_RATE1)
                        #if (BAUD_ERROR_PRECENT1 > 3)
                                #warning UART frequency error is worse than 3%
                        #elif (BAUD_ERROR_PRECENT1 > 2)
                                #warning UART frequency error is worse than 2%
                        #endif
                        U1BRG = CLOSEST_UBRG_VALUE1;
                //Enableing Interupts
                        IFS0bits.U1RXIF = 0;
                        IEC0bits.U1RXIE = 1; //Enable UART Rx receive interrupt

                        IPC2bits.U1RXIP = 3;
```

```
#endif
//Roomba Init

        //WriteUART1(128);
        //WriteUART1(130);
        SONIC1_IO = 1;
}

void UpdateRangeFinder(void){

        if (ADCBUF0 >0){
                range_0 = (ADCBUF0);//scaled to produce distance in 100th of an inch
        }
        else
                range_0 = 65000;
        if (ADCBUF1 >0){
                range_1 = (ADCBUF1);//scaled to produce distance in 100th of an inch
        }
        else
                range_1 = 65000;


        WriteUART2((unsigned int)58);
        WriteUART2((unsigned int)32);

        itoa(range_0, &string_buffer[0]);
        putsUART2((unsigned int*)&string_buffer[0]);

        WriteUART2((unsigned int)58);
        WriteUART2((unsigned int)32);

        range_0 = adc_Val_to_mVoltage((unsigned int)range_0);

        itoa((int)range_0, &string_buffer[0]);
        putsUART2((unsigned int*)&string_buffer[0]);

        WriteUART2((unsigned int) 10);
```

```
        WriteUART2((unsigned int) 13);


        sensors[0]=((char)((range_0 & 0xFF00) >>8));
        sensors[1]=((char)(range_0 & 0x00FF));

        sensors[2]=((char)((range_1 & 0xFF00) >>8));
        sensors[3]=((char)(range_1 & 0x00FF));

        sensors[4]=((char)((sonic_0 & 0x0F00) >>8));
        sensors[5]=((char)(sonic_0 & 0x00FF));

        sensors[6]=((char)((sonic_1 & 0x0F00) >>8));
        sensors[7]=((char)(sonic_1 & 0x00FF));

}

 /*********************************************************************/
```

ADC.c

```
#include "WPIO.h"

int GetADC(int chan){
        if (chan == 0)
                return ADCBUF0;
        else if (chan == 1)
                return ADCBUF1;
        else if (chan == 2)
                return ADCBUF2;
        return -1;
}

void InitADC(void){
        AD1CHS = 0x0007;                //enable ADC0
        AD1PCFG = 0xFFF8;               //enable ADC0
        // ADC
        AD1CON1 = 0x04E4;                       // Turn on, auto sample start, auto-convert, 12
bit mode (on parts with a 12bit A/D)
```

```
        AD1CON2 = 0x0404;                   // AVdd, AVss, int every 2 conversions, MUXA
only, scan
        AD1CON3 = 0x1003;                   // 16 Tad auto-sample, Tad = 3*Tcy
        AD1CSSL = 0x0007;                       // scan range finders
        ADCON1bits.FORM      = 0;               // Output in Integer Format
        ADCON1bits.ADON = 1;                    // Start the ADC module

        ADCBUF0 = 0;
        ADCBUF1 = 0;
        ADCBUF2 = 0;
}
```

UART.c

```
#define __UART_C


#include "WPIO.h"
#if defined(STACK_USE_UART)


#if defined(__C30__) // PIC24F, PIC24H, dsPIC30, dsPIC33

/*****************************************************************************
* Function Name     : putsUART2                                             *
* Description       : This function puts the data string to be transmitted *
*                     into the transmit buffer (till NULL character)       *
* Parameters        : unsigned int * address of the string buffer to be    *
*                     transmitted                                          *
* Return Value      : None                                                  *
*****************************************************************************/

void putsUART2(unsigned int *buffer)
{
   char * temp_ptr = (char *) buffer;

   /* transmit till NULL character is encountered */
```

```
    if(U2MODEbits.PDSEL == 3)        /* check if TX is 8bits or 9bits */
    {
        while(*buffer != '\0')
        {
            while(U2STAbits.UTXBF); /* wait if the buffer is full */
            U2TXREG = *buffer++;    /* transfer data word to TX reg */
        }
    }
    else
    {
        while(*temp_ptr != '\0')
        {
            while(U2STAbits.UTXBF);  /* wait if the buffer is full */
            U2TXREG = *temp_ptr++;   /* transfer data byte to TX reg */
        }
    }
}


/*****************************************************************************
* Function Name     : getsUART2                                              *
* Description       : This function gets a string of data of specified length *
*                     if available in the UxRXREG buffer into the buffer     *
*                     specified.                                             *
* Parameters        : unsigned int length the length expected                *
*                     unsigned int *buffer  the received data to be          *
*                             recorded to this array                         *
*                     unsigned int uart_data_wait timeout value              *
* Return Value      : unsigned int number of data bytes yet to be received   *
*****************************************************************************/

unsigned int getsUART2(unsigned int length,unsigned int *buffer,
                       unsigned int uart_data_wait)

{
    unsigned int wait = 0;
    char *temp_ptr = (char *) buffer;
```

```
   while(length)                    /* read till length is 0 */
   {
      while(!DataRdyUART2())
      {
         if(wait < uart_data_wait)
            wait++ ;                /*wait for more data */
         else
            return(length);         /*Time out- Return words/bytes to be read */
      }
      wait=0;
      if(U2MODEbits.PDSEL == 3)        /* check if TX/RX is 8bits or 9bits */
         *buffer++ = U2RXREG;          /* data word from HW buffer to SW buffer */
                  else
         *temp_ptr++ = U2RXREG & 0xFF; /* data byte from HW buffer to SW buffer */

      length--;
   }

   return(length);                  /* number of data yet to be received i.e.,0 */
}


/*********************************************************************
* Function Name     : DataRdyUart2                                   *
* Description       : This function checks whether there is any data *
*                     that can be read from the input buffer, by     *
*                     checking URXDA bit                             *
* Parameters        : None                                           *
* Return Value      : char if any data available in buffer           *
*********************************************************************/

char DataRdyUART2(void)
{
   return(U2STAbits.URXDA);
}
```

```
/*****************************************************************************
* Function Name     : BusyUART2                                             *
* Description       : This returns status whether the transmission        *
*                     is in progress or not, by checking Status bit TRMT *
* Parameters        : None                                                  *
* Return Value      : char info whether transmission is in progress      *
*****************************************************************************/

char BusyUART2(void)
{
   return(!U2STAbits.TRMT);
}


/*****************************************************************************
* Function Name     : ReadUART2                                             *
* Description       : This function returns the contents of UxRXREG buffer *
* Parameters        : None                                                  *
* Return Value      : unsigned int value from UxRXREG receive buffer       *
*****************************************************************************/

unsigned int ReadUART2(void)
{
   if(U2MODEbits.PDSEL == 3)
      return (U2RXREG);
   else
      return (U2RXREG & 0xFF);
}


/*****************************************************************************
* Function Name     : WriteUART2                                            *
* Description       : This function writes data into the UxTXREG,    *
* Parameters        : unsigned int data the data to be written       *
* Return Value      : None                                                  *
*****************************************************************************/
```

```
void WriteUART2(unsigned int data)
{

   if(U2MODEbits.PDSEL == 3)
      U2TXREG = data;
   else
      U2TXREG = data & 0xFF;
         while(U2STAbits.UTXBF);
}



/*****************************************************************************
* Function Name     : putsUART1                                             *
* Description       : This function puts the data string to be transmitted *
*                     into the transmit buffer (till NULL character)       *
* Parameters        : unsigned int * address of the string buffer to be    *
*                     transmitted                                          *
* Return Value      : None                                                  *
*****************************************************************************/

void putsUART1(unsigned int *buffer)
{
   char * temp_ptr = (char *) buffer;

   /* transmit till NULL character is encountered */

   if(U1MODEbits.PDSEL == 3)        /* check if TX is 8bits or 9bits */
   {
      while(*buffer != '\0')
      {
         while(U1STAbits.UTXBF); /* wait if the buffer is full */
         U1TXREG = *buffer++;    /* transfer data word to TX reg */
      }
   }
   else
```

```
    {
        while(*temp_ptr != '\0')
        {
            while(U1STAbits.UTXBF);  /* wait if the buffer is full */
            U1TXREG = *temp_ptr++;   /* transfer data byte to TX reg */
        }
    }
}


/*****************************************************************************
* Function Name     : getsUART1                                              *
* Description       : This function gets a string of data of specified length *
*                     if available in the UxRXREG buffer into the buffer     *
*                     specified.                                             *
* Parameters        : unsigned int length the length expected                *
*                     unsigned int *buffer  the received data to be          *
*                             recorded to this array                         *
*                     unsigned int uart_data_wait timeout value              *
* Return Value      : unsigned int number of data bytes yet to be received   *
*****************************************************************************/

unsigned int getsUART1(unsigned int length,unsigned int *buffer,
                       unsigned int uart_data_wait)

{
    unsigned int wait = 0;
    char *temp_ptr = (char *) buffer;

    while(length)                   /* read till length is 0 */
    {
        while(!DataRdyUART1())
        {
            if(wait < uart_data_wait)
                wait++ ;            /*wait for more data */
            else
                return(length);     /*Time out- Return words/bytes to be read */
```

```
        }
        wait=0;
        if(U1MODEbits.PDSEL == 3)        /* check if TX/RX is 8bits or 9bits */
           *buffer++ = U1RXREG;          /* data word from HW buffer to SW buffer */
                    else
           *temp_ptr++ = U1RXREG & 0xFF; /* data byte from HW buffer to SW buffer */

        length--;
     }

     return(length);                     /* number of data yet to be received i.e.,0 */
}


/*************************************************************************
* Function Name     : DataRdyUART1                                      *
* Description       : This function checks whether there is any data *
*                     that can be read from the input buffer, by     *
*                     checking URXDA bit                              *
* Parameters        : None                                           *
* Return Value      : char if any data available in buffer           *
*************************************************************************/

char DataRdyUART1(void)
{
   return(U1STAbits.URXDA);
}


/*************************************************************************
* Function Name     : BusyUART1                                         *
* Description       : This returns status whether the transmission       *
*                     is in progress or not, by checking Status bit TRMT *
* Parameters        : None                                             *
* Return Value      : char info whether transmission is in progress     *
*************************************************************************/
```

```
char BusyUART1(void)
{
   return(!U1STAbits.TRMT);
}


/*****************************************************************************
* Function Name     : ReadUART1                                             *
* Description       : This function returns the contents of UxRXREG buffer *
* Parameters        : None                                                  *
* Return Value      : unsigned int value from UxRXREG receive buffer        *
*****************************************************************************/

unsigned int ReadUART1(void)
{
   if(U1MODEbits.PDSEL == 3)
      return (U1RXREG);
   else
      return (U1RXREG & 0xFF);
}


/*****************************************************************************
* Function Name     : WriteUART1                                            *
* Description       : This function writes data into the UxTXREG,    *
* Parameters        : unsigned int data the data to be written       *
* Return Value      : None                                                  *
*****************************************************************************/

void WriteUART1(unsigned int data)
{

   if(U1MODEbits.PDSEL == 3)
      U1TXREG = data;
   else
      U1TXREG = data & 0xFF;
         while(U1STAbits.UTXBF);
```

```
}
#endif


#endif   //STACK_USE_UART


/*********************************************************************
 *
 *    UART access routines for C18 and C30
 *
 *********************************************************************
 * FileName:        UART.c
 * Dependencies:    Hardware UART module
 * Processor:       PIC18, PIC24F, PIC24H, dsPIC30F, dsPIC33F
 * Compiler:        Microchip C30 v3.01 or higher
 *                                      Microchip C18 v3.13 or higher
 *                                      HI-TECH PICC-18 STD 9.50PL3 or higher
 * Company:         Microchip Technology, Inc.
 *
 * Software License Agreement
 *
 * Copyright (C) 2002-2008 Microchip Technology Inc.  All rights
 * reserved.
 *
 * Microchip licenses to you the right to use, modify, copy, and
 * distribute:
 * (i)  the Software when embedded on a Microchip microcontroller or
 *      digital signal controller product ("Device") which is
 *      integrated into Licensee's product; or
 * (ii) ONLY the Software driver source files ENC28J60.c and
 *      ENC28J60.h ported to a non-Microchip device used in
 *      conjunction with a Microchip ethernet controller for the
 *      sole purpose of interfacing with the ethernet controller.
 *
 * You should refer to the license agreement accompanying this
 * Software for additional information regarding your rights and
```

```
 * obligations.
 *
 * THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT
 * WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT
 * LIMITATION, ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL
 * MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR
 * CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF
 * PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS
 * BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE
 * THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER
 * SIMILAR COSTS, WHETHER ASSERTED ON THE BASIS OF CONTRACT, TORT
 * (INCLUDING NEGLIGENCE), BREACH OF WARRANTY, OR OTHERWISE.
 *
 *
 * Author            Date                Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Howard Schlunder          4/04/06       Copied from dsPIC30 libraries
 * Howard Schlunder          6/16/06       Added PIC18
********************************************************************/
```

ISR.c

```
#include "WPIO.h"


BYTE sonSel = 0;
void __attribute__((interrupt,auto_psv)) _T2Interrupt(void) {
                if (sonSel == 0){
                        SONIC1_IO = 0;
                        SONIC0_IO = 1;
                        sonSel = 1;
                }else{
                        SONIC1_IO = 1;
                        SONIC0_IO = 0;
                        sonSel = 0;
                }
                IFS0bits.T2IF = 0;
}
```

```
void __attribute__ ((interrupt,auto_psv)) _INT2Interrupt(void)
{
	extern	long sonic_1;

	IFS1bits.INT2IF = 0;
	if ((INTCON2bits.INT2EP == 0)&&(SONIC_INT1_IO == 1)){
		SONIC1_IO = 0;
		//SONIC1_IO = 0;
		TMR2 = 0x0000;
		INTCON2bits.INT2EP = 1;
		sonSel = 1;
	}
	else if((INTCON2bits.INT2EP == 1)&&(SONIC_INT1_IO == 0)) {
		SONIC0_IO = 1;
		sonic_1 = TMR2;
		//SONIC1_IO = 1;
		INTCON2bits.INT2EP = 0;
	}
}

void __attribute__ ((interrupt,auto_psv)) _INT1Interrupt(void)
{
	extern	long sonic_0;
	IFS1bits.INT1IF = 0;
	if ((INTCON2bits.INT1EP == 0)&&(SONIC_INT0_IO == 1)){
		SONIC0_IO = 0;
		//SONIC1_IO = 0;
		TMR2 = 0x0000;
		INTCON2bits.INT1EP = 1;
		sonSel = 0;
	}
	else if ((INTCON2bits.INT1EP == 1)&&(SONIC_INT0_IO == 0)){
		SONIC1_IO = 1;
		sonic_0 = TMR2;
		//SONIC1_IO = 1;
```

```
                INTCON2bits.INT1EP = 0;
        }
}
```

Delay.c

```
/********************************************************************
 *
 *              General Delay rouines
 *
 ********************************************************************
 * FileName:        Delay.c
 * Dependencies:    Compiler.h
 * Processor:       PIC18, PIC24F, PIC24H, dsPIC30F, dsPIC33F, PIC32
 * Compiler:        Microchip C32 v1.00 or higher
 *                                      Microchip C30 v3.01 or higher
 *                                      Microchip C18 v3.13 or higher
 *                                      HI-TECH PICC-18 STD 9.50PL3 or higher
 * Company:         Microchip Technology, Inc.
 *
 * Software License Agreement
 *
 * Copyright (C) 2002-2008 Microchip Technology Inc.  All rights
 * reserved.
 *
 * Microchip licenses to you the right to use, modify, copy, and
 * distribute:
 * (i)  the Software when embedded on a Microchip microcontroller or
 *      digital signal controller product ("Device") which is
 *      integrated into Licensee's product; or
 * (ii) ONLY the Software driver source files ENC28J60.c and
 *      ENC28J60.h ported to a non-Microchip device used in
 *      conjunction with a Microchip ethernet controller for the
 *      sole purpose of interfacing with the ethernet controller.
 *
 * You should refer to the license agreement accompanying this
 * Software for additional information regarding your rights and
 * obligations.
```

```
 *
 * THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT
 * WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT
 * LIMITATION, ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL
 * MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR
 * CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF
 * PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS
 * BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE
 * THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER
 * SIMILAR COSTS, WHETHER ASSERTED ON THE BASIS OF CONTRACT, TORT
 * (INCLUDING NEGLIGENCE), BREACH OF WARRANTY, OR OTHERWISE.
 *
 *
 * Author              Date    Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Nilesh Rajbharti     5/9/02  Original        (Rev 1.0)
 ********************************************************************/
#define __DELAY_C

#include "WPIO.h"

void DelayMs(WORD ms)
{
   unsigned char i;
   while(ms--)
   {
      i=4;
      while(i--)
      {
         Delay10us(25);
      }
   }
}


void DelayPreServo(void)
```

```
{
   unsigned char i;
   i=4;
   while(i--)
   {
      Delay10us(13);
   }
}

void DelayIncServo(void)
{

   Delay1us(2);

}



void Delay10us(DWORD dwCount)
{
        volatile DWORD _dcnt;

        _dcnt = dwCount*((DWORD)(0.00002/(3.0/GetInstructionClock())/10));
        while(_dcnt--);
}

void Delay1us(DWORD dwCount)
{
        while(dwCount--);
}
```

Complier.h

```
/*********************************************************************
 *
 *               Compiler and hardware specific definitions
 *
 *********************************************************************
```

```
* FileName:        Compiler.h
* Dependencies:    None
* Processor:       PIC18, PIC24F, PIC24H, dsPIC30F, dsPIC33F, PIC32
* Compiler:        Microchip C32 v1.00 or higher
*                                       Microchip C30 v3.01 or higher
*                                       Microchip C18 v3.13 or higher
*                                       HI-TECH PICC-18 STD 9.50PL3 or higher
* Company:         Microchip Technology, Inc.
*
* Software License Agreement
*
* Copyright (C) 2002-2008 Microchip Technology Inc.  All rights
* reserved.
*
* Microchip licenses to you the right to use, modify, copy, and
* distribute:
* (i)  the Software when embedded on a Microchip microcontroller or
*      digital signal controller product ("Device") which is
*      integrated into Licensee's product; or
* (ii) ONLY the Software driver source files ENC28J60.c and
*      ENC28J60.h ported to a non-Microchip device used in
*      conjunction with a Microchip ethernet controller for the
*      sole purpose of interfacing with the ethernet controller.
*
* You should refer to the license agreement accompanying this
* Software for additional information regarding your rights and
* obligations.
*
* THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT
* WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT
* LIMITATION, ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL
* MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR
* CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF
* PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS
* BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE
* THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER
```

```
 * SIMILAR COSTS, WHETHER ASSERTED ON THE BASIS OF CONTRACT, TORT
 * (INCLUDING NEGLIGENCE), BREACH OF WARRANTY, OR OTHERWISE.
 *
 *
 * Author            Date            Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Howard Schlunder              10/03/2006      Original, copied from old Compiler.h
 * Howard Schlunder              11/07/2007      Reorganized and simplified
 ********************************************************************/
#ifndef __COMPILER_H
#define __COMPILER_H

// Include proper device header file
#if defined(__18CXX) || defined(HI_TECH_C)
        // All PIC18 processors
        #if defined(HI_TECH_C)          // HI TECH PICC-18 compiler
                #define __18CXX
                #include <htc.h>
        #else                                   // Microchip C18 compiler
           #include <p18cxxx.h>
        #endif
#elif defined(__PIC24F__)       // Microchip C30 compiler
        // PIC24F processor
        #include <p24Fxxxx.h>
#elif defined(__PIC24H__)       // Microchip C30 compiler
        // PIC24H processor
        #include <p24Hxxxx.h>
#elif defined(__dsPIC33F__)     // Microchip C30 compiler
        // dsPIC33F processor
        #include <p33Fxxxx.h>
#elif defined(__dsPIC30F__)     // Microchip C30 compiler
        // dsPIC30F processor
        #include <p30fxxxx.h>
#elif defined(__PIC32MX__)      // Microchip C32 compiler
        #if !defined(__C32__)
                #define __C32__
        #endif
```

```
        #include <p32xxxx.h>
        #include <plib.h>
#else
        #error Unknown processor or compiler.  See Compiler.h
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>


// Base RAM and ROM pointer types for given architecture
#if defined(__C32__)
        #define PTR_BASE                DWORD
        #define ROM_PTR_BASE        DWORD
#elif defined(__C30__)
        #define PTR_BASE                WORD
        #define ROM_PTR_BASE        WORD
#elif defined(__18CXX)
        #define PTR_BASE                WORD
        #define ROM_PTR_BASE        unsigned short long
#endif


// Definitions that apply to all compilers, except C18
#if !defined(__18CXX) || defined(HI_TECH_C)
        #define memcmppgm2ram(a,b,c)        memcmp(a,b,c)
        #define strcmppgm2ram(a,b)          strcmp(a,b)
        #define memcpypgm2ram(a,b,c)memcpy(a,b,c)
        #define strcpypgm2ram(a,b)          strcpy(a,b)
        #define strncpypgm2ram(a,b,c)  strncpy(a,b,c)
        #define strstrrampgm(a,b)           strstr(a,b)
        #define strlenpgm(a)                strlen(a)
        #define strchrpgm(a,b)              strchr(a,b)
        #define strcatpgm2ram(a,b)          strcat(a,b)
#endif
```

```
// Definitions that apply to all 8-bit products
// (PIC18)
#if defined(__18CXX)
        #define __attribute__(a)

   #define FAR                  far

        // Microchip C18 specific defines
        #if !defined(HI_TECH_C)
          #define ROM                   rom
                #define strcpypgm2ram(a, b)             strcpypgm2ram(a,(far rom char*)b)
        #endif

        // HI TECH PICC-18 STD specific defines
        #if defined(HI_TECH_C)
          #define ROM                   const
                #define rom
          #define Nop()                 asm("NOP");
                #define ClrWdt()                                asm("CLRWDT");
          #define Reset()                                       asm("RESET");
        #endif

// Definitions that apply to all 16-bit and 32-bit products
// (PIC24F, PIC24H, dsPIC30F, dsPIC33F, and PIC32)
#else
        #define ROM                                             const

        // 16-bit specific defines (PIC24F, PIC24H, dsPIC30F, dsPIC33F)
        #if defined(__C30__)
                #define Reset()                         asm("reset")
      #define FAR               __attribute__((far))
        #endif

        // 32-bit specific defines (PIC32)
        #if defined(__C32__)
                #define persistent
```

```
                #define far
        #define FAR
                #define Reset()                 SoftReset()
                #define ClrWdt()                (WDTCONSET =
_WDTCON_WDTCLR_MASK)
                #define Nop()                   asm("nop")
            #endif
#endif



#endif
```

Delay.h

```
/*********************************************************************
 *
 *                General Delay rouines
 *
 *********************************************************************
 * FileName:        Delay.h
 * Dependencies:    Compiler.h
 * Processor:       PIC18, PIC24F, PIC24H, dsPIC30F, dsPIC33F, PIC32
 * Compiler:        Microchip C32 v1.00 or higher
 *                                      Microchip C30 v3.01 or higher
 *                                      Microchip C18 v3.13 or higher
 *                                      HI-TECH PICC-18 STD 9.50PL3 or higher
 * Company:         Microchip Technology, Inc.
 *
 * Software License Agreement
 *
 * Copyright (C) 2002-2008 Microchip Technology Inc.  All rights
 * reserved.
 *
 * Microchip licenses to you the right to use, modify, copy, and
 * distribute:
 * (i)  the Software when embedded on a Microchip microcontroller or
 *      digital signal controller product ("Device") which is
```

```
 *      integrated into Licensee's product; or
 * (ii) ONLY the Software driver source files ENC28J60.c and
 *      ENC28J60.h ported to a non-Microchip device used in
 *      conjunction with a Microchip ethernet controller for the
 *      sole purpose of interfacing with the ethernet controller.
 *
 * You should refer to the license agreement accompanying this
 * Software for additional information regarding your rights and
 * obligations.
 *
 * THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT
 * WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT
 * LIMITATION, ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL
 * MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR
 * CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF
 * PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS
 * BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE
 * THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER
 * SIMILAR COSTS, WHETHER ASSERTED ON THE BASIS OF CONTRACT, TORT
 * (INCLUDING NEGLIGENCE), BREACH OF WARRANTY, OR OTHERWISE.
 *
 *
 * Author               Date    Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Nilesh Rajbharti     5/9/02  Original        (Rev 1.0)
 * Nilesh Rajbharti     6/10/02 Fixed C18 ms and us routines
 * Howard Schlunder                4/04/06 Changed for C30
 ********************************************************************/
#ifndef __DELAY_H
#define __DELAY_H

#include "Compiler.h"
#include "HardwareProfile.h"
#if !defined(GetInstructionClock)
        #error GetInstructionClock() must be defined.
#endif
```

```
#if defined(__C30__) || defined(__C32__)
        void Delay10us(DWORD dwCount);
        void DelayMs(WORD ms);
        void Delay1us(DWORD dwCount);
        void DelayPreServo(void);
        void DelayIncServo(void);
#endif



#endif
```

GenericTypeDef.h

```
/*********************************************************************
 *
 *                Generic Type Definitions
 *
 *********************************************************************
 * FileName:        GenericTypeDefs.h
 * Dependencies:          None
 * Processor:       PIC18, PIC24F, PIC24H, dsPIC30F, dsPIC33F, PIC32
 * Complier:        Microchip C18, C30, C32
 *                                          HI-TECH PICC-18
 * Company:         Microchip Technology, Inc.
 *
 * Software License Agreement
 *
 * The software supplied herewith by Microchip Technology Incorporated
 * (the "Company") is intended and supplied to you, the Company's
 * customer, for use solely and exclusively with products manufactured
 * by the Company.
 *
 * The software is owned by the Company and/or its supplier, and is
 * protected under applicable copyright laws. All rights are reserved.
 * Any use in violation of the foregoing restrictions may subject the
 * user to criminal sanctions under applicable laws, as well as to
```

```
 * civil liability for the breach of the terms and conditions of this
 * license.
 *
 * THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES,
 * WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
 * TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 * PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
 * IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
 * CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 *
 *
 * Author                                        Date       Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Nilesh Rajbharti                          07/12/04        Rel 0.9
 * Nilesh Rajbharti                          11/24/04        Rel 0.9.1
 * Rawin Rojvanit                            09/17/05        Rel 0.9.2
 * D Flowers & H Schlunder       08/10/06        Much better now (1.0)
 * D Flowers & H Schlunder       09/11/06        Add base signed types (1.1)
 * D Flo, H Sch, et. al          02/28/07        Add QWORD, LONGLONG, QWORD_VAL (1.2)
 * Bud Caldwell          02/06/08    Added def's for PIC32
 ********************************************************************/

#ifndef __GENERIC_TYPE_DEFS_H_
#define __GENERIC_TYPE_DEFS_H_

typedef enum _BOOL { FALSE = 0, TRUE } BOOL;        // Undefined size

#ifndef NULL
#define NULL    0//((void *)0)
#endif

#define PUBLIC                          // Function attributes
#define PROTECTED
#define PRIVATE   static

typedef unsigned char           BYTE;                       // 8-bit unsigned
typedef unsigned short int      WORD;                       // 16-bit unsigned
```

```
typedef unsigned long       DWORD;                  // 32-bit unsigned
typedef unsigned long long  QWORD;                  // 64-bit unsigned
typedef signed char         CHAR;                   // 8-bit signed
typedef signed short int    SHORT;                  // 16-bit signed
typedef signed long         LONG;                   // 32-bit signed
typedef signed long long    LONGLONG;               // 64-bit signed

/* Alternate definitions */
typedef void        VOID;

typedef char        CHAR8;
typedef unsigned char   UCHAR8;

/* Processor & Compiler independent, size specific definitions */
// To Do:  We need to verify the sizes on each compiler.  These
//      may be compiler specific, we should either move them
//      to "compiler.h" or #ifdef them for compiler type.
typedef signed int      INT;
typedef signed char     INT8;
typedef signed short int    INT16;
typedef signed long int     INT32;
typedef signed long long    INT64;

typedef unsigned int    UINT;
typedef unsigned char   UINT8;
typedef unsigned short int  UINT16;
typedef unsigned long int   UINT32;  // other name for 32-bit integer
typedef unsigned long long  UINT64;

typedef union _BYTE_VAL
{
   BYTE Val;
   struct
   {
      unsigned char b0:1;
      unsigned char b1:1;
      unsigned char b2:1;
```

```
        unsigned char b3:1;
        unsigned char b4:1;
        unsigned char b5:1;
        unsigned char b6:1;
        unsigned char b7:1;
    } bits;
} BYTE_VAL, BYTE_BITS;

typedef union _WORD_VAL
{
    WORD Val;
    BYTE v[2];
    struct
    {
        BYTE LB;
        BYTE HB;
    } byte;
    struct
    {
        unsigned char b0:1;
        unsigned char b1:1;
        unsigned char b2:1;
        unsigned char b3:1;
        unsigned char b4:1;
        unsigned char b5:1;
        unsigned char b6:1;
        unsigned char b7:1;
        unsigned char b8:1;
        unsigned char b9:1;
        unsigned char b10:1;
        unsigned char b11:1;
        unsigned char b12:1;
        unsigned char b13:1;
        unsigned char b14:1;
        unsigned char b15:1;
    } bits;
} WORD_VAL, WORD_BITS;
```

```
typedef union _DWORD_VAL
{
    DWORD Val;
          WORD w[2];
    BYTE v[4];
    struct
    {
        WORD LW;
        WORD HW;
    } word;
    struct
    {
        BYTE LB;
        BYTE HB;
        BYTE UB;
        BYTE MB;
    } byte;
    struct
    {
        WORD_VAL low;
        WORD_VAL high;
    }wordUnion;
    struct
    {
        unsigned char b0:1;
        unsigned char b1:1;
        unsigned char b2:1;
        unsigned char b3:1;
        unsigned char b4:1;
        unsigned char b5:1;
        unsigned char b6:1;
        unsigned char b7:1;
        unsigned char b8:1;
        unsigned char b9:1;
        unsigned char b10:1;
        unsigned char b11:1;
```

```
        unsigned char b12:1;
        unsigned char b13:1;
        unsigned char b14:1;
        unsigned char b15:1;
        unsigned char b16:1;
        unsigned char b17:1;
        unsigned char b18:1;
        unsigned char b19:1;
        unsigned char b20:1;
        unsigned char b21:1;
        unsigned char b22:1;
        unsigned char b23:1;
        unsigned char b24:1;
        unsigned char b25:1;
        unsigned char b26:1;
        unsigned char b27:1;
        unsigned char b28:1;
        unsigned char b29:1;
        unsigned char b30:1;
        unsigned char b31:1;
    } bits;
} DWORD_VAL;

#define LSB(a)          ((a).v[0])
#define MSB(a)          ((a).v[1])

#define LOWER_LSB(a)    ((a).v[0])
#define LOWER_MSB(a)    ((a).v[1])
#define UPPER_LSB(a)    ((a).v[2])
#define UPPER_MSB(a)    ((a).v[3])

typedef union _QWORD_VAL
{
    QWORD Val;
            DWORD d[2];
            WORD w[4];
    BYTE v[8];
```

```
struct
{
    DWORD LD;
    DWORD HD;
} dword;
struct
{
    WORD LW;
    WORD HW;
    WORD UW;
    WORD MW;
} word;
struct
{
    unsigned char b0:1;
    unsigned char b1:1;
    unsigned char b2:1;
    unsigned char b3:1;
    unsigned char b4:1;
    unsigned char b5:1;
    unsigned char b6:1;
    unsigned char b7:1;
    unsigned char b8:1;
    unsigned char b9:1;
    unsigned char b10:1;
    unsigned char b11:1;
    unsigned char b12:1;
    unsigned char b13:1;
    unsigned char b14:1;
    unsigned char b15:1;
    unsigned char b16:1;
    unsigned char b17:1;
    unsigned char b18:1;
    unsigned char b19:1;
    unsigned char b20:1;
    unsigned char b21:1;
    unsigned char b22:1;
```

```
unsigned char b23:1;
unsigned char b24:1;
unsigned char b25:1;
unsigned char b26:1;
unsigned char b27:1;
unsigned char b28:1;
unsigned char b29:1;
unsigned char b30:1;
unsigned char b31:1;
unsigned char b32:1;
unsigned char b33:1;
unsigned char b34:1;
unsigned char b35:1;
unsigned char b36:1;
unsigned char b37:1;
unsigned char b38:1;
unsigned char b39:1;
unsigned char b40:1;
unsigned char b41:1;
unsigned char b42:1;
unsigned char b43:1;
unsigned char b44:1;
unsigned char b45:1;
unsigned char b46:1;
unsigned char b47:1;
unsigned char b48:1;
unsigned char b49:1;
unsigned char b50:1;
unsigned char b51:1;
unsigned char b52:1;
unsigned char b53:1;
unsigned char b54:1;
unsigned char b55:1;
unsigned char b56:1;
unsigned char b57:1;
unsigned char b58:1;
unsigned char b59:1;
```

```
        unsigned char b60:1;
        unsigned char b61:1;
        unsigned char b62:1;
        unsigned char b63:1;
    } bits;
} QWORD_VAL;

#endif //__GENERIC_TYPE_DEFS_H_
```

HardwareProfile.h

```
/********************************************************************
 *
 *          Hardware specific definitions
 *
 ********************************************************************
 * FileName:        HardwareProfile.h
 * Dependencies:    None
 * Processor:       PIC18, PIC24F, PIC24H, dsPIC30F, dsPIC33F, PIC32
 * Compiler:        Microchip C32 v1.00 or higher
 *                                      Microchip C30 v3.01 or higher
 *                                      Microchip C18 v3.13 or higher
 *                                      HI-TECH PICC-18 STD 9.50PL3 or higher
 * Company:         Microchip Technology, Inc.
 *
 * Software License Agreement
 *
 * Copyright (C) 2002-2008 Microchip Technology Inc.  All rights
 * reserved.
 *
 * Microchip licenses to you the right to use, modify, copy, and
 * distribute:
 * (i)  the Software when embedded on a Microchip microcontroller or
 *      digital signal controller product ("Device") which is
 *      integrated into Licensee's product; or
 * (ii) ONLY the Software driver source files ENC28J60.c and
 *      ENC28J60.h ported to a non-Microchip device used in
 *      conjunction with a Microchip ethernet controller for the
 *      sole purpose of interfacing with the ethernet controller.
```

```
 *
 * You should refer to the license agreement accompanying this
 * Software for additional information regarding your rights and
 * obligations.
 *
 * THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT
 * WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT
 * LIMITATION, ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL
 * MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR
 * CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF
 * PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS
 * BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE
 * THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER
 * SIMILAR COSTS, WHETHER ASSERTED ON THE BASIS OF CONTRACT, TORT
 * (INCLUDING NEGLIGENCE), BREACH OF WARRANTY, OR OTHERWISE.
* Author            Date            Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Howard Schlunder              10/03/06        Original, copied from Compiler.h
 ********************************************************************/
#ifndef __HARDWARE_PROFILE_H
#define __HARDWARE_PROFILE_H

#define WPIO

//#define DEBUG

#if defined(DEBUG)

#endif

// Set configuration fuses (but only once)
#if defined(THIS_IS_STACK_APPLICATION)
        #if defined(__dsPIC30F__)
                // dsPICDEM 1.1 board
                //_FOSC(XT_PLL16)
        // XT Osc + 16X PLL
```

```
                _FOSC(FRC_PLL16)                        // Internal Osc + 16X PLL
                _FWDT(WDT_OFF)
                _FBORPOR(MCLR_EN & PWRT_64 & PBOR_ON & BORV_45)
    #endif
#endif // Prevent more than one set of config fuse definitions

// Clock frequency value.
// This value is used to calculate Tick Counter value

#if defined(__dsPIC30F__)
        // dsPIC30F processor
        #define GetSystemClock()            (117920000ul)     // Hz
        #define GetInstructionClock()   (GetSystemClock()/4)
        #define GetPeripheralClock()    GetInstructionClock()
#endif

#if defined(WPIO)
        #if defined(__dsPIC30F4011__)

                #define SERVO0_TRIS             (TRISEbits.TRISE0)
                #define SERVO0_IO                       (PORTEbits.RE0)
                #define SERVO1_TRIS                     (TRISEbits.TRISE1)
                #define SERVO1_IO                       (PORTEbits.RE1)
                #define SERVO2_TRIS             (TRISEbits.TRISE2)
                #define SERVO2_IO                       (PORTEbits.RE2)
                #define SERVO3_TRIS             (TRISEbits.TRISE3)
                #define SERVO3_IO                       (PORTEbits.RE3)
                #define SERVO4_TRIS             (TRISEbits.TRISE4)
                #define SERVO4_IO                       (PORTEbits.RE4)
                #define SERVO5_TRIS                     (TRISEbits.TRISE5)
                #define SERVO5_IO                       (PORTEbits.RE5)

                #define SERVO6_TRIS                     (TRISFbits.TRISF6)
                #define SERVO6_IO                       (PORTFbits.RF6)
                #define SERVO7_TRIS                     (TRISDbits.TRISD2)
                #define SERVO7_IO                       (PORTDbits.RD2)
```

```
#define SERVO8_TRIS             (TRISDbits.TRISD3)
#define SERVO8_IO                   (PORTDbits.RD3)
#define SERVO9_TRIS             (TRISDbits.TRISD1)
#define SERVO9_IO                   (PORTDbits.RD1)

#define SERVO10_TRIS                    (TRISBbits.TRISB8)
#define SERVO10_IO                  (PORTBbits.RB8)
#define SERVO11_TRIS                    (TRISBbits.TRISB7)
#define SERVO11_IO                  (PORTBbits.RB7)
#define SERVO12_TRIS                    (TRISBbits.TRISB6)
#define SERVO12_IO                  (PORTBbits.RB6)
#define SERVO13_TRIS                    (TRISBbits.TRISB5)
#define SERVO13_IO                  (PORTBbits.RB5)
#define SERVO14_TRIS                    (TRISBbits.TRISB4)
#define SERVO14_IO                  (PORTBbits.RB4)
#define SERVO15_TRIS                (TRISBbits.TRISB3)
#define SERVO15_IO                  (PORTBbits.RB3)

#define SERVO16_TRIS                (TRISEbits.TRISE8)
#define SERVO16_IO                  (PORTEbits.RE8)

#define SERVO17_TRIS                (TRISCbits.TRISC14)
#define SERVO17_IO                  (PORTCbits.RC14)

#define SERVO18_TRIS                (TRISCbits.TRISC13)
#define SERVO18_IO                  (PORTCbits.RC13)

#define SERVO19_TRIS                (TRISCbits.TRISC15)
#define SERVO19_IO                  (PORTCbits.RC15)

#define SERVO20_TRIS                    dummy
#define SERVO20_IO                      dummy
#define SERVO21_TRIS                    dummy
#define SERVO21_IO                      dummy
#define SERVO22_TRIS                    dummy
#define SERVO22_IO                      dummy
#define SERVO23_TRIS                    dummy
```

```
        #define SERVO23_IO                      dummy

        #define SERVO24_TRIS                    dummy
        #define SERVO24_IO                      dummy
        #define SERVO25_TRIS                    dummy
        #define SERVO25_IO                      dummy
#elif defined(__dsPIC30F4013__)

        //#define SERVO0_TRIS                   (TRISBbits.TRISB0)
        //#define SERVO0_IO                     (PORTBbits.RB0)
        //#define SERVO1_TRIS                   (TRISBbits.TRISB1)
        //#define SERVO1_IO                     (PORTBbits.RB1)
        //#define SERVO2_TRIS                   (TRISBbits.TRISB2)
        //#define SERVO2_IO                     (PORTBbits.RB2)
        #define SERVO3_TRIS                     (TRISBbits.TRISB3)
        #define SERVO3_IO                       (PORTBbits.RB3)
        #define SERVO4_TRIS                     (TRISBbits.TRISB4)
        #define SERVO4_IO                       (PORTBbits.RB4)
        #define SERVO5_TRIS                     (TRISBbits.TRISB5)
        #define SERVO5_IO                       (PORTBbits.RB5)

        #define SERVO6_TRIS                     (TRISBbits.TRISB8)
        #define SERVO6_IO                       (PORTBbits.RB8)

        #define SERVO7_TRIS                     (TRISCbits.TRISC15)
        #define SERVO7_IO                       (PORTCbits.RC15)
        #define SERVO8_TRIS                     (TRISCbits.TRISC13)
        #define SERVO8_IO                       (PORTCbits.RC13)
        #define SERVO9_TRIS                     (TRISCbits.TRISC14)
        #define SERVO9_IO                       (PORTCbits.RC14)

        #define SERVO10_TRIS                    (TRISAbits.TRISA11)
        #define SERVO10_IO                      (PORTAbits.RA11)

        #define SERVO11_TRIS                    (TRISDbits.TRISD9)
        #define SERVO11_IO                      (PORTDbits.RD9)
        #define SERVO12_TRIS                    (TRISDbits.TRISD3)
```

```
#define SERVO12_IO                      (PORTDbits.RD3)

#define SERVO13_TRIS                    (TRISBbits.TRISB9)
#define SERVO13_IO                      (PORTBbits.RB9)
#define SERVO14_TRIS                    (TRISBbits.TRISB10)
#define SERVO14_IO                      (PORTBbits.RB10)
#define SERVO15_TRIS                    (TRISBbits.TRISB11)
#define SERVO15_IO                      (PORTBbits.RB11)
#define SERVO16_TRIS                    (TRISBbits.TRISB12)
#define SERVO16_IO                      (PORTBbits.RB12)

#define SERVO17_TRIS                    (TRISDbits.TRISD0)
#define SERVO17_IO                      (PORTDbits.RD0)
#define SERVO18_TRIS                    (TRISDbits.TRISD1)
#define SERVO18_IO                      (PORTDbits.RD1)

#define SERVO19_TRIS                    (TRISFbits.TRISF0)
#define SERVO19_IO                      (PORTFbits.RF0)
#define SERVO20_TRIS                    (TRISFbits.TRISF1)
#define SERVO20_IO                      (PORTFbits.RF1)
#if defined(USE_UART1)

                #define SERVO21_TRIS
(TRISFbits.TRISF6)
                #define SERVO21_IO
(PORTFbits.RF6)

                #define SERVO22_TRIS
(TRISDbits.TRISD8)
                #define SERVO22_IO
(PORTDbits.RD8)
                #define SERVO23_TRIS
(TRISDbits.TRISD2)
                #define SERVO23_IO
(PORTDbits.RD2)

                #define SERVO24_TRIS                    dummy
```

```
                #define SERVO24_IO                  dummy
                #define SERVO25_TRIS                dummy
                #define SERVO25_IO                  dummy
        #else
                #define SERVO21_TRIS
(TRISFbits.TRISF2)
                #define SERVO21_IO
(PORTFbits.RF2)
                #define SERVO22_TRIS
(TRISFbits.TRISF3)
                #define SERVO22_IO
(PORTFbits.RF3)
                #define SERVO23_TRIS
(TRISFbits.TRISF6)
                #define SERVO23_IO
(PORTFbits.RF6)

                #define SERVO24_TRIS
(TRISDbits.TRISD8)
                #define SERVO24_IO
(PORTDbits.RD8)
                #define SERVO25_TRIS
(TRISDbits.TRISD2)
                #define SERVO25_IO
(PORTDbits.RD2)
        #endif
#elif defined(__dsPIC30F3014__)

        //#define SERVO0_TRIS               (TRISBbits.TRISB0)
        //#define SERVO0_IO             (PORTBbits.RB0)
        //#define SERVO1_TRIS               (TRISBbits.TRISB1)
        //#define SERVO1_IO             (PORTBbits.RB1)
        //#define SERVO2_TRIS               (TRISBbits.TRISB2)
        //#define SERVO2_IO             (PORTBbits.RB2)
        #define SERVO3_TRIS             (TRISBbits.TRISB3)
        #define SERVO3_IO               (PORTBbits.RB3)
        #define SERVO4_TRIS             (TRISBbits.TRISB4)
```

```
#define SERVO4_IO                (PORTBbits.RB4)
#define SERVO5_TRIS              (TRISBbits.TRISB5)
#define SERVO5_IO                (PORTBbits.RB5)

#define SERVO6_TRIS              (TRISBbits.TRISB8)
#define SERVO6_IO                (PORTBbits.RB8)

#define SERVO7_TRIS              (TRISCbits.TRISC15)
#define SERVO7_IO                (PORTCbits.RC15)
#define SERVO8_TRIS              (TRISCbits.TRISC13)
#define SERVO8_IO                (PORTCbits.RC13)
#define SERVO9_TRIS              (TRISCbits.TRISC14)
#define SERVO9_IO                (PORTCbits.RC14)

#define SERVO10_TRIS             (TRISAbits.TRISA11)
#define SERVO10_IO               (PORTAbits.RA11)

//#define SERVO11_TRIS           (TRISDbits.TRISD9)
//#define SERVO11_IO             (PORTDbits.RD9)
//#define SERVO12_TRIS           (TRISDbits.TRISD3)
//#define SERVO12_IO             (PORTDbits.RD3)

#define SERVO13_TRIS             (TRISBbits.TRISB9)
#define SERVO13_IO               (PORTBbits.RB9)
#define SERVO14_TRIS             (TRISBbits.TRISB10)
#define SERVO14_IO               (PORTBbits.RB10)
#define SERVO15_TRIS             (TRISBbits.TRISB11)
#define SERVO15_IO               (PORTBbits.RB11)
#define SERVO16_TRIS             (TRISBbits.TRISB12)
#define SERVO16_IO               (PORTBbits.RB12)

#define SERVO17_TRIS             (TRISDbits.TRISD0)
#define SERVO17_IO               (PORTDbits.RD0)
#define SERVO18_TRIS             (TRISDbits.TRISD1)
#define SERVO18_IO               (PORTDbits.RD1)

#define SERVO19_TRIS             (TRISFbits.TRISF0)
```

```
        #define SERVO19_IO                      (PORTFbits.RF0)
        #define SERVO20_TRIS                    (TRISFbits.TRISF1)
        #define SERVO20_IO                      (PORTFbits.RF1)
        #if defined(USE_UART1)

                        #define SERVO21_TRIS
(TRISFbits.TRISF6)
                        #define SERVO21_IO
(PORTFbits.RF6)

                        #define SERVO22_TRIS
(TRISDbits.TRISD8)
                        #define SERVO22_IO
(PORTDbits.RD8)
                        #define SERVO23_TRIS
(TRISDbits.TRISD2)
                        #define SERVO23_IO
(PORTDbits.RD2)

                        #define SERVO24_TRIS            dummy
                        #define SERVO24_IO              dummy
                        #define SERVO25_TRIS            dummy
                        #define SERVO25_IO              dummy
        #else
                        #define SERVO21_TRIS
(TRISFbits.TRISF2)
                        #define SERVO21_IO
(PORTFbits.RF2)
                        #define SERVO22_TRIS
(TRISFbits.TRISF3)
                        #define SERVO22_IO
(PORTFbits.RF3)
                        #define SERVO23_TRIS
(TRISFbits.TRISF6)
                        #define SERVO23_IO
(PORTFbits.RF6)
```

```
                        //#define SERVO24_TRIS
(TRISDbits.TRISD8)
                        //#define SERVO24_IO
(PORTDbits.RD8)
                        //#define SERVO25_TRIS
(TRISDbits.TRISD2)
                        //#define SERVO25_IO
(PORTDbits.RD2)
        #endif


#else

        #define SERVO0_TRIS             dummy
        #define SERVO0_IO               dummy
        #define SERVO1_TRIS             dummy
        #define SERVO1_IO               dummy
        #define SERVO2_TRIS             dummy
        #define SERVO2_IO               dummy
        #define SERVO3_TRIS             dummy
        #define SERVO3_IO               dummy
        #define SERVO4_TRIS             dummy
        #define SERVO4_IO               dummy
        #define SERVO5_TRIS             dummy
        #define SERVO5_IO               dummy
        #define SERVO6_TRIS             dummy
        #define SERVO6_IO               dummy
        #define SERVO7_TRIS             dummy
        #define SERVO7_IO               dummy
        #define SERVO8_TRIS             dummy
        #define SERVO8_IO               dummy
        #define SERVO9_TRIS             dummy
        #define SERVO9_IO               dummy
        #define SERVO10_TRIS            dummy
        #define SERVO10_IO              dummy
        #define SERVO11_TRIS            dummy
        #define SERVO11_IO              dummy
```

```
        #define SERVO12_TRIS            dummy
        #define SERVO12_IO              dummy
        #define SERVO13_TRIS            dummy
        #define SERVO13_IO              dummy
        #define SERVO14_TRIS            dummy
        #define SERVO14_IO              dummy
        #define SERVO15_TRIS            dummy
        #define SERVO15_IO              dummy
        #define SERVO16_TRIS            dummy
        #define SERVO16_IO              dummy
        #define SERVO17_TRIS            dummy
        #define SERVO17_IO              dummy
        #define SERVO18_TRIS            dummy
        #define SERVO18_IO              dummy
        #define SERVO19_TRIS            dummy
        #define SERVO19_IO              dummy
        #define SERVO20_TRIS            dummy
        #define SERVO20_IO              dummy
        #define SERVO21_TRIS            dummy
        #define SERVO21_IO              dummy
        #define SERVO22_TRIS            dummy
        #define SERVO22_IO              dummy
        #define SERVO23_TRIS            dummy
        #define SERVO23_IO              dummy

        #define SERVO24_TRI             dummy
        #define SERVO24_IO              dummy
        #define SERVO25_TRI             dummy
        #define SERVO25_IO              dummy

#endif

#define UART2TX_TRIS                    (TRISFbits.TRISF5)
#define UART2TX_IO              (PORTFbits.RF5)
#define UART2RX_TRIS                    (TRISFbits.TRISF4)
#define UART2RX_IO              (PORTFbits.RF4)
#if defined(USE_UART1)
```

```
        #define UART1TX_TRIS                (TRISFbits.TRISF3)
        #define UART1TX_IO                  (PORTFbits.RF3)
        #define UART1RX_TRIS                (TRISFbits.TRISF2)
        #define UART1RX_IO                  (PORTFbits.RF2)
    #endif

#endif

    #define SONIC0_TRIS                     (TRISDbits.TRISD2)
    #define SONIC0_IO                       (PORTDbits.RD2)
    #define SONIC1_TRIS                     (TRISDbits.TRISD3)
    #define SONIC1_IO                       (PORTDbits.RD3)

    #define SONIC_INT0_TRIS                 (TRISDbits.TRISD8)
    #define SONIC_INT0_IO                   (PORTDbits.RD8)
    #define SONIC_INT1_TRIS                 (TRISDbits.TRISD9)
    #define SONIC_INT1_IO                   (PORTDbits.RD9)

    #define POSITION_SENSE_TRIS             (TRISBbits.TRISB0)
    #define POSITION_SENSE_IO               (PORTBbits.RB0)
    #define CURRENT_SENSE_TRIS              (TRISBbits.TRISB1)
    #define CURRENT_SENSE_IO                (PORTBbits.RB1)
    #define VOLTAGE_SENSE_TRIS              (TRISBbits.TRISB2)
    #define VOLTAGE_SENSE_IO                (PORTBbits.RB2)


    // Some A/D converter registers on dsPIC30s are named slightly differently
    // on other procesors, so we need to rename them.
    #define ADC1BUF0            ADCBUF0
    #define AD1CHS                  ADCHS
    #define AD1CON1                 ADCON1
    #define AD1CON2                 ADCON2
    #define AD1CON3                 ADCON3
    #define AD1PCFG                 ADPCFG
    #define AD1CSSL                 ADCSSL
    #define AD1IF               ADIF
    #define AD1IE               ADIE
```

```
        #define _ADC1Interrupt          _ADCInterrupt


#endif
```

UART.h

```
/******************************************************************************
 *
 *    UART access routines for C18 and C30
 *
 ******************************************************************************
 * FileName:        UART.h
 * Processor:       PIC18, PIC24F, PIC24H, dsPIC30F, dsPIC33F
 * Compiler:        Microchip C30 v3.01 or higher
 *                                      Microchip C18 v3.13 or higher
 *                                      HI-TECH PICC-18 STD 9.50PL3 or higher
 * Company:         Microchip Technology, Inc.
 *
 * Software License Agreement
 *
 * Copyright (C) 2002-2008 Microchip Technology Inc.  All rights
 * reserved.
 *
 * Microchip licenses to you the right to use, modify, copy, and
 * distribute:
 * (i)  the Software when embedded on a Microchip microcontroller or
 *      digital signal controller product ("Device") which is
 *      integrated into Licensee's product; or
 * (ii) ONLY the Software driver source files ENC28J60.c and
 *      ENC28J60.h ported to a non-Microchip device used in
 *      conjunction with a Microchip ethernet controller for the
 *      sole purpose of interfacing with the ethernet controller.
 *
 * You should refer to the license agreement accompanying this
 * Software for additional information regarding your rights and
 * obligations.
```

```
 *
 * THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT
 * WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT
 * LIMITATION, ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL
 * MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR
 * CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF
 * PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS
 * BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE
 * THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER
 * SIMILAR COSTS, WHETHER ASSERTED ON THE BASIS OF CONTRACT, TORT
 * (INCLUDING NEGLIGENCE), BREACH OF WARRANTY, OR OTHERWISE.
 *
 *
 * Author            Date                Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Howard Schlunder          6/16/06          Original
********************************************************************/
#ifndef __UART_H
#define __UART_H

#include "Compiler.h"
#include "HardwareProfile.h"




#if defined(__C30__) // PIC24F, PIC24H, dsPIC30, dsPIC33
        void putsUART2(unsigned int *buffer);
        unsigned int getsUART2(unsigned int length,unsigned int *buffer,unsigned int
uart_data_wait);
        char DataRdyUART2(void);
        char BusyUART2(void);
        unsigned int ReadUART2(void);
        void WriteUART2(unsigned int data);
```

```
        void putsUART1(unsigned int *buffer);
        unsigned int getsUART1(unsigned int length,unsigned int *buffer,unsigned int
uart_data_wait);
        char DataRdyUART1(void);
        char BusyUART1(void);
        unsigned int ReadUART1(void);
        void WriteUART1(unsigned int data);
#endif



#endif
```

WPIO.h

```
#include <string.h>
#include <stdlib.h>
#include "GenericTypeDefs.h"
#include "Compiler.h"
#include "HardwareProfile.h"
#include "UART.h"
#include "Delay.h"
#include <stdlib.h>

#define NUM_SERVOS 26
#define INIT_VALUE 140
#define Nop()    {__asm__ volatile ("nop");}
#define ClrWdt() {__asm__ volatile ("clrwdt");}
#define Sleep()  {__asm__ volatile ("pwrsav #0");}
#define Idle()   {__asm__ volatile ("pwrsav #1");}
#define StreamSizeIndex                 3
#define WASPheaderSize                  4
#define BAUD_RATE2                      (115200)
//#define BAUD_RATE1          (57600)          // bps
//#define BAUD_RATE1          (9600)           // bps
#define BAUD_RATE1          (115200)    // bps bluetooth baud
#define STACK_USE_UART
#define DECIMAL 10
#define TABLE_SIZE 200
```

```
void InitializeBoard(void);
void ProcessPacket(void);
void allOn(int);
void ServoOff(int);

int GetADC(int chan);
void InitADC(void);
void __attribute__((__interrupt__)) _ADCInterrupt(void);

void CheckPacket(void);
void TimeoutPacket(void);

void setServo(unsigned char number, double value);
void setAngle(unsigned char number, double value);
void waveForm(double * step_table, long step_index);
double adc_Val_to_mVoltage(int adc_value);
void itoa(unsigned int Value, char *Buffer);


//test adc
#define FOSC   7372800
#define PLL    16
#define FCY    FOSC*PLL/4

#define NUMSAMP        256
//NOTE: The actual sampling rate realized may be 7998.698 Hz
//     due to a small round off error. Ensure you provide the
//     true sampling rate to dsPICworks if you are trying to plot
//     the sampled or filtered signal.
#define SAMPLINGRATE   8000
#define SAMPCOUNT      (FCY/SAMPLINGRATE)+1
//end test

void UpdateRangeFinder(void);
```