ROBOTIC ENDOSCOPE

Jon K Edwards

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Biomedical Engineering at the School of Medicine.

Chapel Hill
2012

Approved By:

Robert Dennis, PhD

Nancy Allbritton, MD, PhD

Mark Tommerdahl, PhD

Jeffrey MacDonald, PhD

Shawn Gomez, PhD

Abstract

JON K EDWARDS: Robotic Endoscope
Under the Direction of Dr. Robert Dennis

The endoscope has become ubiquitous and indispensable, changing many surgical procedures from life-threatening to outpatient. Use of the endoscope is limited by the ability to safely navigate circuitous paths, provide a stable tip in situ, and generate force where and as needed. Our lab developed a prototype robotic endoscope which mitigates these limits and is able both to contort to follow a convolved path and to generate tip-force in any direction. This design can be mounted at the extracorporeal end of any control system, extending existing surgical robots' utility. The effectors' actuators (stepper motors in this prototype) are external to the effector and transmit force via cables (aka tendons), and, assuming nonferromagnetic robot-segment composition, the design is safe for use with MRI and X-ray. A hollow core allows for in situ tool exchange, and hollow wall channels allows for routing permanent tools to the effector tip (e.g., vacuum, saline, fiber optics).

To Cheryl

Acknowledgements

The author would like to thank Dr. Bob Dennis for his time, insight, and apt assistance, my committee for their generous and patient guidance, and Steven Emanuel for his help converting theory to practice.

Preface

I had atrial fibrillation surgery twice. The first time killed me, but I got better. Surgery entailed going between two ribs, pushing the lung aside, and burning arcs across the surface of my heart in hopes of interrupting ectopic pathways. My heart stopped and was restarted. They sewed me up, and then reopened me when internal hemorrhaging wouldn't stop. After a week in intensive care and several months as a semi-invalid, I was unimpressed.

The second surgery was endoscopic, took most of a day to perform and a day to recover from. The endoscope itself looked crude but performed perfectly. The surgery worked. I was inspired.

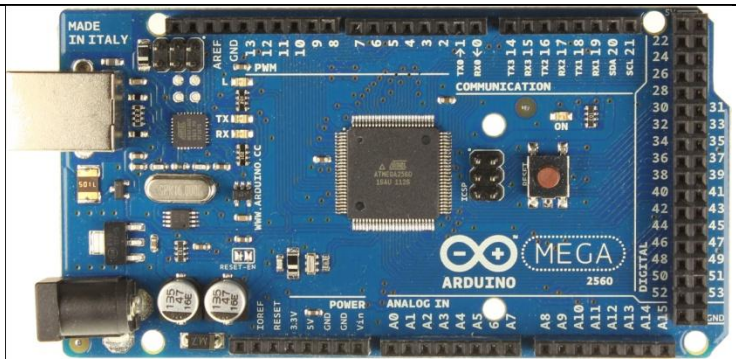
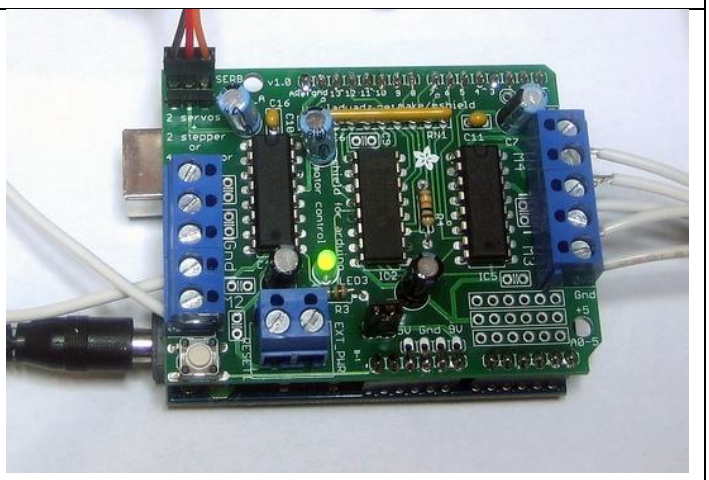
The goal to designing this prototypic endoscope was to develop a tool which would overcome the perceived shortcomings of existing designs. I wanted the endoscope to be able to navigate circuitous paths without using interstitial tissue to redirect the endoscope, to be able to generate force omnidirectionally at the tip and orthogonally along the shaft, to improve the surgeon's toolset.

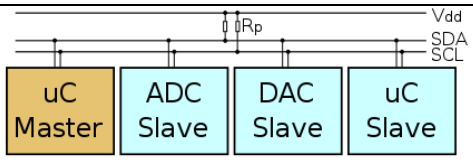
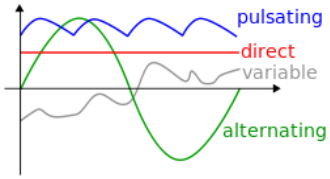
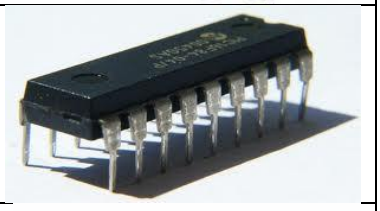
Table of Contents

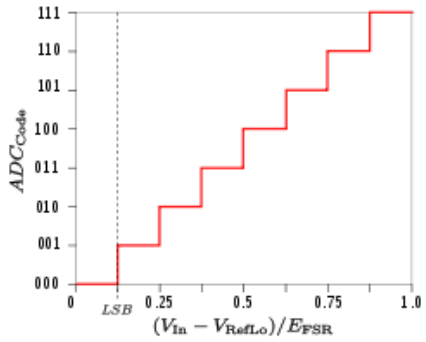
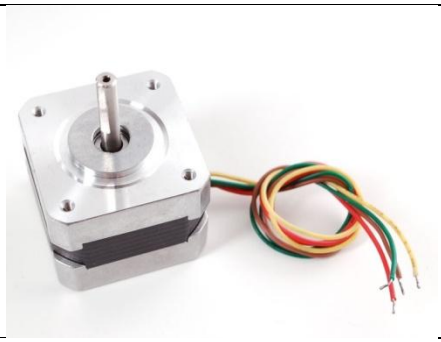
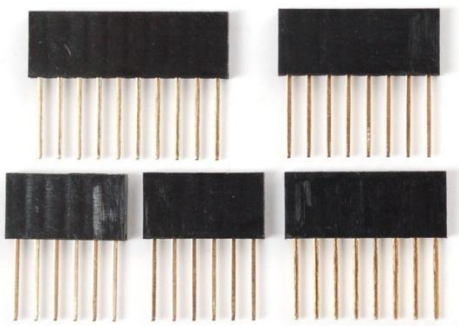
List of Tables

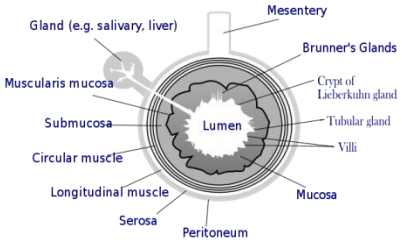
List of Figures

Terminology

2560 – Arduino 2560 is a Microcontroller, also known as a microcomputer, SBC (single board computer) or Embedded controller. Arduino is a manufacturer	
328p – Arduino 328p is a less powerful version of the 2560	
Daughter board - an extension of the motherboard by pins and sockets	
Motor shield – A 'daughter board' that plugs into the sockets along the edges of the Arduino 328p and is controlled by TTL signals from the 328p. The output of this board is stepper-motor control or a variety of DC motor applications, depending on how the 328p is programmed	
Flash – type of nonvolatile memory	Faster than EEPROM, slower than RAM
EEPROM – electrically erasable programmable read-only memory. This is nonvolatile	much slower than RAM; provides a means to store nonvolatile data (eg, operating system configuration data, or, in the case of the snakebot, axes' configurations for

computer memory	'claymation' motion)
USB – Universal Serial Bus a serial communications protocol	
USB hub – a tool for the distribution of USB signals. One input is distributed to many outputs. Responding signals from any of the 'output' plugs are routed to the 'input' plug. Functionally this is a multiplexer (mux).	
I2C – Inter Integrated Circuit also known as TWI (two wire Interface) is a low-speed communication protocol	
TTL – transistor to transistor logic	A logic false is represented by 0VDC, a true by +5VDC
DC – direct current AC – alternating current	
Claymation – a form of stop-motion animation: objects (eg a snakebot) are moved incrementally, then the configuration 'frozen' (eg, with a camera or by writing configuration data to memory)	
IC – integrated circuit	
IC socket – a device for physically holding an IC on a circuit board and electrically connecting the IC to the circuit	

CAD – computer aided design Use of computer software to simulate an object before it is physically produced in hopes of an efficient and correct design process	
Cables – aka tendons, strings, wires	
ASCII - American Standard Code for Information Interchange	A protocol for digitally encoding alphanumeric data
A/D – analog to digital converter	A tool for approximating analog inputs for digital processing 
Stepper motor – a DC controlled motor where the shaft position is controlled by changing applied DC voltage. Since the shaft moves in discrete increments, no encoder is necessary to know shaft radial position	
Stacking Header – this computer-board hardware allows the electrical and physical connection of two printed-circuit-boards. Once soldered to the first board, the pins are pressed into the female socket of a similar header.	

PVC - Poly Vinyl Chloride. a type of plastic from which plumbing pipes are commonly fabricated	
LESS – Laporoendoscopic Single-Site Surgery – endoscopic surgery performed using a single perforation in the patient's body.	
Lumen – the inside space of a structure	
Cannula – latin for 'little reed', a hollow tube for introducing foreign material to the body, taking samples, or creating a portal through which to access internal material	

Chapter 1

Background

Endoscope design is an intersection of many disciplines, including robot design, computer science, material science, human-machine interface (HID), all joining to meet the need of safely navigating the patient's body and performing the task the surgeon requires once in position. Whether surgical access is through existing orifices, as with NOTES, or by small incisions, as with LESS and MIS and RSP (1; 2), the goal is to provide the surgeon a stable platform within the patient from which to cut, sew, and grasp, all while clearly imaging patient tissues. This design is for that portion of the tool between the extracorporeal positioning structures (robot arms) and the end-effector (gripper, scalpel, saline, suction, etc.) and can be adapted to any hierarchal positioning mechanism; issues of extracorporeal routing of the structures supporting this effector around the patient's body and whether the robot is to be the surgeon's slave or collaborator are not addressed here.

Endoscopes allow the surgeon to navigate parts of the circulatory system and much of the body, performing many surgeries more quickly, safely, and with better outcome than traditional methods. Too flexible an endoscope cannot offer sufficient stability to perform cutting and grasping, while too stiff an endoscope cannot navigate safely (3; 4). Some flexible endoscopes navigate by curling the tip of the scope and twisting the base to orient towards the goal, then pushing the scope forward. This works as long as the forward force at the base is redirected along the shaft of the endoscope by the tissue of the patient towards the desired target, but can damage the patient if the route requires too sharp a turn or impinges delicate tissues. Other

designs incorporate multiple controllable segments (continuum manipulator endoscopes), but still rely on interstitial tissue to redirect force to the tip (5).

Our prototype design is derived from a cannula (6), generating force and motion at the tip without requiring force against interstitial tissue. Inspiration for the design came from study of a human hand (7), which also utilizes remote actuators (muscles). Force at the snakebot tip is transferred to the base through robot-segment compression and cable tension. The snakebot can use interstitial tissue to push off against, just as is done with conventional endoscope navigation, or it can conform to lumen and create minimal tissue stress. With the former paradigm, the tip flex is reduced and tip-force application is made more deterministic.

This snakebot endoscope is a delivery system for surgeons' tools, with a hollow core through which scalpels, grippers, or cameras can be exchanged in situ. The robot segments' wall thickness contains channels or vias which can facilitate routing in the walls of fiber optics for illumination and endoscopic-tip-vision (8), delivery of saline, and vacuum lines, all without compromising surgical-implement-payload. This concept of a snakebot is neither new nor unique, being but a series of universal joints, but this design improves on previous implementations by maintaining a lumen though which tools can be interchanged and from its use of cabling to replace hinges and pivots. (9; 10; 11; 12)

The snakebot can self-propel through the patient's body, producing motive force not just from the endoscope base being manipulated but by serpentine motion of the snakebot's body. Clearly it can be dangerous to have a tool forcing itself through a patient and perhaps perforating an organ, so the robot is equipped with force sensors and would proceed under the surgeon's direct control. Path planning becomes nontrivial as both modes of propulsion (serpentine and base-generated) must consider potential interstitial damage.

Snakelike robot designs vary widely (13), ranging from climbing robots (14) to serpentine rolling robots (15) to inspection robots (16) to 'elephant trunks (17)'. Actuator type and placement vary as well, from external pneumatics (18; 19) to internal (to the snakebot) motor

or memory metal (20). The design described here is a puppet, controlled by cables (tendons) attached to robot segments at one end and to a stepper motor at the other. The advantages are the motors' ferromagnetic parts are as remote as is necessary to allow the effector to be used in MRI or Xray machinery, and the effector can be as small as necessary to do the surgery and still generate significant force at the tip.

Chapter 2

History and State of the Art

Endoscope history is over four millennia long, with Egyptian an papyrus referring to endoscope use in 2460BCE (21). Modern endoscope history begins in 1806 with Bozzini's lichleiter or 'Light Conductor' (22). The first internal lights were used by Charles David in 1908. Endoscopes were used to diagnose liver and gallbladder disease by Hienz Kalk in the 1930's and by Raoul Palmer in 1944 to perform gynecologic surgery. Commercial flexible gastroscopes were available in 1911 and semi-flexible gastroscopes in 1930, but real progress in endoscope use and design began when, in 1960, Storz marketed a 'cold light', an extracorporeal light source that routed illumination without heat into the body.

In combination with the medical optics of Harold Hopkins, the endoscope became a viable tool. A camera at the gastroscope tip was developed in 1950 by Sugiura and was used to diagnose stomach ulcers. Fiber optics were introduced in the 1950's by Hopkins but were unreliable and low-resolution until Hopkins and Storz optimized the optics, an evolution that enabled modern key-hole surgeries and earned Hopkins the Rumford Medal in 1984.

Endoscope use changed with the inclusion of robotics, which never tire, do not tremble, and can report applied forces accurately. The level of automation integration varies with platform, from a generic holder attached to the same endoscope a human would use (23) to highly integrated master-slave systems where the surgeon interfaces a remote display and may not even be in the same city as the patient (24).

The design of hyper-redundant robots and their close relative, the continuum robot, also has a long history. Implementation of a 'Tensor Arm Manipulator' was developed by Anderson in 1967 (25; 26), which demonstrated the utility of hyper-redundant puppet robot design. Continuum robots share the tendon-actuation with this design, but rely on a flexible core for force redirection and, having fewer cables, are not as inherently compliant (27).

Current research is focused on developing capsule endoscopes, surgeon-interface refinements, and types of extracorporeal robot positioning systems. The design of the portion of the endoscope actually in situ varies from rigid to flexible (28), and of varying degrees of automation, depending on the system and application. Designs similar to this prototype are also being developed (29; 30), but none combines the simplicity of segment design with the maneuverability offered by individually controlled axes.

Chapter 3

Methods to Design the Snakebot

This prototype snakebot was developed using a top-down approach of 'what should the tool accomplish' and iterated across the potential solution-space to refine the design. Design criteria included .

1. maximize the 'payload' of the snakebot (the lumen of the snakebot through which tools might be inserted with the snakebot in situ)
2. provide a stable platform for tip-manipulation of payload-borne tools (scalpels, grippers, optics, etc.)
3. be able to generate tip-force omnidirectionally and deterministically
4. be strong enough to sustain the rigors of surgery
5. be flexible enough to navigate three-dimensional curved paths
6. be small enough to minimize unnecessary incidental damage incurred by navigating the patient's body
7. be fabricated from non ferromagnetic material so can be used in an MRI

The design effort was intended to produce the snakebot actuator, that part of an endoscopic system between the endoscope's extracorporeal (to the patient) positioning system and the tools at the tip. Demonstration of the capability of the snakebot required design of ancillary systems, including force sensors, actuators, and a rudimentary extracorporeal positioning system.

Actuators

Various actuators were considered, including stepper motors, DC motors, memory metal, artificial muscles, and inflatable muscles. Actuators integral to the snakebot's body were deemed impractical for three reasons:

1. a motor small enough to fit within a narrow endoscope would be too weak to be useful.
2. if a motor strong enough to meet the needs of a snakebot existed it would require wiring for power, control, and encoder data, wiring which would require at least as much of the endoscope's volume as would tendons.
3. Any ferromagnetic content would render the endoscope incapable of working in an MRI, a feature which cable actuation preserved by removing motors from the endoscope body.

These premises made clear the design would incorporate tendons (a.k.a. cables or strings) to transfer force from a motor to the segment. Low cost and ease-of-implementation refined the actuator choice to stepper motors.

Tensegrity

Drawing inspiration from Buckminster Fuller, the first attempt used tensegrity structures to construct a robust controllable endoscope.

While this design might be made to work and would be strong, deterministic, and flexible, it would be difficult to implement kinematics and controls. For example, to bend the tensegrity would require simultaneous

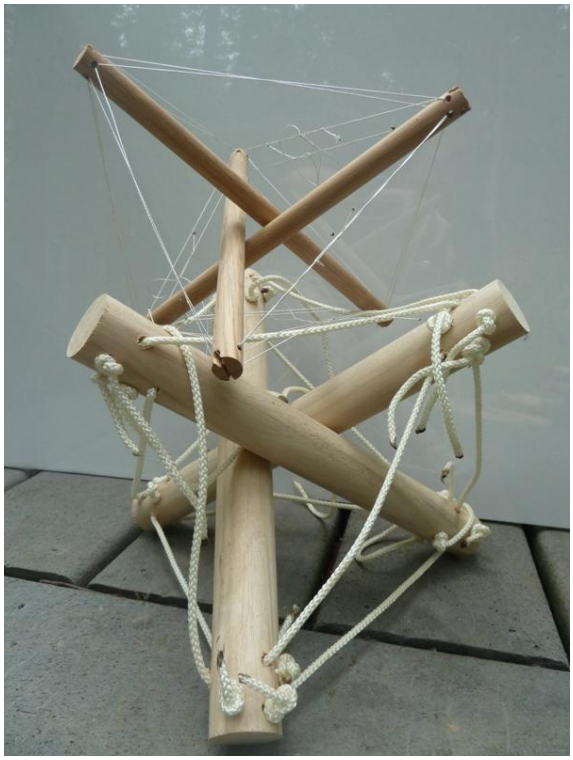


Figure 1 - Tensegrity Structure

adjustment of all twelve cables per segment. Furthermore, such a design would have protruding edges which would inevitably catch on interstitial tissues. This approach was abandoned.

Stacked Segments

The following figure shows a first gesture towards narrowing the range of potential designs and showing the utility of cables-as-controls. This plywood-and-wood-ball prototype shows the alternating orthogonal cams and pivoting platforms which will be implemented in the final prototype. The routing of cables through common vias and the use of a single cable per platform (routed from platform edge to the base and back to the opposite edge of the same platform) is also a feature of the final prototype.

Figure 2 - Wooden Prototype of Stacked Segments

Experimenting with this prototype demonstrated how moving proximal segment (one closer to the base) would change the orientation of all distal segments which share that degree-of-freedom (ie, all segments which have their pivots on the same plane). This issue becomes more urgent as more segments are added.

Segment Naming Paradigm

Segment designs are named to describe the features permuted for that particular instance. Permutation parameters include placement of holes for routing of cables (a.k.a. 'vias'), symmetry of the segment, cam placement, and pivot surface type. These parameters are described by single letters and appear in a consistent order to make parsing the name simple, as shown in the following table.

1st – Cam	2nd - Shape	3rd - Type	4th	5th - Cable Route	6th - Point Type
o – offset	s – symmetric	c – cap	Iteration #	H1 – holes at points	Pn – unrounded
c – centered	a - asymmetric	b – base		H2 – holes offset 45	Pa – cut unrounded
		s - segment			Rn – hemicylinder
					Fn - flattened

Table 1 - Snakebot Segment Naming Paradigm

Abandoned Designs

Having established the general design of the endoscope as a sequence of stacked segments, an exploration of possible segment shapes ensued. There were thirty-seven permutations on the stacked-segment theme that were tried and abandoned for a variety of reasons. Segment aspects that were permuted include wall thickness, segment diameter, segment length, and a variety of shapes for the interface between segments. The inventory of these permutations appears in the Appendix of Segment-Shape Solution Space, and a synopsis follows.

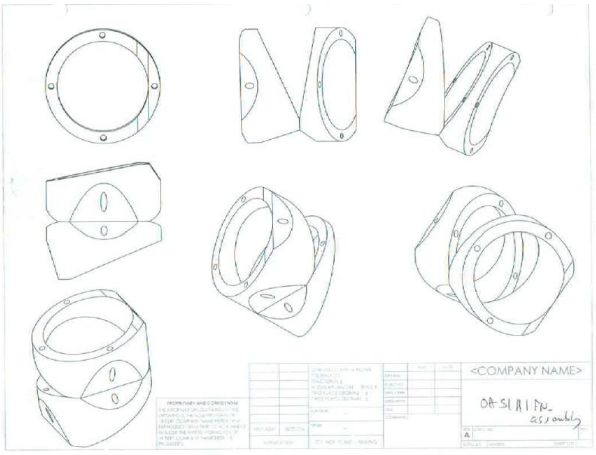


Figure 3 - OAS1H1FN - asymmetric cam assembly

Asymmetrical Segment

The first permutation direction

abandoned was the nonsymmetric cam (eg the OAS1H1FN segment). These were envisioned to enable the snakebot to flex more in one direction than the other, so it might navigate particularly sharp corners. In practice, it required too much force to move an axis so designed and the transition from applying tension to motion too abrupt. Also, for such a segment pair to move, the cables to adjacent axes would have to both elongate, requiring two motors per segment and an unwieldy control paradigm.

Symmetric Segment with Offset Bearings

A variant on the asymmetric cam is a symmetric cam with asymmetric rotation faces. This had the same shortcomings of the asymmetric cam.

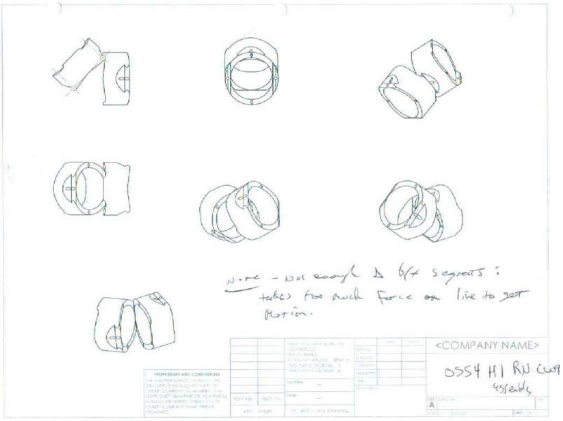


Figure 4 - OSS4H1RN –
Symmetric Segment with Offset Bearings

Asymmetric Segments with Centered Bearings

Asymmetry had the clear problem of requiring complex control-cable manipulation of all segments from that which was being moved to the tip of the snakebot. It was not yet clear that this issue could be resolved only by routing the control cables through the axis of rotation between segments, so the next permutation attempted continued with the offset cable routing but attempted a symmetric bearing configuration.

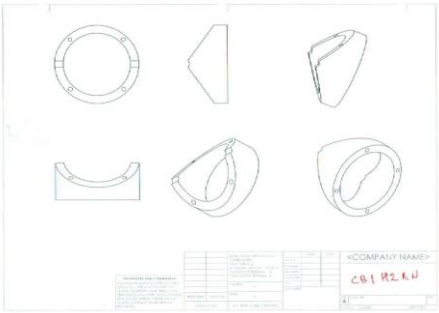


Figure 5 - CB1H2RN - Asymmetric Segments with Centered Bearing - Female

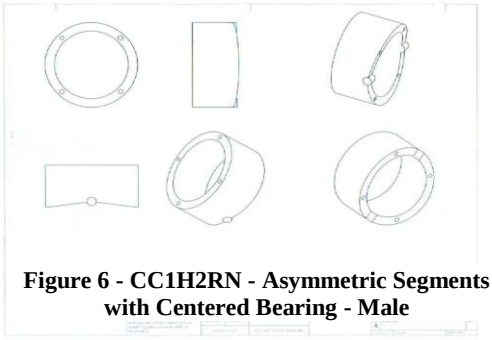


Figure 6 - CC1H2RN - Asymmetric Segments with Centered Bearing - Male

Symmetric Segment with Orthogonal Cams

Having explored a variety of designs and found the asymmetrical paradigm infeasible, a line of symmetric orthogonal cams was designed.

Symmetric Segment with Small Cam

The next permutation line of inquiry abandoned involved forming a bearing surface between adjacent cams. For example, segments CC1H2RN and CB1H2RN form a mating pair, and multiple-segment CS1H2RN forms a segment set.

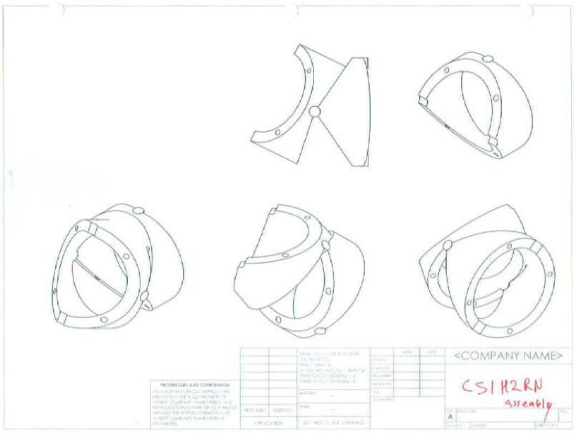


Figure 7 - CS1H2RN – Symmetric Orthogonal Cams with Small Bearing Surface

This approach worked but was difficult to control. The small bearing surface between segments created a high-friction point such that the transition from statically applied tension and segment motion was jerky and nondeterministic, making incremental change impossible.

The next version of this design (and the end of the cumbersome labeling paradigm which preceded it) was labeled Version 1.

Version 1 - Flat Faced Cam

By increasing the cam size, the next evolution became segment CS2H2FN. This design worked better than all previous, and the flat surface between segments made axial alignment easy, but the transition from applied cable-tension to motion was abrupt, which

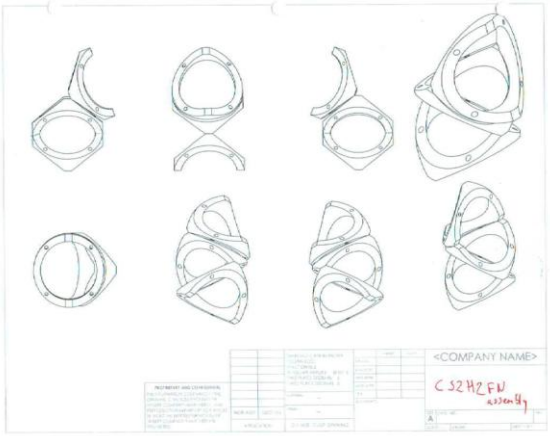


Figure 8- Version 1 - Segment CS2H2FN Assembly

presented a problem.

Version 4 - Hemicylindrical Cams

By rounding the flat section of adjacent cam-faces of a segment, a smooth transition across the entire range of motion was achieved.

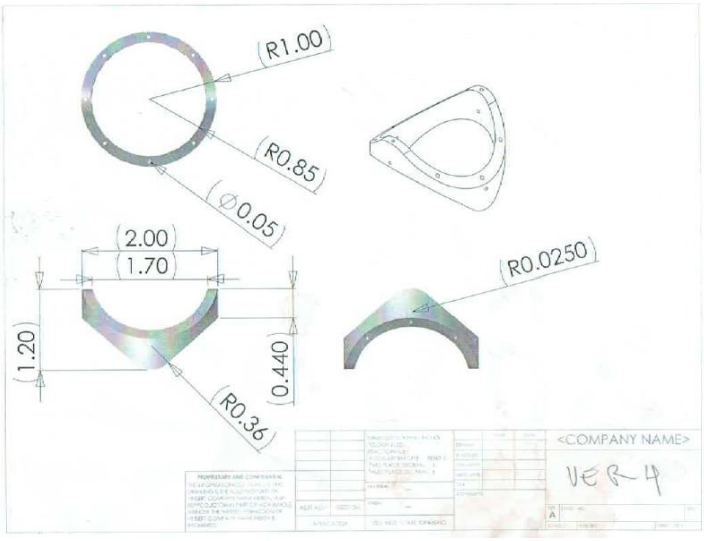


Figure 9 - Version 4 – Orthogonal Hemicylindrical Cams

This implementation worked, but the control cable routing was problematic. If routed through the center of the cam-face, the cables would limit cam-travel and, when adjacent cams were not axially aligned, caused tension in distal segments' control cables. The cable routing through the apex of the cam did keep adjacent segments aligned, an advantage worth retaining.

Note the segment design includes eight vias, four aligned with a cam apex and four offset from the cam apex by 45-degrees. If the control cables were routed through the vias' 45-degrees offset from the cam apex, this removed the effect of keeping adjacent segments aligned, allowing segments to become misaligned and the snakebot to collapse.

Version 5 - Y-Shaped Control Cable Via

The solution was to route the control cable through the cam apex through a via shaped to keep cable-length of distal segment's control cables constant throughout the range of motion. The first attempt at this solution is Version 5 of the orthogonal-cam line.

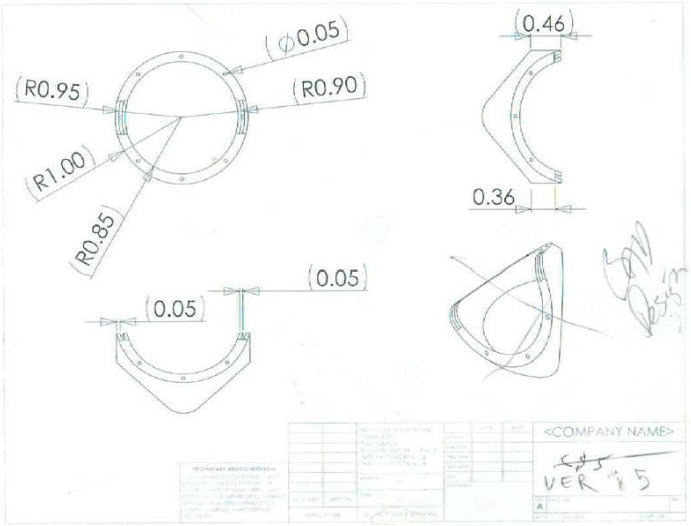


Figure 10 - Version 5

This design proved to be difficult to manufacture and prone to misalignment issues. The former problem is an artifact of the process used to produce segments, Force Deposition Modeling (FDM), which is the process of laying down successive layers of thin lines of plastic in a matrix. To create the gap through which control cables route through the cam's apex, a purportedly removable plastic was used to fill the gap to support the permanent plastic that was 'printed' around it. Removing the thin layer of placeholder plastic proved a challenge.

The control cable via was designed with a 'Y' profile so the total cable length to distal segments remained constant when the segment was moved. This design was much better than that of Version 4, and the Y-profile-channel feature was kept in future designs, but there was still some cable-tension hysteresis between the configurations of vertically-aligned cams and angularly-displaced cams.

Once assembled, a snakebot of these segments quickly misaligned: the shoulders of one segment's Y-channel groove fit into the Y-channel of the adjacent segment, so a very slight axial displacement of one segment resulted in the snakebot collapsing. The design was envisioned with the control cables acting to keep adjacent segments aligned, but this did not work since the

Y-channel was designed for maximum control-cable bundle diameter. At the proximal end of the snakebot, with control cables for all segments running through the via, the system remained aligned, but at more distal segment axes, with fewer control cables present in the via, more off-axis motion was possible and the segments became misaligned. Before misaligning, however, the design did show the utility of routing cables through the cam apex and of a shaped via to keep constant the lengths of distal segments' cables.

Version 6 - Surface Control Cable Vias

The next iteration of the orthogonal-cams line is Version 6, which routed control cables through surface vias

This design worked but not well. The design of the vias on alternating segment-faces was intended to create a 'weave' of the control cable surfaces such that the segments would be held in place in the snakebot while allowing adjacent segments to move. In practice, when adjacent segments were moved, the control cables tended to pop out of the via. Furthermore, there was enough lateral motion between adjacent segments to allow these segments to become misaligned, allowing the snakebot to collapse.

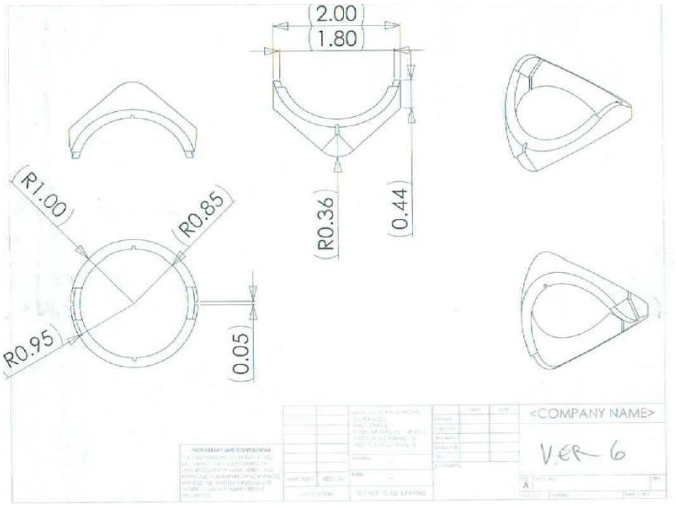


Figure 11 - Version 6

To attempt to keep adjacent segments aligned, an additional set of cables was added. These construction cables were routed across the cam surface to act as hinges while keeping

adjacent segments correctly aligned. Grooves were created in the cam face to route the construction cables while keeping them from being damaged by the cam faces.

Version 7 - Construction Cables

The addition of the construction cable did help keep segments aligned, but was not a complete success: segments still came unaligned. Versions 7, 8, and 9 all explored permutations of construction cable via design in attempts to keep the segments aligned. All worked poorly since the construction cable tended to align with the stresses that occurred between segments rather than where the vias were inset to the segment, and adapting the via to follow the lines of force was iterative. In addition, the control cables tended to stay within the Y-channel vias as was designed, but would occasionally come out, particularly at high degrees of deflection of adjacent segments.

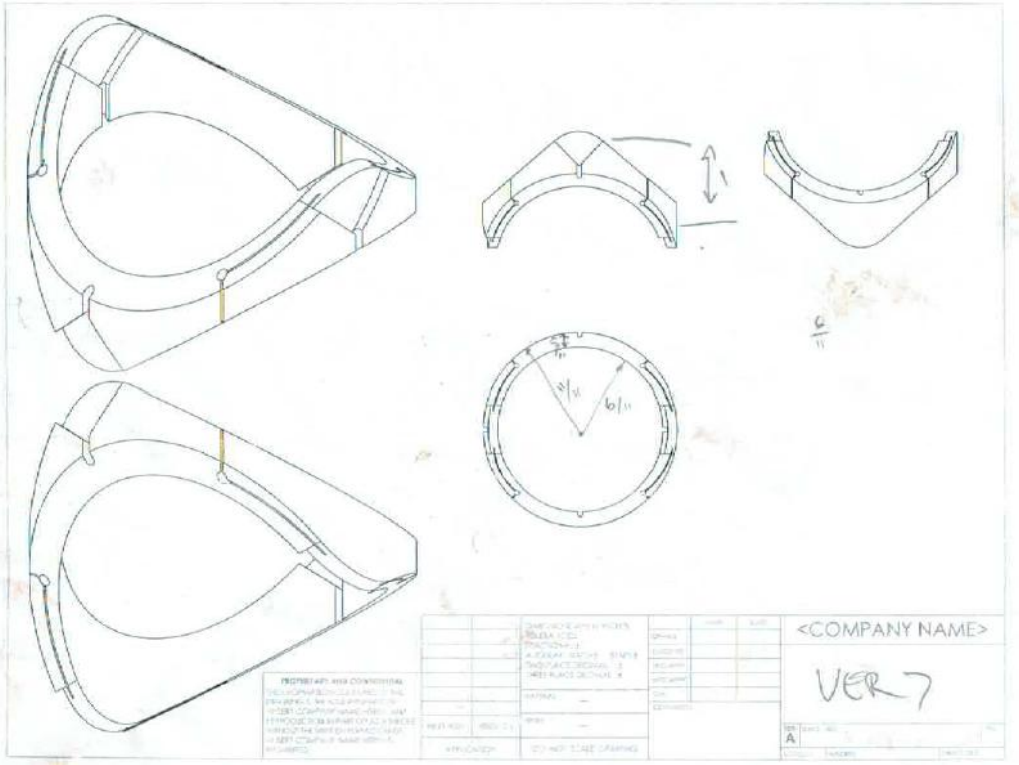


Figure 12 - Version 7

Version 10 - Lidded Control Cable Via

To keep the control cables in the vias, a 'lid' was added to Version 10. This lid kept the control cables in place, but the cam surface was too thin to keep adjacent segments aligned. One of the design criteria was to maximize the 'payload' of the snakebot, the lumen of the snakebot through which tools might be inserted in situ. Through Version 10, the payload was 90% of the snake's cross section. Despite the addition of the construction cables, adjacent segments still became misaligned. To address this shortcoming, the wall thickness was increased in Version 11, which reduced the payload but resolved the misalignment problem.

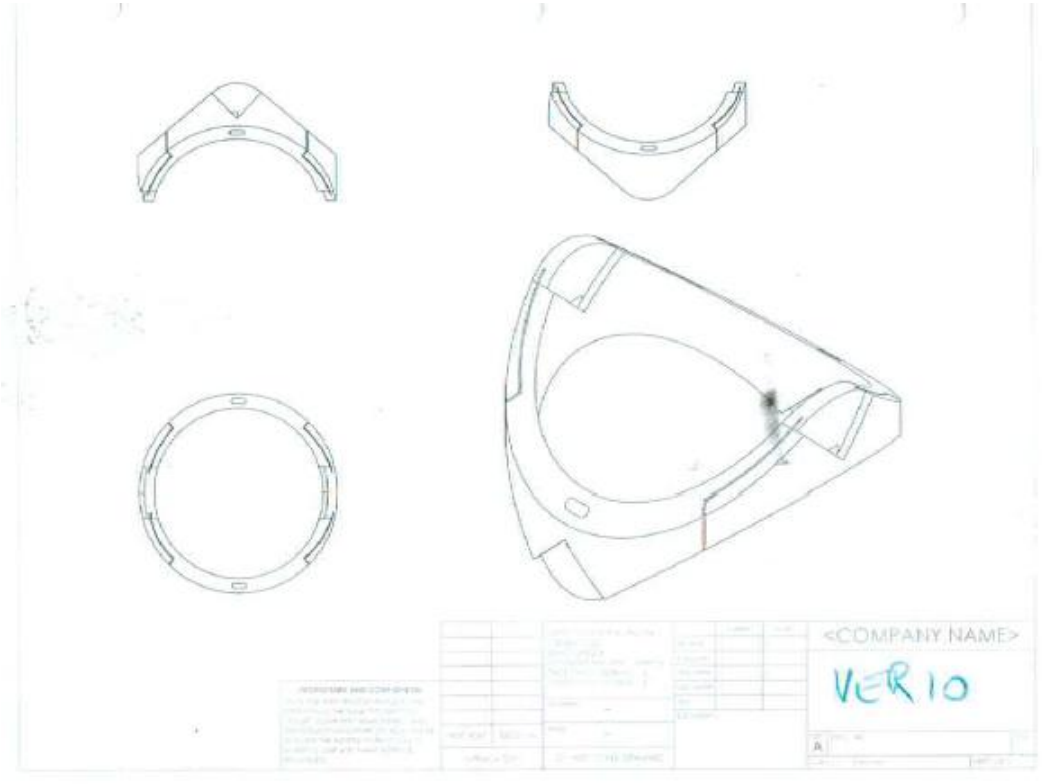


Figure 13 - Version 10

Version 11 -Thicker Segment Walls

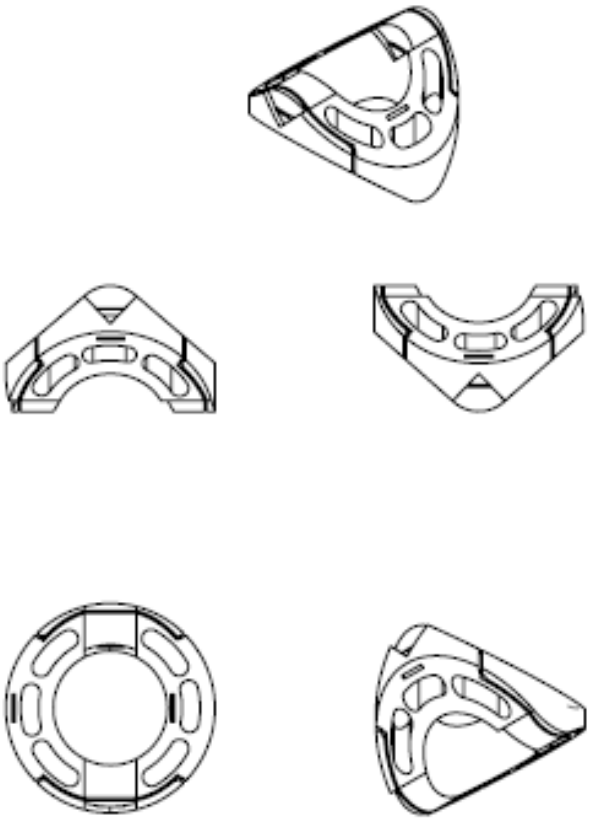


Figure 14 - Version 11

The cam surface was now thick enough to keep adjacent cams aligned throughout the range of travel of adjacent segments. The hollowing of segment walls was done initially to reduce material usage, but serendipitously these vias allow routing of permanent tools such as saline, vacuum lines, and fiber optic cables for lights and vision.

Production of these segments was problematic, however, due to the thin dimension of the control cable via. This narrow via also made assembly of the snakebot a challenge, an issue which, should this prototype ever be shrunk to production size, would become a serious obstacle.

Version 12 – Enlarged Control-Cable Via

Version 12 addressed the shortcomings of Version 11, combining the narrow control-cable-via with a material-saving via to create a larger channel. Also, fillets to via channel transitions were added to remove potential snag points.

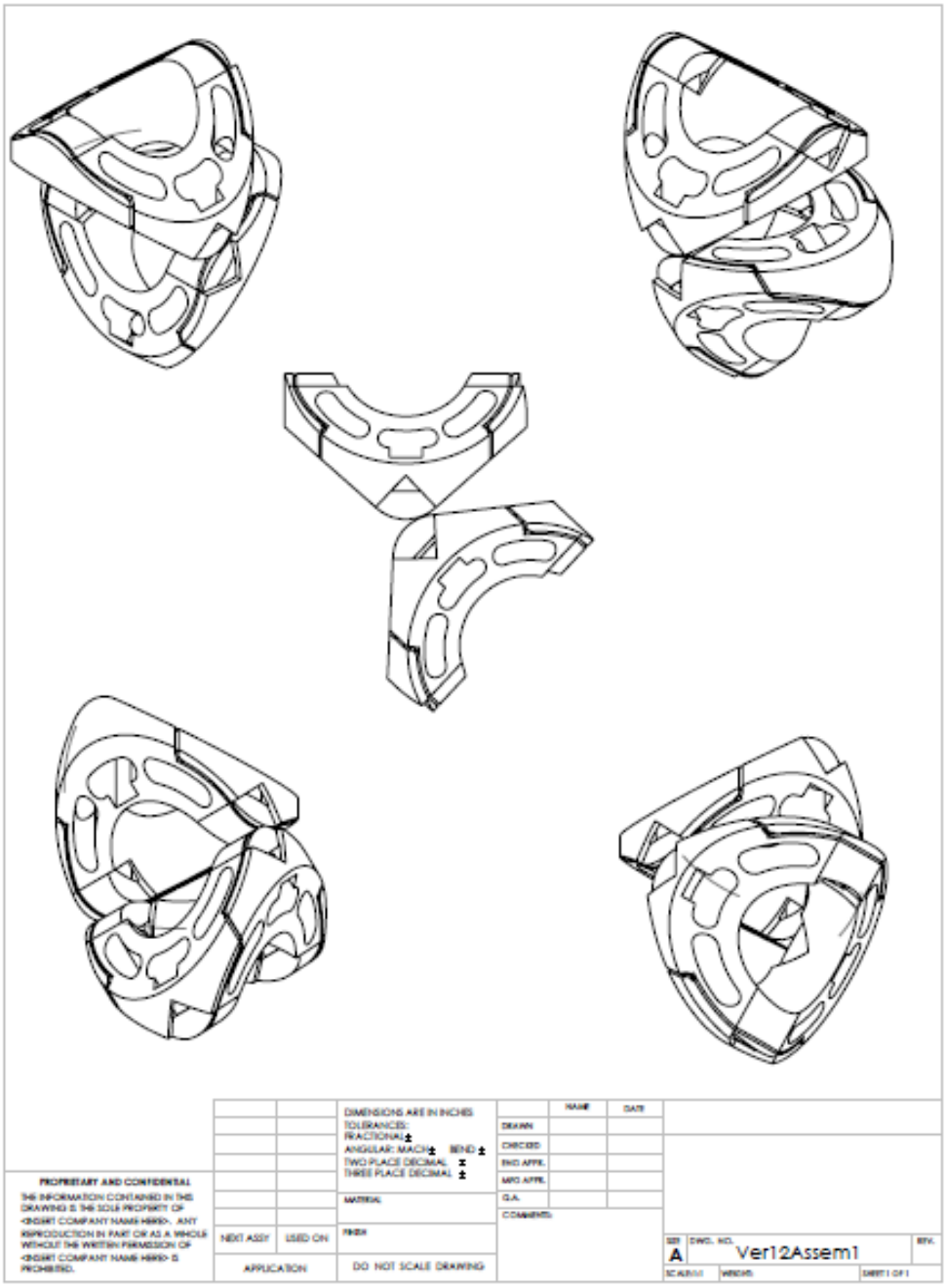


Figure 15 - Version 12 Assembly

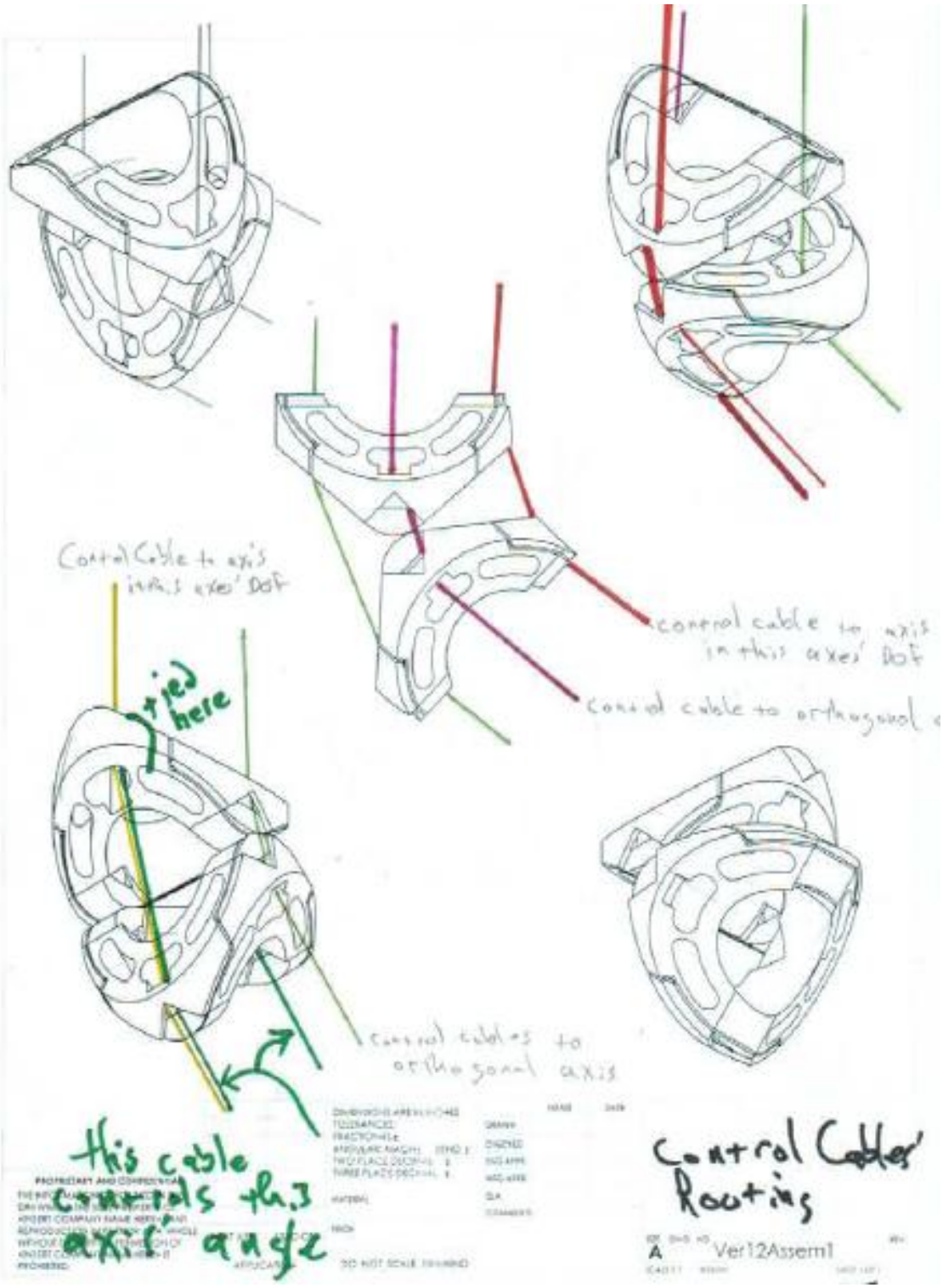


Figure 16 - Version 12 - Control Cable Routing

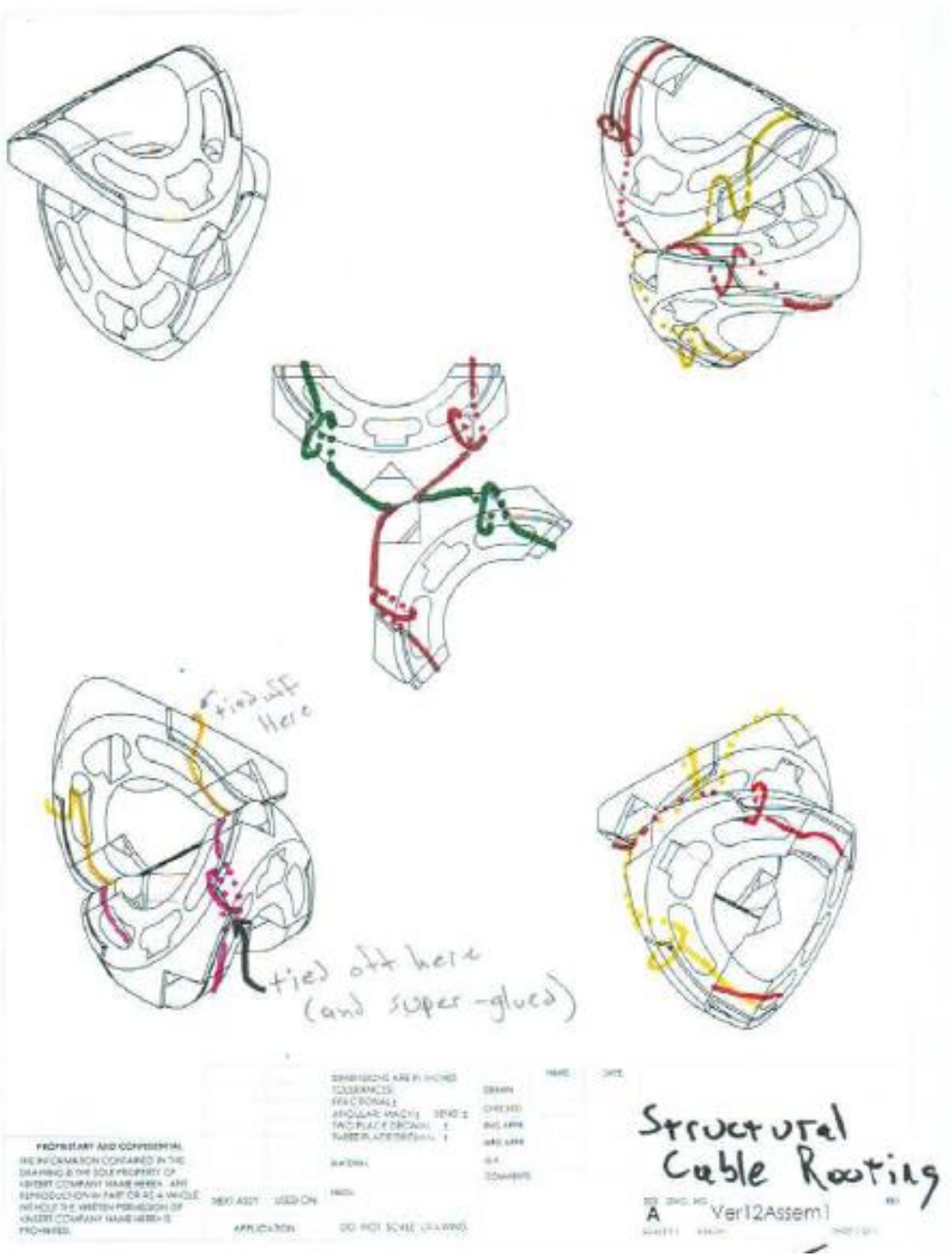


Figure 17 - Version 12 - Structural Cable Routing

Segment Dimension Solution-Space

Cam Radius

The radius of the segment's cam determines how sharp a bend the snakebot can navigate. In the prototype, the radius of the hemicylindrical cam is less than the radius of the snakebot segment, but this dimension was arbitrary, a function of the process of exploring the segment-dimension solution space.

In the figure below, note that the configuration in which the radius of the cam is greater than that of the segment body becomes uncontrollable at angles where the cam faces are no longer in contact.

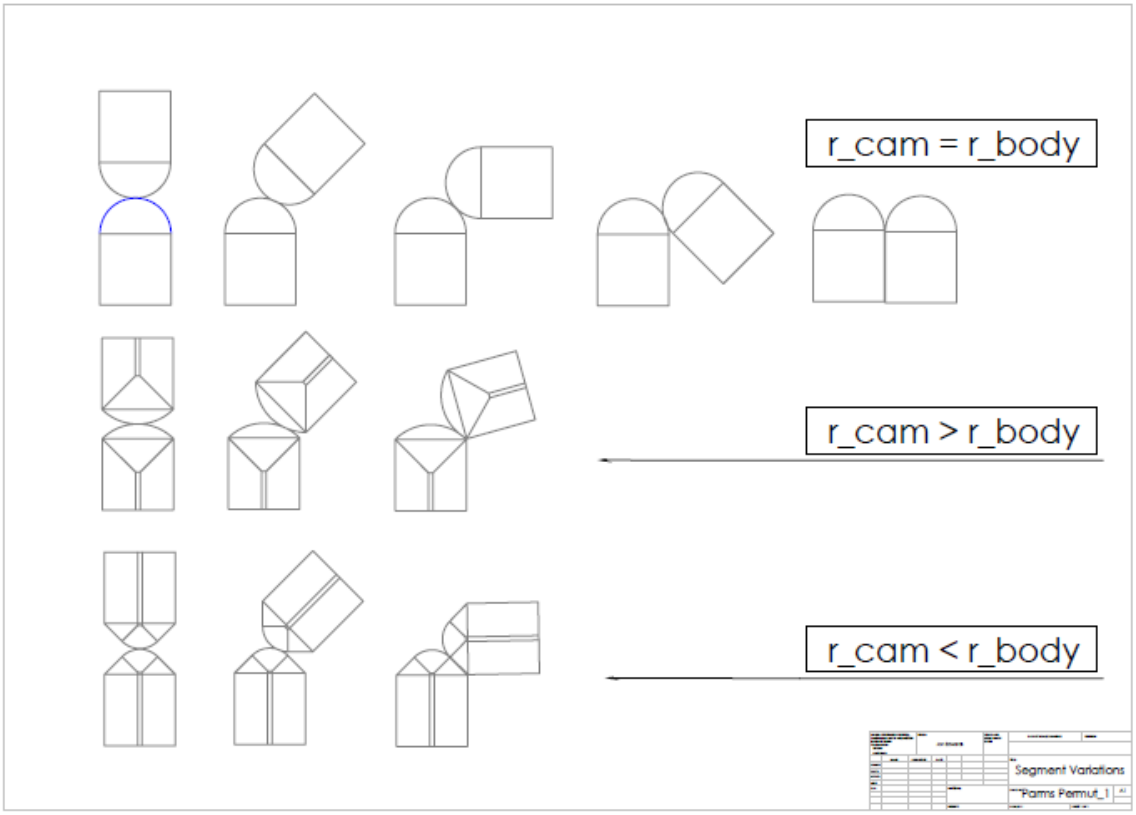


Figure 18 - Varying Cam Radius (Shoulders at 45-degrees)

As the cam radius is increased, so must the control-cable via be lowered (for a given shoulder-angle), placing a physical limit on minimum segment length (ie, the segment has to be long enough to hold the control-cable via). In the case where the Cam Radius equals the Segment Radius, the control cable via is merely a radial hole through the segment body located at the center of the cam's circle.

One problem with the configuration of the Cam-Radius equal to the Segment-Radius was the routing of the control cables at large angles-of-deflection: these cables crossed the payload space, either impacting and potentially crushing the payload or creating a cable-length-hysteresis wherein the total cable length varies with deflection-angle.

A smaller cam radius recreated the failures of segment CS1H2RN where the transition between applied tension and motion was difficult to control, making motions jerky. A larger cam radius increases the controllability and sensitivity of the snakebot.

Cam-Shoulder Angle

Another degree-of-freedom in the cam-dimension solution space is the angle of the cam 'shoulders', those flat surfaces adjoining the cam on segments where the radius of the cam is less than the radius of the segment body. Note it is this shoulder angle that determines the bend radius the snakebot can navigate, with larger shoulder angles generating snakebots that cannot navigate smaller-circumference paths.

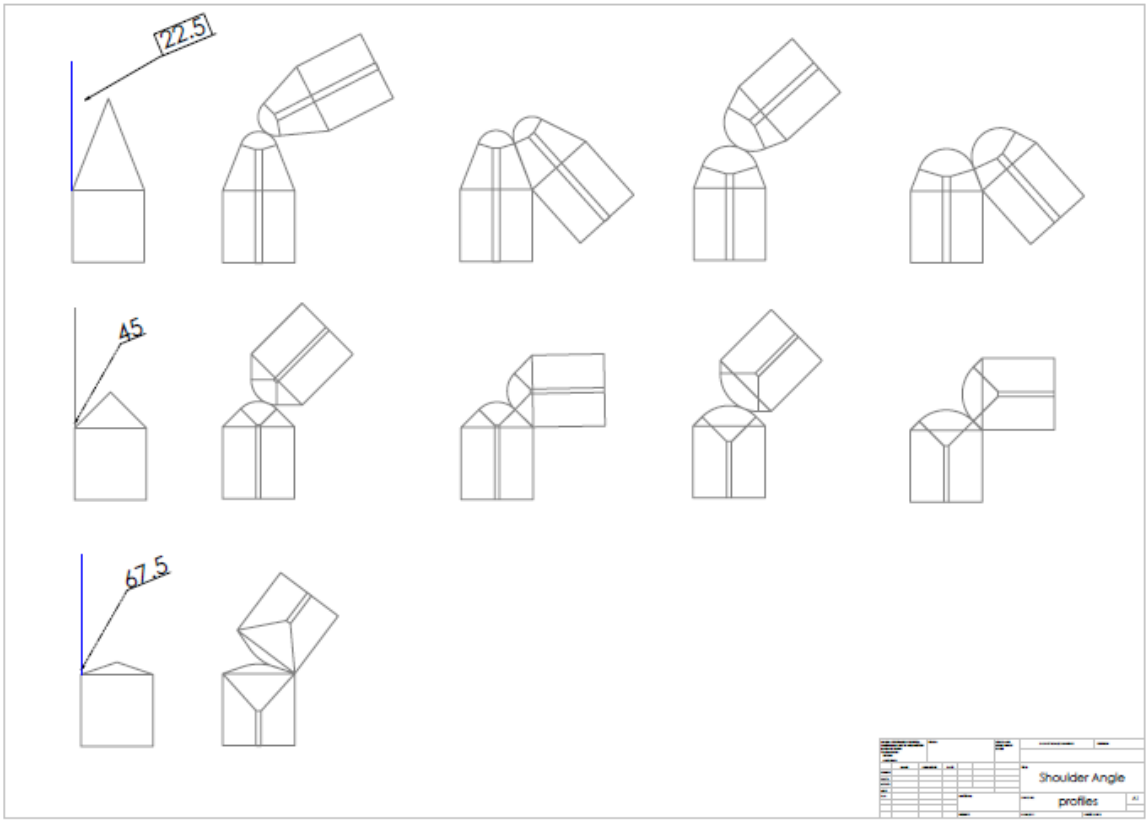


Figure 19 - Cam Shoulder Angles

Segment Length

The minimum bend radius is proportional to the segment length. Increased segment length increased the lever-arm of each segment-joint, which increased cable tension per unit tip-force.

Segment Diameter

The prototype has a diameter of two inches. This was chosen because segments of smaller diameters are physically difficult to manufacture and assemble into a snakebot, and because the FDM material being used tended to break with smaller prototypes. When segment radius is increased, so does lateral stability and strength, but the minimum bend radius unfortunately increases as well, decreasing snakebot capability to navigate sharp bends.

Segment Evolution Results

The process of exploring the solution space was heuristic, evolutionary, and reactionary: permutations were proposed, simulated, produced, and improvements noted and used as the basis for the next iteration.

Snakebot dimensions can be customized for a given application, with a tradeoff of flexibility and complexity. Segments need not all be of identical dimension, so the tip can be composed of shorter, more dexterous segments while the body is composed of longer, more stable segments.

Permutations of parameters and their results appear in the following table.

Cam radius	R_cam > R_segment: bad design. Uncontrollable past angle_of_deflection > included_angle_of_ cam_surface. Also requires long control-cable vias	R_cam = R_segment: Good and bad: max snakebot flexibility, but control cables cross payload-space at large angle-of-deflection	R_Cam < R_segment: Good. Mandates use of 'cam shoulder', which controls min snakebot-navigable radius, which keeps control cables out of payload
Cam shoulder angle	Angle < 45-degrees: Good and bad: smaller minimum-radius-navigable, but control cables cross payload	Angle = 45-degrees: Good compromise: Control cables only impinge payload slightly, while snakebot can still navigate circle where R_circle >= (2.5 * R_segment)	Angle > 45-degrees: Bad: payload is unimpinged, but snakebot loses navigability and cannot navigate sharp corners
Segment length	L_segment < R_segment: Good and bad: many parts to create a long snakebot, with commensurate complexity to construction and controls, but very maneuverable. Optimal for tip of snakebot	L_segment = R_segment: Good compromise. Perhaps optimal for mid-snakebot-body creation	L_segment > R_segment: Simplifies design effort for long snakebot, but less maneuverable. Optimal for base (distal segments) of snakebot.
Wall thickness	90% payload: pre version 10 Bad for FDM snakebot prototype: segments cracked under pressure and would not stay axially aligned	50% payload: Version 10 and later: good. Stable. Perhaps excessive.	Less than 50% payload: excessive for FDM, but may be necessary with sintered TiCu alloy
Segment radius	1" – FDM prototype: smallest size that can be hand-assembled. Fragile when thin-walled. Too large for production endoscope	2" – FDM prototype: good for proof-of-concept. Features that would be too small for naked-eye-assembly are manageable at this scale	>2" FDM: Unnecessary and wasteful

Table 2 - Segment Parameters

Known Segment Issues, Solutions, and Improvements

The last iteration, Version 12, is not quite optimal. It has control-cable tension hysteresis when moved from axially aligned to maximum deflection. The slack produced in the associated control cable (and all cables to distal segments that move in that degree-of-freedom) is small but enough to prevent deterministic kinematics, and the cumulative slack of multiple axes in the same degree-of-freedom all bent in the same direction (as when the snakebot curls in on itself) is not inconsiderable. Also, the wall thickness of Version 12 is excessive, a reaction to the inadequate wall-thickness of versions preceding it.

Despite the flaws of Version 12, it demonstrates the viability of a stacked segment design to generate tip-force omnidirectionally while the snakebot is contorted.

Control Cable Tension Hysteresis

The most pressing issue with Segment Version 12 is the cable-tension hysteresis. This problem manifests when the snakebot bends from axially aligned to a small-diameter curve: when aligned, all control cables are at maximum tension. As the snakebot bends, the control-cable path-length decreases due to the width of the 'shielded via' where it meets the 'Y-shaped' via.

The solution to this appears to be to curve the face of the 'Y-channel', increasing the total path length such that control cable tension remains constant over the range of segment motion.

Segment Wall Thickness

The second issue with Segment Version 12 is its payload. The current version has a segment diameter of 2", with a payload of 1" diameter or 25% of the segment axial area. This provides a stable snakebot, but robs it of its purpose, which is to deliver usable payload space. Note that the transition from Version 10 to Version 11 increased the wall thickness from 0.15" to 1", reducing the payload from 72.25% to 25%, which changed the snakebot from very unstable to

completely stable. The optimal design lies between these extremes, and will be a function of the snakebot diameter and material from which it is formed.

Tapered Y-Channel

The 'Y-Channel' of segment Version 12 removes part of the cam surface area. This may not be necessary. If the Y-Channel depth (as measured from the segment's center axis) were tapered from a maximum depth at the point where it joins the 'Shielded via' to zero at the cam surface, the utility of the via would be accomplished while maximizing the segment stability.

Manually Manipulated Snakebot

A manually manipulated version of the snakebot segment Version 11 was constructed.

Figure 20 - Segment 11 Snakebot – nonmotorized

This prototype was controlled by turning the spools at the snakebot's base. Since each axis is controlled by one spool, one person could simultaneously manipulate at most two segments. This demonstrated the ability of the snakebot to contort and to generate tip-force omnidirectionally. It was not clear that this paradigm would work once motorized, nor could this prototype demonstrate serpentine motion.

Motorized Snakebot

To enable the snakebot to perform serpentine motion and demonstrate the utility of the design of segment Version 12, a motorized snakebot was designed.

Baseplate

Design parameters for the motorized snakebot included a rigid baseplate to hold the snakebot with a motor for each snakebot joint (segment intersection). The prototype has eight segments, four in each of the two degrees-of-freedom. The baseplate also holds eight force sensors to detect cable tension on the control-cable. This latter feature is described in a subsequent section.

The baseplate chosen was formed from 12”x8”x3/8” aluminum, chosen because that thickness would not flex under applied control-cable tension. The baseplate also acts as a heatsink for the stepper motors.

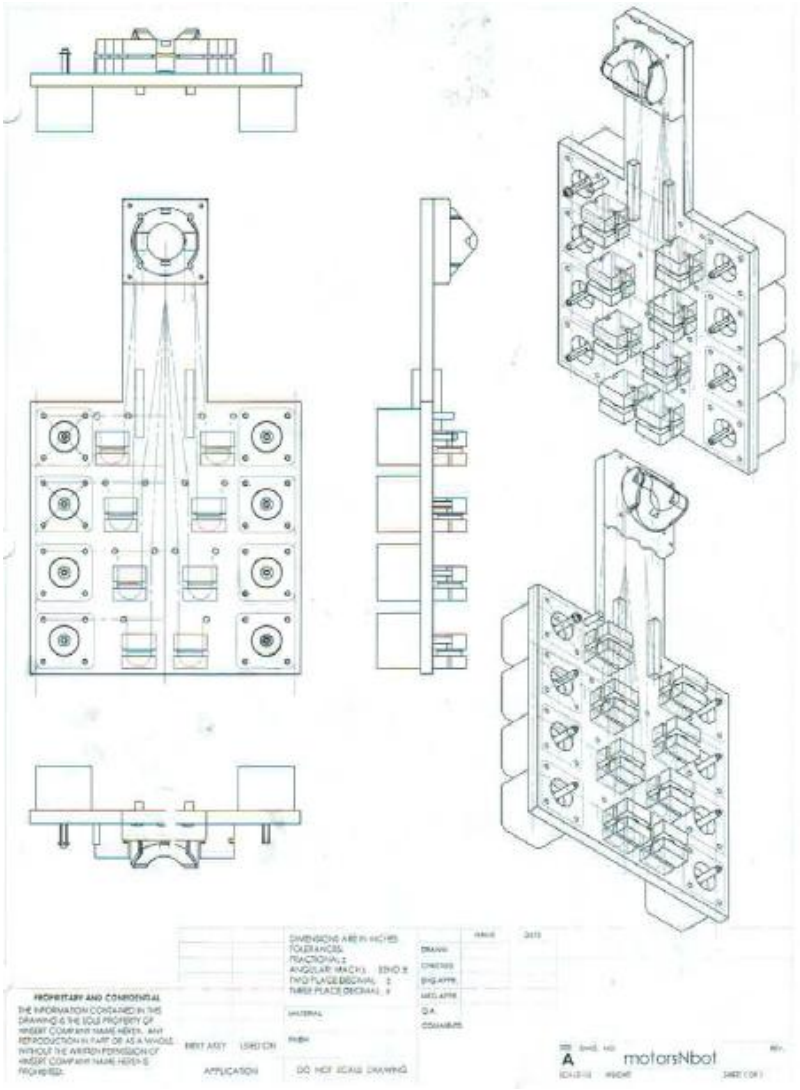


Figure 21 - Baseplate with Motors, Force Sensor Blocks, and Snakebot Base

The snakebot mounts at the end of the protruding arm, with the snakebot's lowermost segment's cam-face rotating against the cam-face of the hemicylindrical cam mounted to the baseplate. Both control and structural cables from the snakebot are routed to the baseplate: structural cables terminate at the baseplate, attaching the snakebot. Control cables route through the hemicylindrical cam attached to the baseplate, across the baseplate to the force sensor blocks, then to a stepper motor shaft.

Force Sensor Blocks

The prototype has a force sensor for each axis built into the baseplate. This sensor monitors the tension of the cable actuating that axis. There are several purposes to collecting this data. First, it is a way to generate axis limit data, since the limit-count of the stepper motor associated with a given axis changes as axes between that axis and the base move. Second, it offers the possibility of improved effector control and increased safety (31). Third, it offers the possibility of detection of interstitial tissue-type by 'palpitation'; bone will offer little deflection per unit force, while soft tissues will offer more. Last, it offers the potential of active position maintenance, of having the steppers increase cable tension in response to applied force.

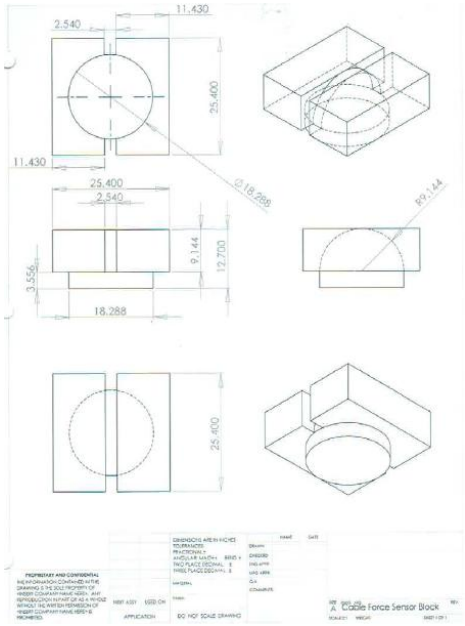


Figure 22 - Force Sensor Sliding Block

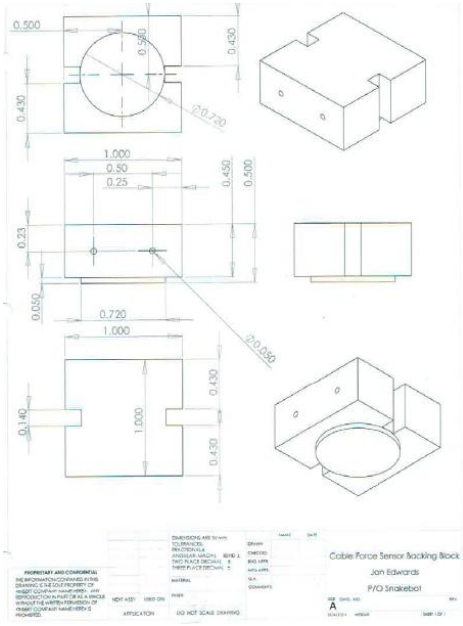


Figure 23- Force Sensor Stationary Block

The force sensor consists of three pieces: a stationary block bolted to the baseplate, a sliding block held against the stationary part by the control cable tension across the rounded groove in the back of the sliding block, and a force sensor pad located between the two blocks.

Before the baseplate was mounted to the sled (described below), the force sensors were electrically connected to the 328p microcontrollers (also described below). When the baseplate was mounted on the sled, the challenge of reconnecting the force sensors was greater than the possible utility they might provide. On the sled-mounted version of the snakebot, the sensors are in place, just not connected.

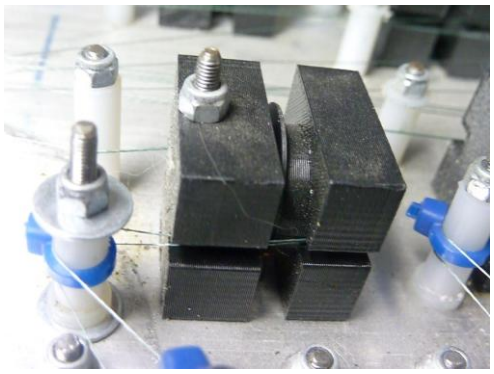


Figure 24 - Force Sensor Blocks on Baseplate

Stepper Motors

Steppers were chosen over alternatives because they are relatively inexpensive, powerful enough to move the snakebot, and able to move in discrete increments and therefore do not require an encoder for their controller to 'know' their position. Alternative actuators considered include pneumatics, DC motors, and memory metal.

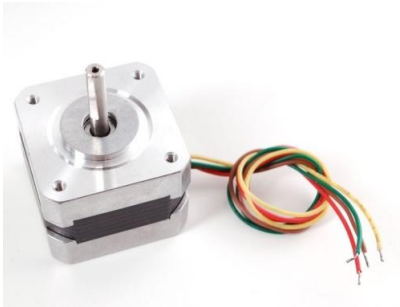


Figure 25 - Stepper Motor

The requirement of strength and small size of the actuator inside the snakebot dictated the use of remote actuation and cables. Low cost and the ability to use the stepper-count as an encoder further refined the option pool.

Note that in the figure "Motorized Snakebot (without sled)', the steppers have attached to their shaft a wheel to which the control cables are glued. This increased the effective diameter of the shaft, causing the segments to move faster for a given stepper rotation. The wheel also affected the force that could be generated by the snakebot segments to such a degree that in some

configurations the steppers were unable to produce enough torque to move the snakebot and would ‘skip’. When the stepper ‘skipped’, its controller issued a move command and updated its internal count of the stepper’s position, but the stepper did not move. Later iterations of the motorized snakebot replaced the wheel with flexible tubing of smaller diameter, which reduced skipping by the stepper and added a slight flexion between forward and reverse actuation, reducing the cable-tension hysteresis problem.

Figure 26 - Motorized Snakebot (without sled)

Stepper Controller

Arduino 328p

To control the snakebot, steppers were chosen. To control the steppers, the Arudino 328p was chosen because, when combined with the associated ‘motor shield’ (described below), it offered the advantages of:

1. Inexpensive: an Arduino 328p with motor shield costs roughly $50 and will control two stepper motors.
2. Programmable: the Arduino development platform allows for programming in C, C++, and assembly.
3. Digital IO – the Arduino 328p has 23 digital Input/Output lines, of which 21 are used by the motor shield in controlling two stepper motors
4. Analog to Digital Conversion – the 328p has six analog inputs.

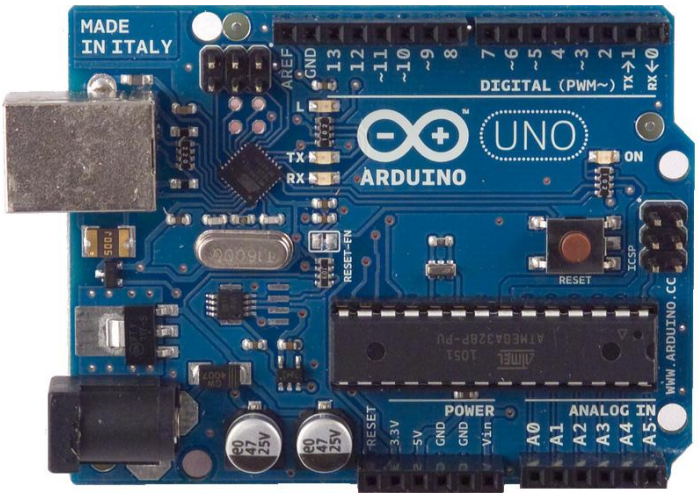
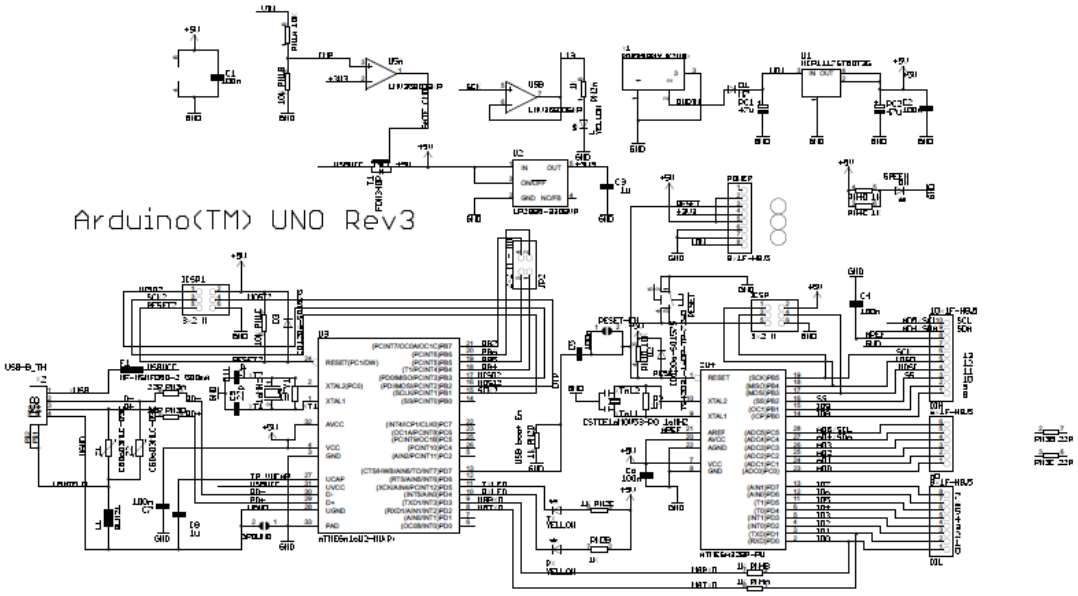


Figure 27 - Arduino 328p Microcontroller



Reference Designs ARE PROVIDED "AS IS" AND "WITH ALL FAULTS. Arduino DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, REGARDING PRODUCTS, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE Arduino may make changes to specifications and product descriptions at any time, without notice. The Customer must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Arduino reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The product information on the Web Site or Materials is subject to change without notice. Do not finalize a design with this information.

ARDUINO is a registered trademark.

Use of the ARDUINO name must be compliant with http://www.arduino.cc/en/Main/Policy

Figure 28 - Arduino 328p schematic

Arduino Motor Shield

The Arduino 'Motor Shield' is a daughter board kit that, once assembled, mounts to the 328p via 'stacking headers'. The motor shield is capable of controlling two stepper motors or four DC motors. It allows an external power supply to power the steppers instead of drawing power through the 328p, making possible the control of motors requiring higher voltages and currents.

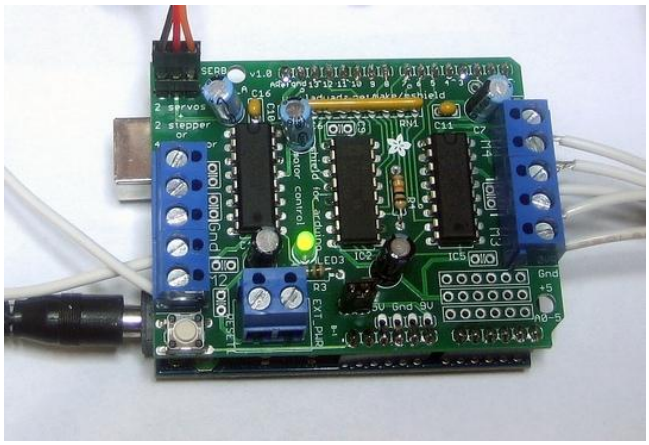


Figure 29 - Motor Shield

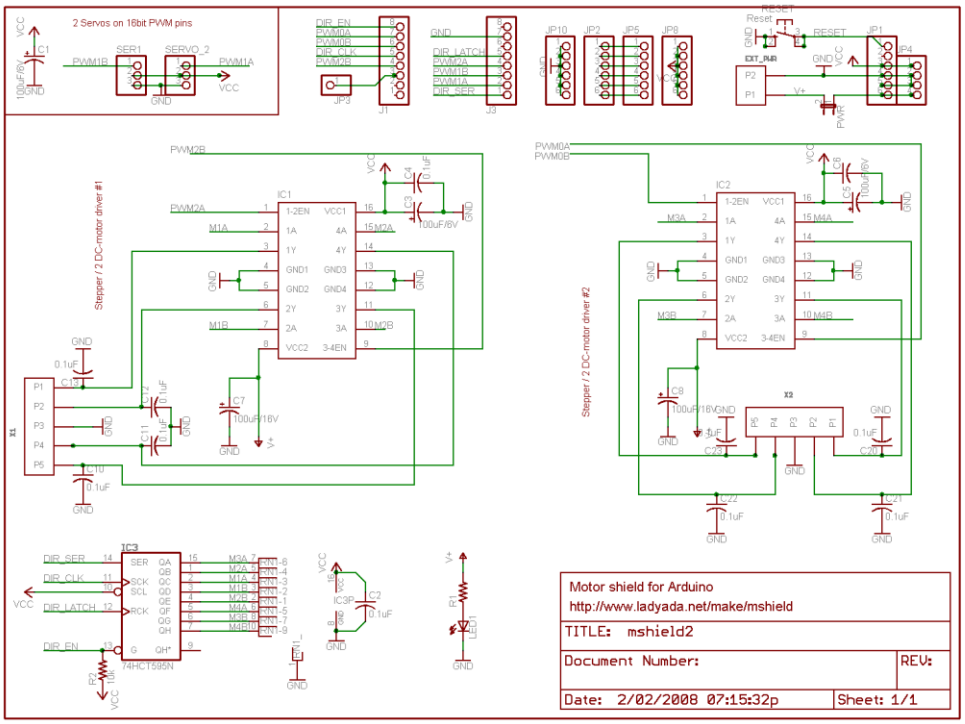


Figure 30 - Motor Shield Schematic

The motor shield as it is designed has several shortcomings, primarily the lack of fuse in the stepper-power input and the capability to create short circuits within the multilayered printed circuit board. The former issue is addressed in the 'sled' version of the snakebot. The latter is a known bug with no resolution.

Motor Shield Adaptation

To implement the sled version of the snakebot requires the snakebot baseplate, five 328p controllers (four to control the snakebot and one to move the sled), a 'sled' to support the baseplate and its controllers, and a stationary frame to support the sled. The moving portion is labeled the 'sled', and its weight is not inconsiderable. To move the sled, two steppers were used together. To ensure simultaneous movement of the two steppers, the motor controller attached to those steppers was adapted.

This adaptation consisted of disconnecting the lines from the motor shield's 74HCT595N to IC2 (pins 5, 6, 7, and 15 of the 74HCT595N), then connecting the lines from 74HCT595N to IC1 to IC2.

74HCT595N	Disconnect	Reconnect
	5	1
	6	3
	7	4
	15	2

Table 3 – Adaptation of Motor Shield

This disconnection and reconnection was accomplished by rewiring a chip socket so that all the changes were confined to the chip socket: if the motor shield, the 328p, or the 74HCT595N needed to be replaced, no alteration of those components would be required.

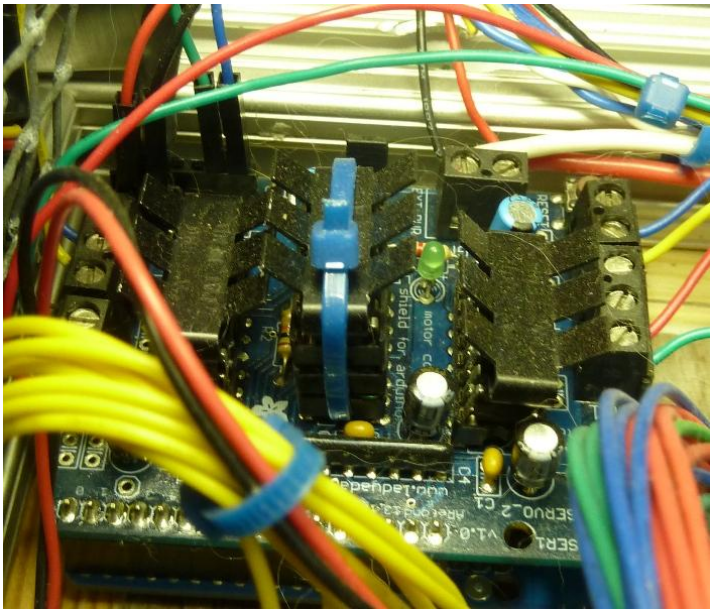


Figure 31 - Motor Shield Adaptation - Custom IC Socket

The blue zip tie holds the IC Socket, the IC, and the heat sink. It is there to keep the stack compressed and so ensure positive electrical connection of the socket's pins.

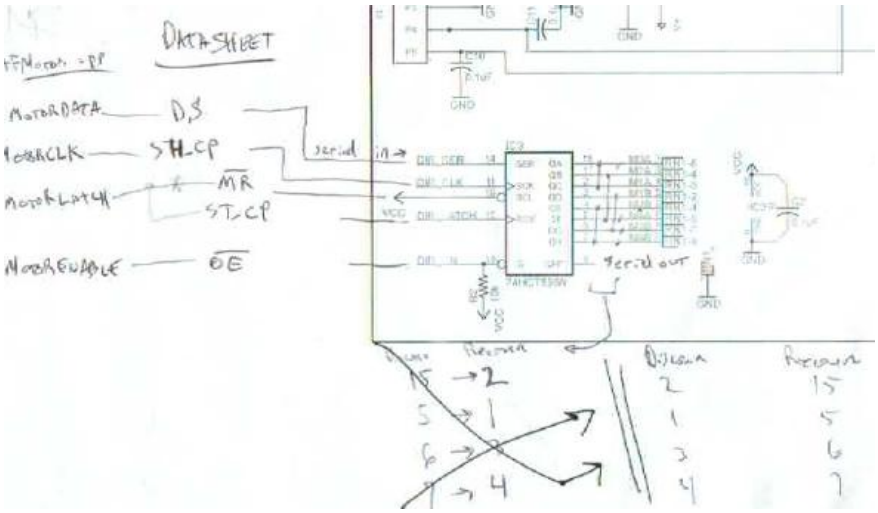


Figure 32 - Schematic of Motor Shield Adaptation

The effect of this alteration is that both ganged steppers are controlled by the signaling intended for IC1, ensuring simultaneous movement of both steppers. The fan-out on 74HCT595N was not exceeded, but a heat-sink was added to prevent that IC from overheating.

Further documentation of the ganged steppers appears in the section on the snakebot's sled.

Arduino 2560 microcontroller

Early in the process of software development of the motorized snakebot, the hierarchical control system consisted entirely of a network of 328p microcontrollers. A bug manifested as an asynchronous reset of the central 328p. The memory map of allocatable RAM and ROM had been violated: the stack was blown. There was insufficient RAM for the software application run on a 328p.

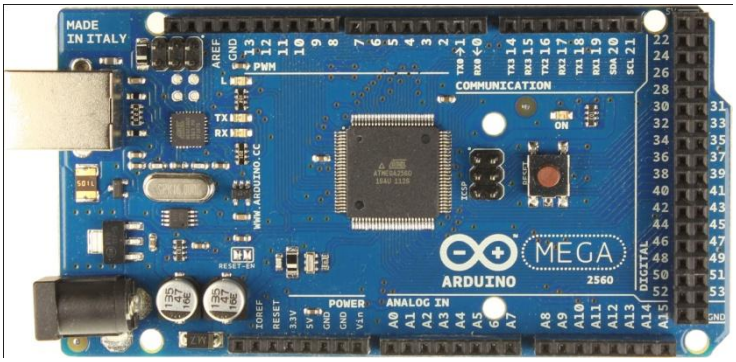


Figure 33 - - Arduino 2560 microcontroller

The 2560 was selected to replace the central 328p. It used the same development environment, had the same TWI communications hardware as well as the same USB hardware, and supplied larger RAM and ROM. It also offered 16 analog-to-digital input lines, offering the capability to monitor all the force sensors currently in use.

Joysticks

Control of each axis is achieved through software. This can be implemented through several interfaces, including a joystick. Each joystick controls two orthogonal and sequential axes of the snakebot. Segment number zero is the one closest to the snakebot's base.

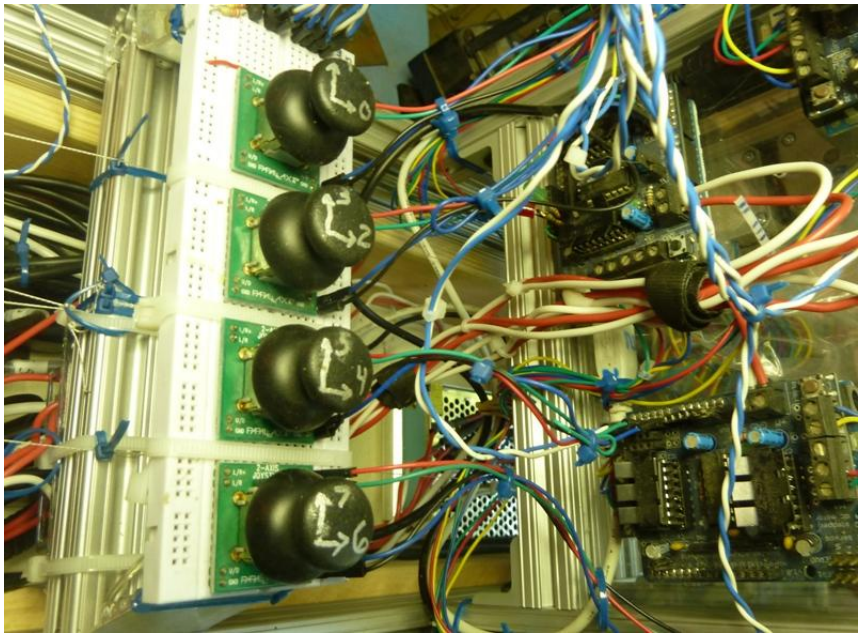


Figure 34 - Joysticks Controlling Eight Snakebot Axes

Each joysticks consists of two variable resistors connected between 0VDC and +5VDC, generating a voltage between zero and five volts proportional to the deflection of the joystick. This voltage is fed to the analog-to-digital converter of the 328p which controls the axes associated with the joystick.

The sled is also controlled by a joystick. Although both axes of the joystick are connected to the 328p's A/D input, only one axis' data is used since the sled has only one degree-of-freedom.

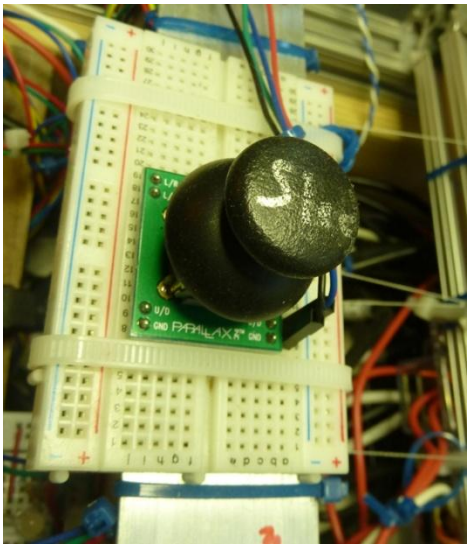


Figure 35 - Sled Joystick

Although the snakebot can be controlled using joystick input, this is only marginally better than the manually manipulated snakebot described in the previous section. Subsequent software development built on the joystick input to implement 'claymation'.

Stepper Power Supply

The stepper motors require more power than can be supplied through the 328p's USB connection. To provide the correct voltage and sufficient current, a power supply is connected to each motor shield.

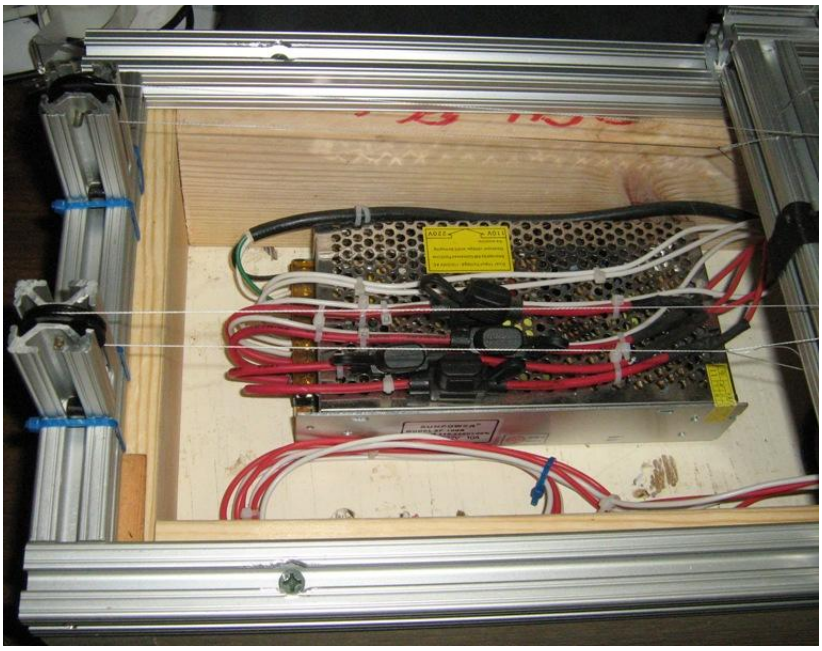


Figure 36 - Power Supply for Stepper Motors

Each line from the power supply has a fuse. The power supply converts 115VAC to +12VDC. The 115VAC is switched on/off by a relay, which is itself controlled by a digital line from the Arduino 2560 (described in the computer control section). This is necessary for several reasons, including the need to be able to move the snakebot

Figure 37 - Relay controlling power supply

manually, a feat impossible when the steppers are powered. In addition, once the power supply is on, the power it supplies effectively latches the 328p 'on', preventing the 328p array from powering off, which prevents safe access to circuitry and the snakebot.

Control of the relay is described in the section on 'Computer Hardware'.

The Sled

After the snakebot was mounted on the baseplate, the ability to move the entire baseplate and thereby create axial snakebot motion became necessary. To that end, various paradigms were considered, including mounting the baseplate on a vertically oriented elevator, a gantry robot to surround the baseplate, and an inflatable cushion. Since the goal was to move the snakebot axially, not necessarily vertically, the simplest implementation was to mount the baseplate vertically and translate it horizontally. A sled would fulfill these requirements.

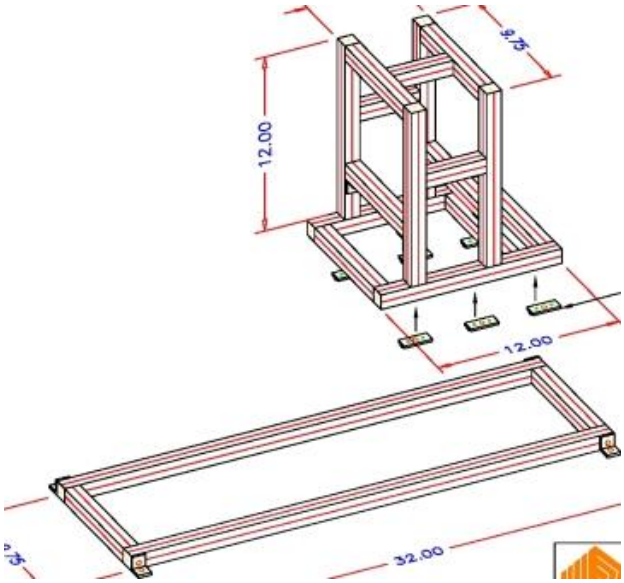


Figure 38 - Snakebot Sled and Frame

To that end, the author worked with Bertelkamp to produce a horizontal frame on which a sled would slide. Roller bearings would have reduced the static friction to be overcome in moving the sled, but costs indicated the use of a teflon bearing. The frame and sled were built from 8020 aluminum, chosen for both low cost and low weight.

The 8020 aluminum frame was mounted on a 2x6 pine frame to create a space beneath for storage of the USB hub, power pigtail, and wiring.

Figure 39 - Snakebot mounted on Sled

hub, power supply, power relay, and the many cables necessary to connect the electrical components.

The sled is dragged along the base by two ganged stepper motors. The motors share control voltage from a 328p microcontroller and 'Motor Shield', so they step together with one motor stepping clockwise while the other steps counter-clockwise. The shafts of the motor are connected by a piece of vinyl tubing with an

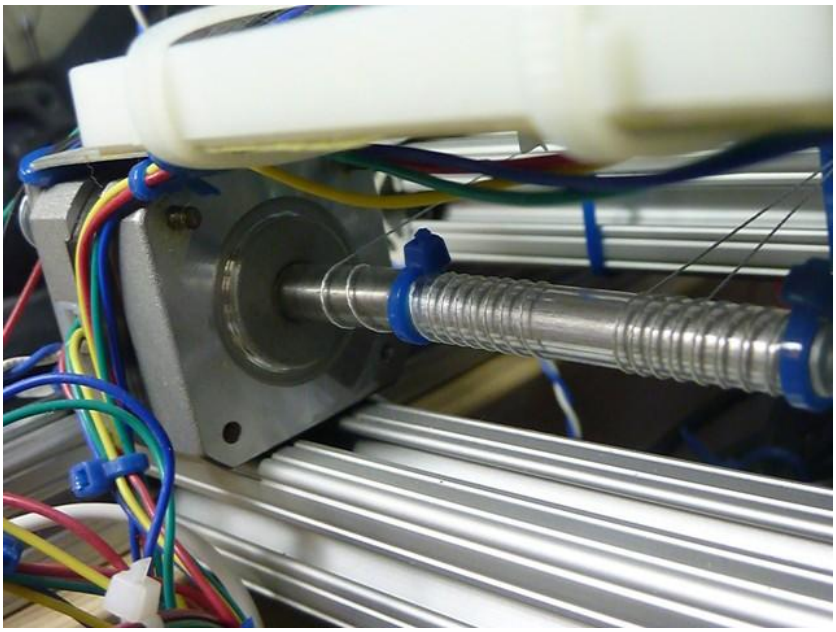


Figure 40 - Ganged Steppers to Drag Sled

aluminum tube inserted in the tubing between the stepper's shafts. The vinyl tubing serves two purposes. First, it increases the effective shaft length, allowing more control cable to spool onto the shaft without knotting over the control cable end which pulls in the opposite direction. Second, the short length of unsupported vinyl tubing between the motor shaft and the control-cable attachment acts as a shock absorber, allowing some 'flex' to the system. This flex reduces the torques when the motors do not move exactly simultaneously and thereby reduces startup currents in the motors, keeping the motors from burning each other out.

The cables that move the sled are tied to the sled, routed to the steppers' conjoined shafts, back across the sled to a set of pulleys at the opposite end of the base, then back to the sled where they are tied off.

Figure 41 - Sled Pulleys

The stepper motor controlling the snakebot's axes has a short length of vinyl tubing over its shaft, which replaced the wheels that appear in the figure "Motorized Snakebot (without sled)". The vinyl tubing has an inserted piece of the same diameter aluminum tubing mentioned in the section on the sled motors. The aluminum is present to control the deformation of the vinyl from stress from the attached control cable. The vinyl tubing serves a similar purpose to the tubing

Figure 42 - Axis Controlling Stepper's Flexible Shaft

used in the ganged steppers described above: it increases the shaft diameter, increasing the cable speed per stepper rotation, which moves the snakebot faster. It also acts as a damper of the control-cable hysteresis problem because the vinyl can flex slightly, absorbing the tension when the axes are aligned, and taking up some of the slack when the snakebot curls. This decreases the determinism of the kinematics of the snakebot, but since the snakebot was designed to work with minimal determinism, this seemed to be an acceptable tradeoff.

Computer Hardware

The computer hardware controlling the snakebot's stepper-motors consists of several parts arranged hierarchically. At the top level is a PC sending ASCII commands over USB to an Arduino 2560 via a USB hub. Currently, the command set available for PC-to-2560 is basic, putting the onus for implementing intelligent serpentine control onto the to-be-developed PC-based application.

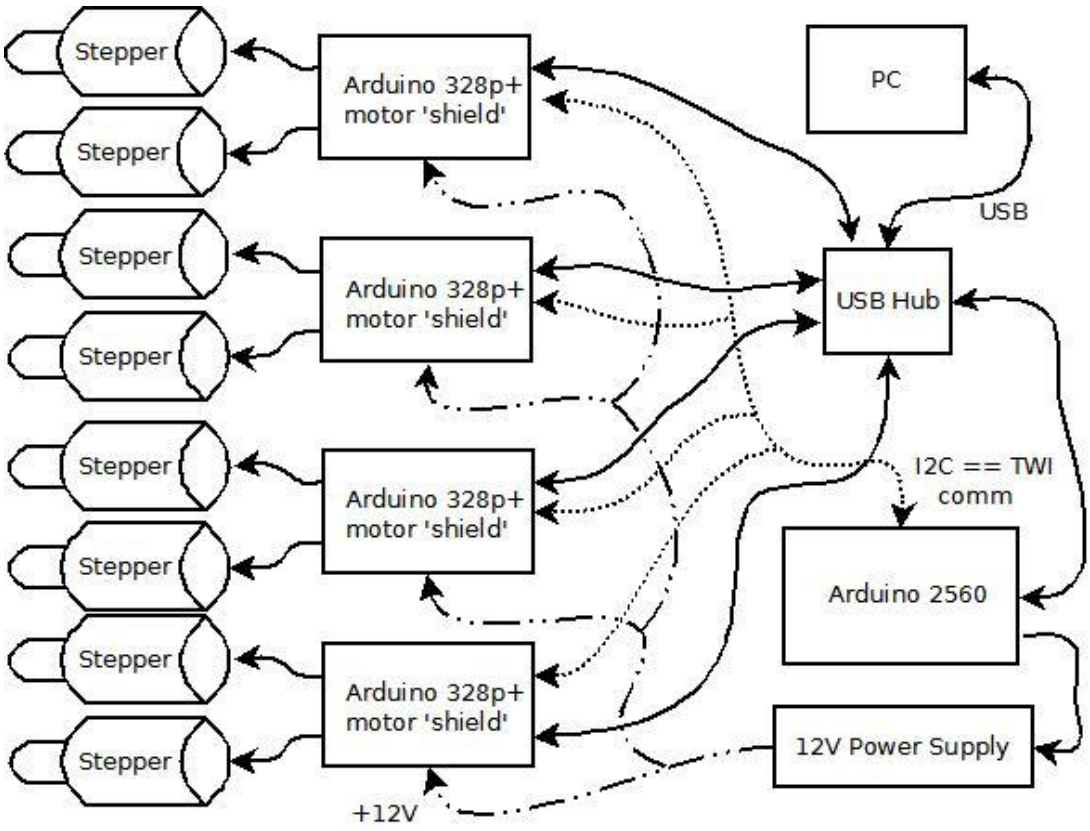


Figure 43 - Computer Hardware Topology

Power Relay

On receiving a POWER_ON command from the USB interface, the 2560 will activate a TTL relay (a.k.a. 'pigtail', described in the section regarding the power supply) which applies 115VAC to a 12VDC power supply. This 12VDC is distributed to the 328p microcontrollers, which use this power source both to run the microcontrollers and to power the steppers. Note that

the 328p can run off USB power, but cannot move the steppers without the 12VDC. The 12VDC power supply lights a green LED to provide a clear indication to the user that the steppers are powered.

Communications

Distributed control systems such as the one implemented in this multi-axis snakebot require communication between the control elements. This system has two communication systems: a USB port on each microcontroller through a USB hub to a PC, and a TWI system between microcontrollers.

USB

The software is designed as a top-down control system with the PC issuing commands (from a user, a script, or higher-level software) to the 2560 via a USB hub. Since the 328p microcontrollers must be programmed and their software debugged, the ability to communicate from the PC over USB to the 328p is also necessary. This can occur in parallel with the PC-to-2560-over-USB and 2560-to-328p-over-TWI communications. In addition, the 328p software is written such that it cannot distinguish between commands from the 2560 and commands from the PC-via-USB, so direct control of each 328p by the PC is possible and may be the means by which higher level software control by the PC of the snakebot is implemented.

TWI

The software design includes the 2560 receiving commands from the PC over USB, then implementing those commands by issuing commands over the TWI bus to an array of 328p and collecting subsequent status messages from the 328p over the same medium. TWI is two wire interface, a.k.a. I2C or the Phillips protocol. This top-down hierarchical control enables high-level and abstract control of the entire distributed control network using the PC-to-2560-over-USB communications.

Since TWI is a shared medium, communication lag can be nondeterministic, making coordinated motion problematic. Since the snakebot moves very slowly, and since the objective of the snakebot is not tight control but rather proof-of-concept for the snakebot design, this rather sloppy level of control was acceptable.

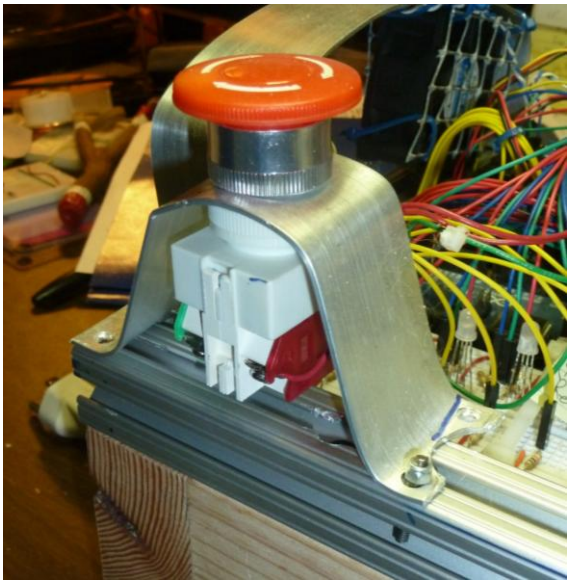


Figure 44 - Stepper Motor Interrupter Slap Switch

Mushroom Slap Switch

There is a 'slap switch' to interrupt the 2560-to-pigtail-relay TTL signal, thereby removing power to the steppers should the snakebot be in a position where it might harm itself or the environment.

2560 Status LEDs

The 2560 has an array of 24 LEDs (contained in 8 triple-LED fixtures) with which to signal conditions to the user. For each of the 8 snakebot axes, the 2560 can signal Upper Limit, Lower Limit, and Overforce. Since the sensors to detect the limit conditions are not yet in place, this is available for future work, but the LEDs do provide a useful software debug interface.

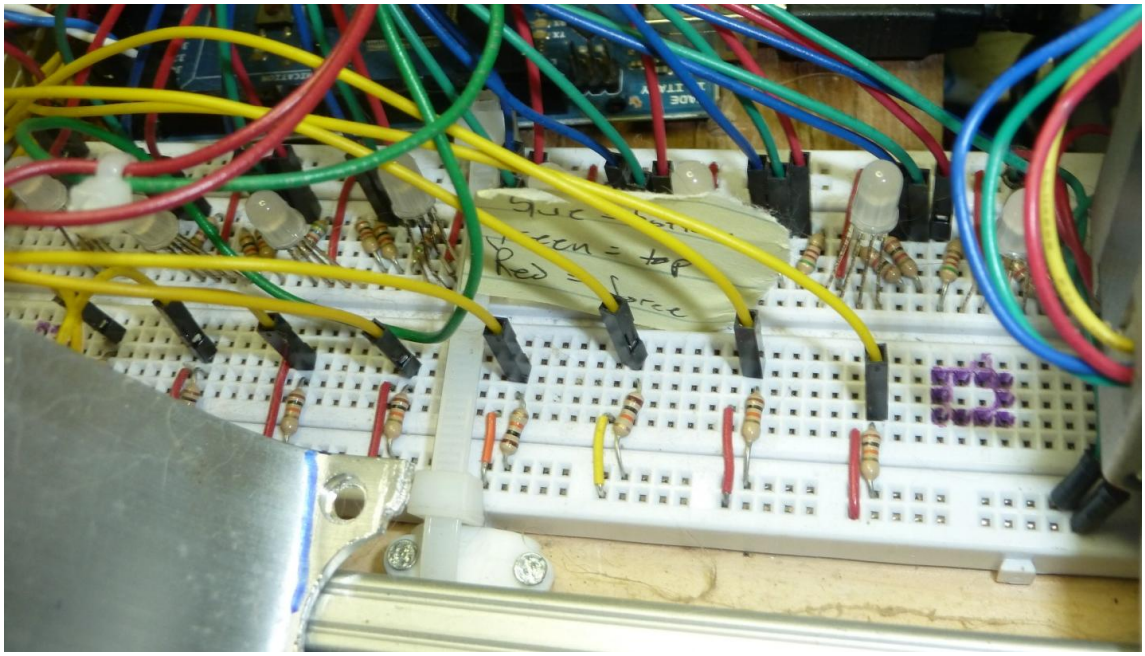


Figure 45 - LEDs controlled by 2560

The resistors and the yellow wires connected to them in the above figure are the connections for the force sensors, should that functionality be required.

Software

The software that controls the motorized snakebot is intended primarily as an interface for subsequent to-be-written software which will implement higher-level intelligent kinematics and controls. This is a necessary design constraint due to the use of microcontrollers with limited resources: the processors are all 8-bit, relatively slow (16MHz for the 328p, 20 MHz for the 2560), with limited RAM, EEPROM, flash, and IO. By deferring the high-level kinematics to a hypothetical external computer, the interface and command set requirements are diminished to levels manageable by the microcontrollers, and the high-level kinematics calculations can be deferred to a processor with the resources to handle them.

Source Code Overview

The software was developed in C and C++ using AVR Studio version 4.19. There are many platforms for developing embedded software, but this application had a toolset intended for the Arduino chipset. It is also freeware, and as such had shortcomings. In particular, the 328p must have its flash code loaded using a DOS application called 'AVRDUDE', while the 2560 has to be loaded using special hardware called 'AVRISP mkII'.

Both the 2560 and 328p software is, at root, an endless loop scanning for control commands. These commands can come from the USB or TWI (described in the 'Communications' section above). Commands from either source are treated identically, though the USB is checked first and therefore has priority over TWI.

There is, in the background, a timer-generated ISR that polls the A/D inputs and writes those values into global variables which are evaluated in the above loop. The A/D inputs are used, in the 2560, to monitor the force sensors which report control-cable tension. Although those sensors are present both physically and in the software, they are not connected to a microcontroller. The A/D inputs in the 328p are used to monitor the two axes of the joystick.

There are three sets of source code: that code which pertains only to the 2560, that which pertains to the 328p, and that which is common to both. The source code will be presented in that order in the software appendix.

USB Interface

The software controlling the snakebot is hierarchical, with all snakebot motion originating from the user interface software running on the PC. This top-level application sends and receives ASCII over a USB port. These commands can be directed at a variety of USB ports: to the 2560 for high-level snakebot control or to one of the five 328p microcomputers which control stepper motors.

The PC application chosen is called "Terminal v1.9b by Br@y++", and is downloadable from

https://sites.google.com/site/terminalbpp/

This application was chosen because it can run multiple instantiations to communicate with multiple USB ports simultaneously, can record the contents to and from a port, and has macros to load commands quickly.

PC-to-2560 Command Set

The PC-to-2560 via USB connectivity is intended both as a software development interface and as the primary means of control of the snakebot. The command syntax is a single-letter command followed by arguments as needed.

Command	Intent	Implementation	Text Into 2560	Text from 2560
M	Move axis	Move axis	Mx,123	n/a
J	Move axis	Joystick move	Jx,123	n/a
U	Axes limit adj	Flaky code		
L	LED	Turn LEDs on/off	LF0F3	none
T	Ping TWI	Ping 328p	TxP	TxR
P	Axes status req	Status from 328p	PRx	PA…<status>
			PC	Text current axes
F	Force sensor	Read a/d	Fab, ab==bitmap	Axes' force
G	115 VAC on/off	Relay activation	G0 or G1	n/a
E	EEPROM access	Read/write EE	See below	
H	Help menu	Print menus	H	

Table 4 - PC to 2560 Command Set

parseMoveCommand

The parseMoveCommand is a 5-byte command beginning with ‘M’ or ‘J’, both of which are handled identically. The first two arguments are:

Arg[1] = ascii-decimal axis number to move

Arg[2] = ‘,’

Arg[3,4]	#def (snakebot_common.h)	Parsed
32767	RELEASE_AXES	De-power steppers
(RELEASE_AXES-1)	INIT_AXES	Init axes’ data struct. axisPos=0
(INIT_AXES-1)	ZERO_AXES	Move axes to axisPos=0
200	MAX_AXIS_COUNT_LIM	Max stepper-count (arbitrary value)

Table 5 - PC to 2560 Move Command Parameters

parseLEDcommand

The parseLEDcommand is a five byte command string with ‘L’ as arg[0]:

ex LF0F3

returns: TRUE if successful, FALSE if not

Arg[1]	Parsed	Arg[2]	Arg[3,4]
F	Force LED (red)	0|1 – on/off	ascii-HEX bit-map
T	Top-limit (green)	0|1 – on/off	ascii-HEX bit-map
B	Low-limit (blue)	0|1 – on/off	ascii-HEX bit-map
W	Control 328p LED	L	21|20 |51|50

Table 6 - PC to 2560 LED control commands

The ascii-HEX bit-map in args [3,4] controls which LEDs to turn on/off with each HI (==1) bit turning that LED ON. The LEDs are LO-actuated writing that bit LO but the args are HI==ON.

parseEEPROMdataReq

The parseEEPROMdataReq command is a minimum of two bytes but can be as long as six bytes and begins with arg[0] = 'E'

This function performs read/write/goto_axes_configuration data in EEPROM

Arg[1]	Parsed	Arg[2,3]	Arg[4,5]
B	new stored sequence	Index in LL	n/a
E	add new config data to end of LL	Index of LL root	n/a
I	insert new config data into LL	Index after which to insert	n/a
U	update current axes config	Index to update	n/a
D	remove config from LL	Index to delete	n/a
X	Remove all elements of LL	Index of root of LL	n/a
Z	Clear EE of all LL data		n/a
L	Print list	Index start printing	n/a
G	Goto this config	Index of config	n/a
N	Goto next config	n/a	n/a
P	Goto previous config	n/a	n/a
C	Cycle between configs	Index of starting config	Ending config
Q	Iterate axes' config	Index of starting config	Ending config
H	Halt axes' config iteration	n/a	n/a
M	Screen-print (USB) EE-access menu	n/a	n/a

Table 7 - PC to 2560 - EEPROM access command set

PC-to-328 Command Set

Overview

The PC-to-328p via USB connectivity is not intended for control of the snakebot. Direct control of the snakebot can be accomplished through that interface, but would require writing a PC-based application. Rather, this utility is intended to be used in development and debugging of software running on the 328p.

Command	Intent	Implementation	Ascii Content in	Ascii Content Back
M	Move axis		M1+123	none
J	Move axis (joystick)	Joystick moved	J0-67	None
T	Ping TWI	Ping/ack	T3P	T3R
L	Turn LED on/off	Set LED on or off	L21 or L50	none
E	EE Prom access	Set TWI_id	ET32	Text verifying success
		Set Joystick parms	EJUT	Text verifying success
U	Axes' limit update	Lower axis moved	Axis status msg	none
P	Axes' status request	PR2	PA2…	Sends text
G	Power Pigtail On/Off	ping from 2560	G1 or G0	G1 or G0

Table 8 - PC-to-328p Command Set

joystickEE Command

The primary parse index to determine this command is EJxx, where 'xx' is Argument 2 and 3

Arg 2	What it does	Arg 3
U	up/down	T or B – current val -> UD top/bottom
L	Left/right	T or B – current val -> LR top/bottom
C	Center – avrg a/d for cntr val	n/a
G	Read from EE to global variables	n/a
P	Print current joystick A/D to USB	1|0 – streaming of joystick A/D to USB
V	Print current joystick min/max	n/a

Table 9 - PC to 328p - Joystick Calibration Command Set

Chapter 4

Testing the Snakebot

The snakebot endoscope was designed to provide surgeons a toolset capable of generating tip-force deterministically and omnidirectionally. This section will describe tests to discern whether this motion is deterministic and repeatable. That the snakebot can contort and subsequently produce tip-force is demonstrated in a video viewable at:

https://www.youtube.com/watch?v=mSu23Sg4Imw

Zeroing the Snakebot Steppers

The snakebot endoscope is a robotic puppet design. As with all robots, it is necessary to put the actuator into a known physical configuration to facilitate 'zeroing' the stepper motor count. The snakebot has nine axes which all must be in known physical configurations when the stepper-count is assigned the value of zero. The two methods used to accomplish zeroing are described in the subsequent section.

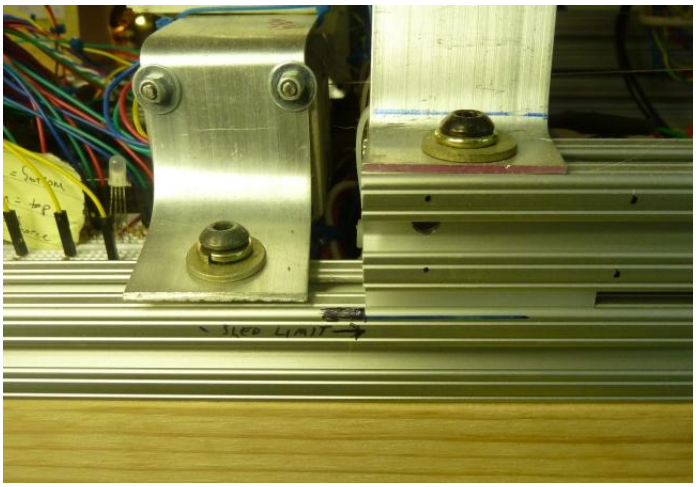


Figure 46 - Sled Zeroing

Zero Sled Position

Of the nine axes to be 'zeroed', the sled was the simplest to accomplish. In an industrial robot, axes are zeroed by moving them until a limit switch is encountered. The sled-mounted snakebot uses a similar paradigm but with a visible limit and manual positioning of the sled.

The sled must be manually positioned, not placed using the joystick since the hysteresis between sled-control-cable tension of pulling the sled backward versus forward is enough to spoil the determinism.

Note this step is not necessary for the test of determinancy. If the sled were forward from this zero-configuration, the only effect on the test would be a loss of sled-axis forward motion range and a possible collision between the sled-pulleys and the maze-table (the maze table is described in the subsequent section).

External Bracing to Zero Snakebot Segments

The first method of placing the snakebot in a known configuration is to externally hold it in position. Since the proof of deterministic motion requires the snakebot to navigate a three-dimensional maze, it was necessary to not only position the snakebot in its 'zero' configuration, but also to position the maze relative to the snakebot.

Maze-Table Fixture

The goal of creating a known physical configuration from which to run determinacy tests was achieved by building a table to hold the maze, marking where on the table the maze was positioned, and positioning the table relative to the snakebot.

Figure 47 - Fixture to positon Maze-Table relative to Snakebot Sled

The fixture to position the maze-table is rigid and fits precisely into the inset table surface. It has an extension which aligns with the sled-base, establishing the distance and orientation of the table relative to the base of the snakebot..Once the table is positioned relative to the snakebot by the fixture, this is the 'zero' position of the sled.

Zeroing Snakebot Axes

Zeroing the eight snakebot axes is accomplished by the PVC frame attached to the aluminum fixture. This frame is rotated into place after the fixture has established the snakebot-frame and maze-table positions.

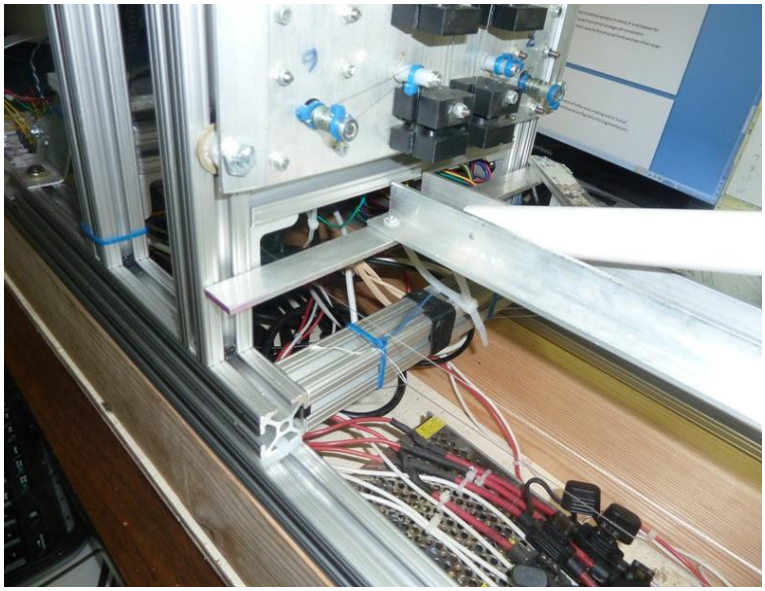


Figure 49 - Fixture to Position Maze-Table Relative to Sled Base

Results of External Bracing

The external bracing fixture worked well for positioning the maze-table relative to the snakebot sled-base. It worked less well for positioning the snakebot. This latter failure was due to the difficulty of normalizing tension on the control cables to both sides of each segment. Asymmetric residual tension predisposed the snakebot to bend in the direction of the more-tense cable once the frame was removed.

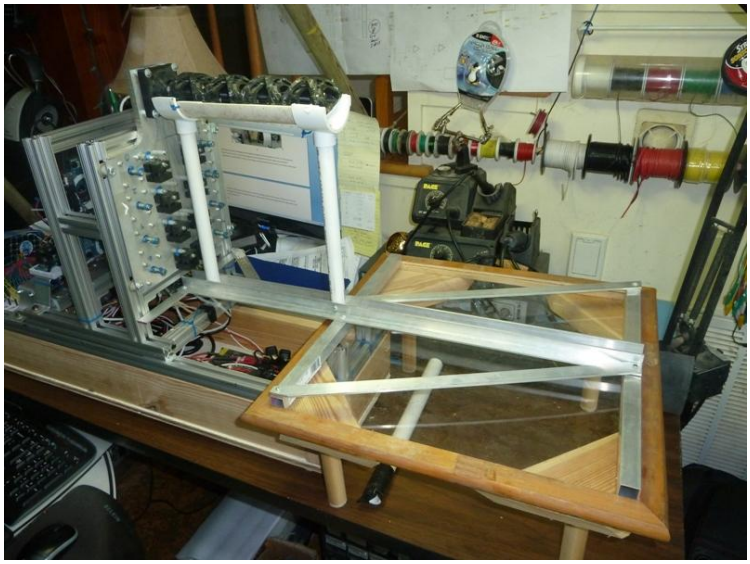


Figure 48 - Fixture to Zero Table, Sled, Snakebot – External Bracing

Internal Bracing to Zero Snakebot Segments

A simple and serendipitous solution to the problem of zeroing the snakebot segments was to insert a piece of PVC pipe that exactly matched the inner diameter of the segments. The external brace fixture was still necessary to ensure the position of the maze-table relative to the sled base, but the PVC external bracing portion of the Maze-Table Fixture became superfluous.

Figure 50 - Fixture to Zero Table, Sled, Snakebot – Internal Bracing

Results of Internal Bracing

The internal bracing worked. The process of working the PVC pipe down the series of snakebot-segments seemed to normalize control-cable tension across the segment. If the stepper-motor power were to be applied while the internal bracing pipe was in place, the steppers would hold the snakebot in its zero configuration.

The black tape at the distal end of the PVC pipe is to denote which end to insert into the snakebot, since it was necessary to lubricate the pipe before insertion.

The Maze

To test the ability of the snakebot to repeatably and deterministically navigate a three-dimensional path, a maze was developed. The maze consists of two hoops that have only 0.36" clearance for the snakebot.

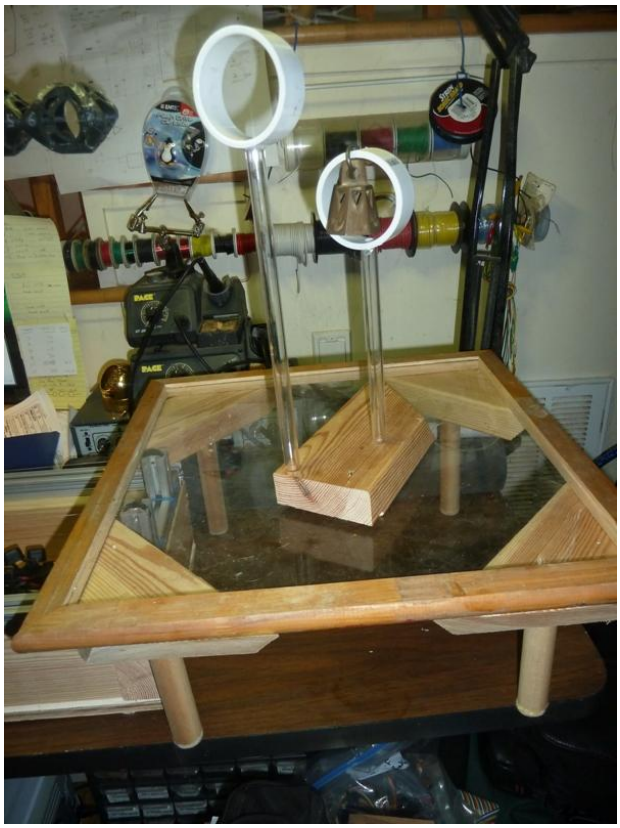


Figure 51 - The Maze

The two hoops are at different heights and on different planes, requiring a multiple-axis maneuver to navigate. Once the table is zeroed by the Maze-Table Fixture, the maze is placed on the table, aligned by a notch at one end of the maze and by marks on the table.

Navigation of the maze starts with the snakebot entering the maze at the upper hoop.

The snakebot is not particularly long relative to its diameter, and every bit of the snakebot length is required to thread the maze. Because sled motion does not demonstrate the navigability of the snakebot which is to be proven, the starting configuration has the snakebot already threaded into the start of the maze.

Figure 52 - Maze in position on Table with Snakebot at Zero

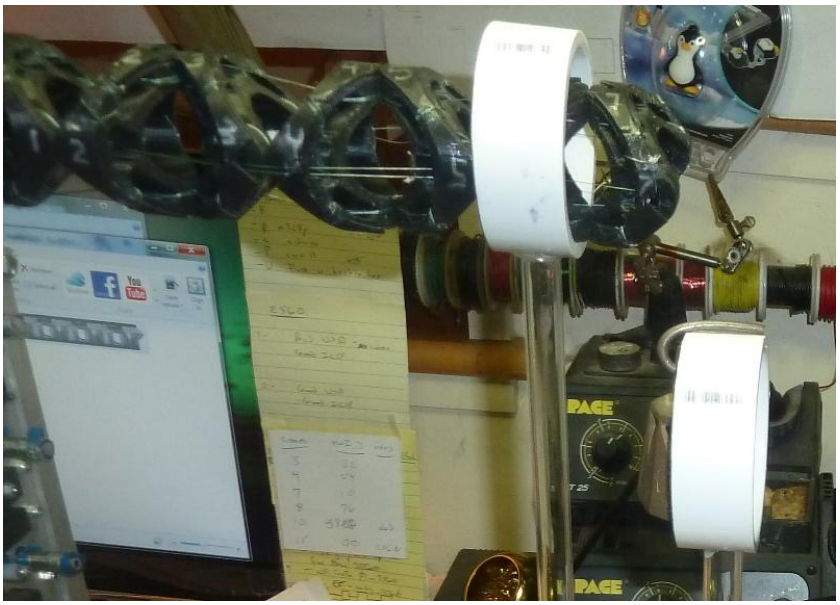


Figure 55 – Snakebot Zeroed at Start of Maze

Figure 54 – Snakebot Proceeding through Maze

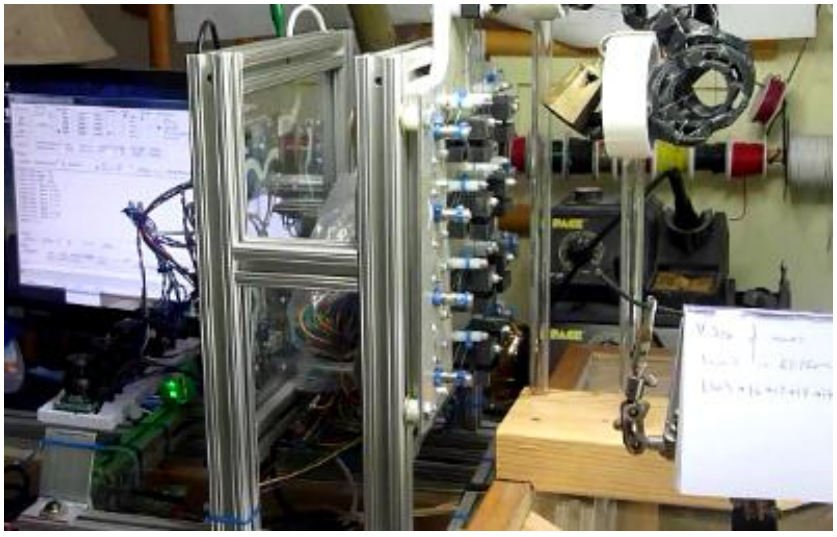


Figure 53 – Snakebot at End of Maze

Maze-Test Results

The snakebot was too slow to ring the bell, but as was shown in the videos (links to videos appear above), the snakebot did move the bell. This demonstrates the generation of force while contorted by further contortion (as opposed to force generated by moving the snakebot sled-and-base).

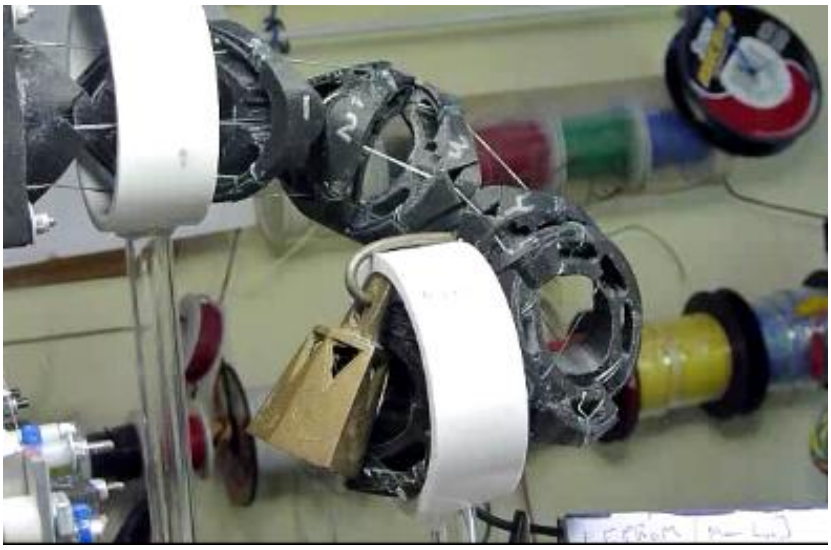


Figure 56 - Snakebot at End of Maze (closeup)

Since the maze is three-dimensional, navigating requires coordinated manipulation of orthogonal axes. This demonstrates the capability of the snakebot to navigate to and operate on distal surfaces of a patient's internal anatomy. For example, this capability would enable a surgeon to operate on the dorsal surface of the liver from a naval keyhole incision.

Figure 57 - Snakebot in End-of-Maze Configuration without Maze

Chapter 5

Results of Snakebot Design

The snakebot's serpentine motion is rudimentary with only eight segments (four segments per degree-of-freedom). That the snakebot can perform serpentine motion can be inferred, but, like attempting pattern recognition from extremely low-resolution images, is not clear from the tests performed. The control software in the prototype is also rudimentary, offering an interface for hypothetical high-level PC software control, but implementing only a basic movement command set. Because of this limitation, any serpentine motion performed by the prototype will be only as good as can be accomplished by moving the snakebot from configuration to configuration using joystick control of each individual axis.

The prototype is 2" diameter with a 1" hollow core, is 9.6" long and can curve into a 5" (outer diameter) circle. Assuming this flexibility is a function of segment dimension and will remain constant as the segment size is reduced, this means that a 10mm diameter snakebot will be able to navigate a 25mm diameter curved path. This is by no means a panacea for all surgical challenges, but does increase the surgeon's toolset and lexicon of techniques.

Movement of a proximal segment changes the orientation of all distal segments, so coordinated motion of multiple axes is necessary to maintain orientation of the tip during proximal segment motion. The kinematics is complicated by flex of the snakebot due to several factors, including cable stretch, stepper flex, control cable hysteresis, and segment deformation, factors which vary as the snakebot's position changes (32). Deterministic control may be

inherently impossible, but heuristics, fuzzy logic, or expert system control may compensate for physical challenges. (33; 34; 35).

The segment physical design may be optimized further by successive iterations. Since each permutation entails weeks of work, this may preclude permuting the entire solution space. Since control cable-tension hysteresis is the most immediate problem, and since the solution seems clear, that should be addressed first.

Chapter 6

Future Development

Pair of Coordinated Snakebots: The Caduceus

The real strength of this robotic endoscope design becomes apparent when it is used in a caduceus configuration, with two snakebots working together as one tool. In the caduceus paradigm, the snakebots alternate function, with one holding itself steady while the second uses the first as a stable, safe set of points against which to perform serpentine motion to generate tip force, minimizing force against patient's tissue.

Such a paradigm offers an increase in stability and strength with the stationary snakebot providing active force-compensation for forces applied by the moving snakebot. This paradigm also offers the possibility of stereo vision from two snakebot tips and of two-handed-tool usage.

The kinematics will be greatly complicated by the need to work with and around a second snakebot, further underscoring the need for a new non-Von Neumann control-system architecture.

Integrating Force Sensor Data

Each axis' control cable crosses a force sensor, offering the ability to infer tissue type by palpitating (by deflection/force). When unsupported by interstitia, the snakebot tip will flex if subjected to force. Using force-sensor data, the snakebot can actively compensate for applied forces. Implementation of such a system would provide a potential safeguard against tissue damage by inadvertent generation of excessive force.

A force/torque sensor should be inserted between the plate holding the stepper motors and the snakebot base to collect axial snakebot force/torque data for control during sled-induced push or pull. (31)

Intelligent Path Planning, Kinematics, Control

The interface presented to the surgeon is to be a drive-from-the-tip perspective, with the body of the snakebot performing a follow-the-leader paradigm. Path planning will be complicated by the performance of serpentine motion for self-propagation, plotting safe 'hard points' against which the endoscope can press, and by the necessity of monitoring force sensors to prevent collateral damage. Movement of the tip must conform to the surgeon's controls but movement of the body will be a function of positioning constraints, interstitial tissue limitations, and the need to generate force in useful vectors.

Inflatable Collar

Cumulative flex along the snakebot body presents an issue for surgeons wishing a stable platform from which to perform delicate surgeries. One solution is to add an inflatable collar to the snakebot which, when inflated in situ, would provide stability to distal segments. Multiple collars along the snakebot length could anchor the snakebot in situ for work at the tip requiring more force or precision.

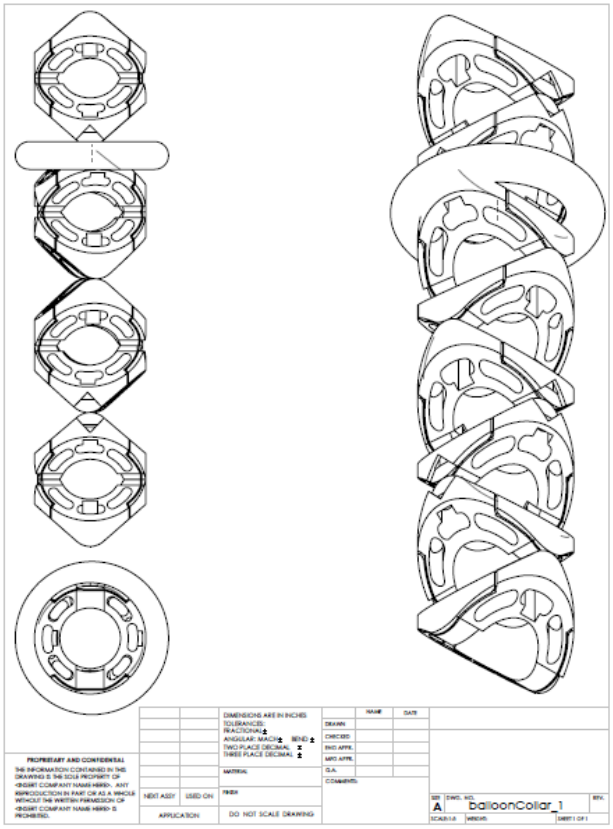


Figure 58 - Snakebot Balloon Collar

Materials Choices for Small-Diameter Endoscope

Implementing the design in a non-ferromagnetic alloy to allow for use in MRI machinery and at a size useful to surgeons remains a challenge. Sintered titanium and copper (TiCu) alloys presents a cost effective manufacturing possibility. The payload of Snakebot Version 12 is 25% of the snakebot's axial surface area, which will be increased in subsequent iterations by thinning segment walls. Snakebot stability decreases in proportion to decreases of wall thickness, a tradeoff that will be further defined as the prototype is implemented at its target dimensions and of structural material.

Snakebot-Tip Vision

Routing of fiber-optic cable within the segment walls will allow both light and vision at the robot's tip. Since the tip is a ring, a form of raster-scanning can be implemented. By arranging multiple fibers oriented axially at the tip of the robot and post-processing successive 'snapshots', a higher density image can be generated (8).

Miscellaneous Improvements

A ring needs to be added at the proximal end to surround the snakebot to ensure that only that portion of the endoscope in situ is subject to force-flex. More segments must be added to the prototype to further explore serpentine motion. A second snakebot is needed to explore the caduceus paradigm.

Chapter 7

Conclusions

The prototype demonstrates a new endoscope design offering a strong platform from which the surgeon can manipulate grippers and scalpels, a hollow payload so tools can be replaced in situ, and the ability to navigate complex trajectories. It provides the capability to generate tip-force in any direction and without requiring interstitial tissue to redirect force.

While this prototype demonstrates the utility of a new paradigm of robotic endoscope design, it is not completed, with cable-tension hysteresis issues that prevent deterministic kinematics. The evolutionary design process requires more iterations at a smaller scale and with other materials before any useful product may devolve.

Appendix 1

Robotic Segment-Shape Solution-Space

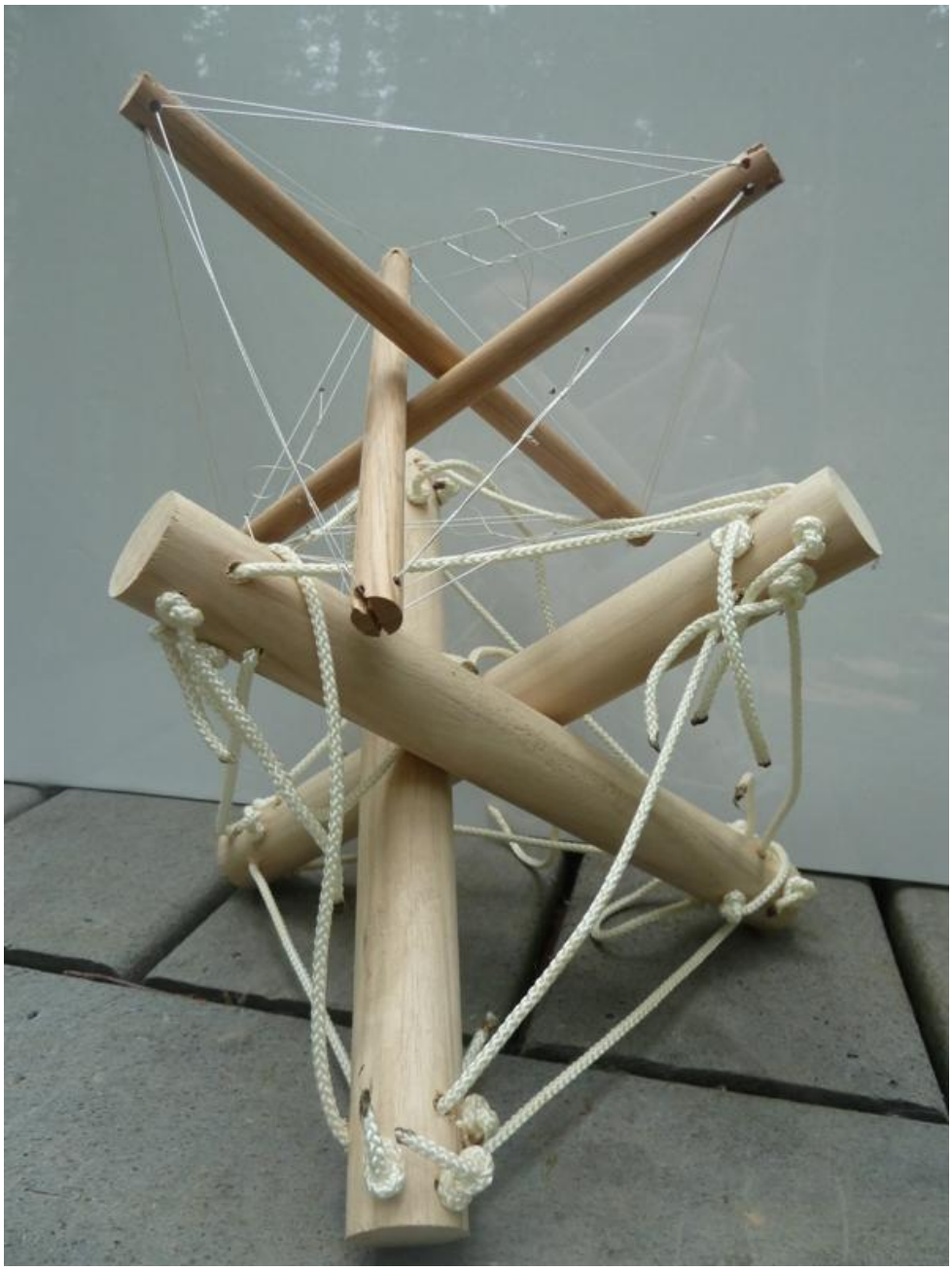


Figure 59 - Tensegrity Prototype

Figure 60 - First Prototype Using Segments

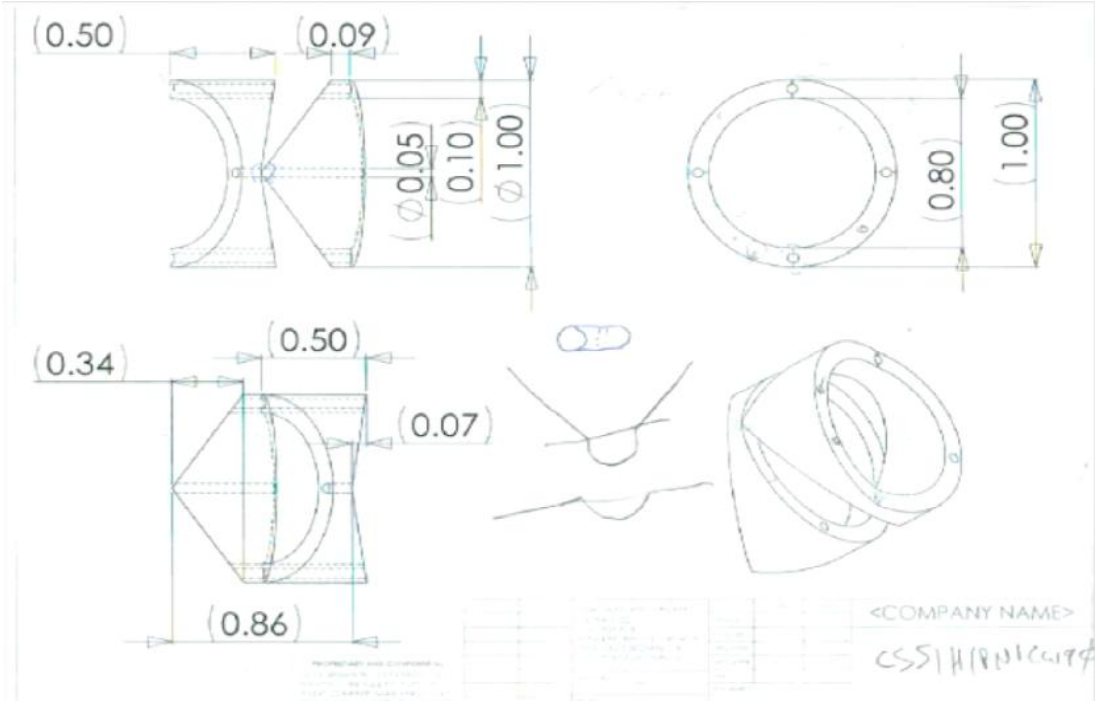


Figure 61 - Segment Design CSS1H1PN1CW90

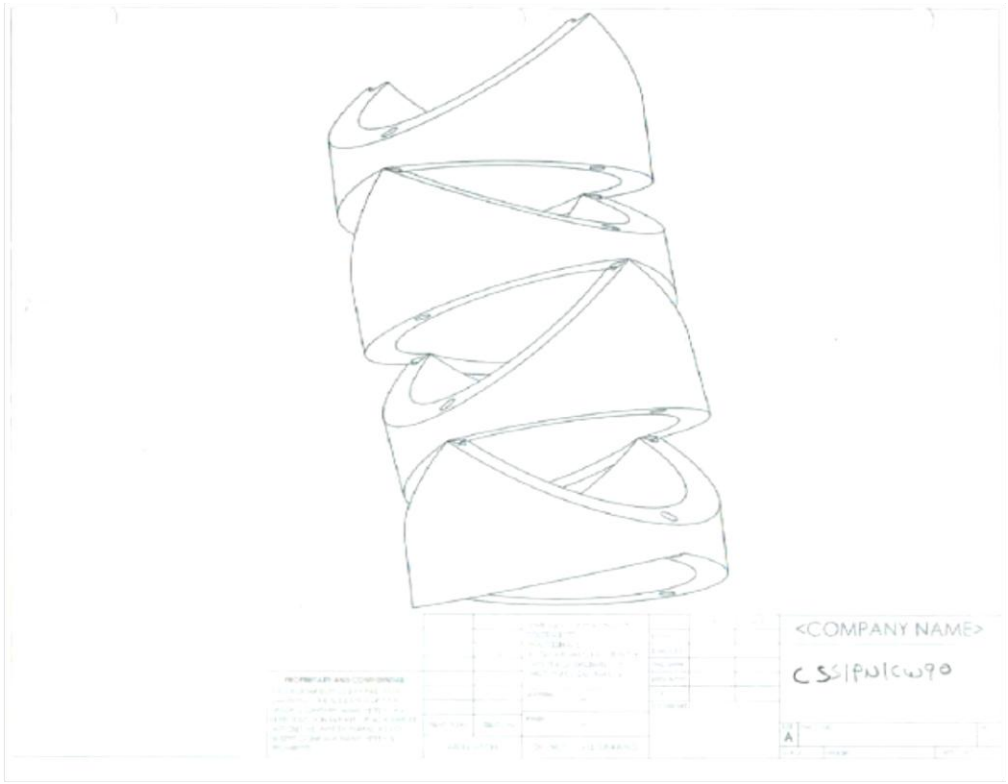


Figure 62 –Segment Design CSS1PN1CW90

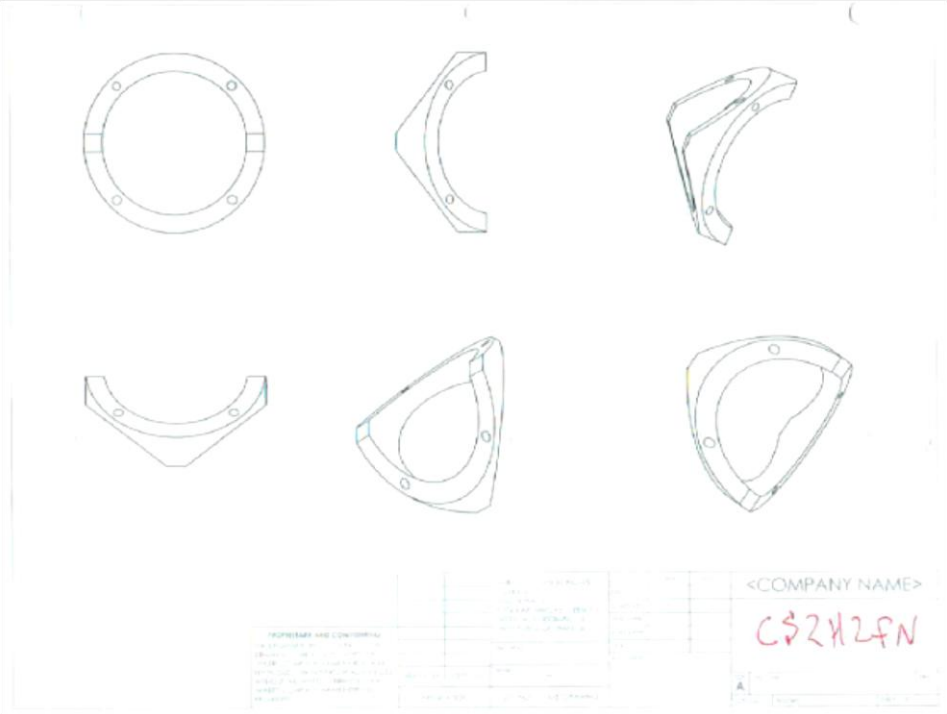


Figure 63 - Segment Design CS2H2FN

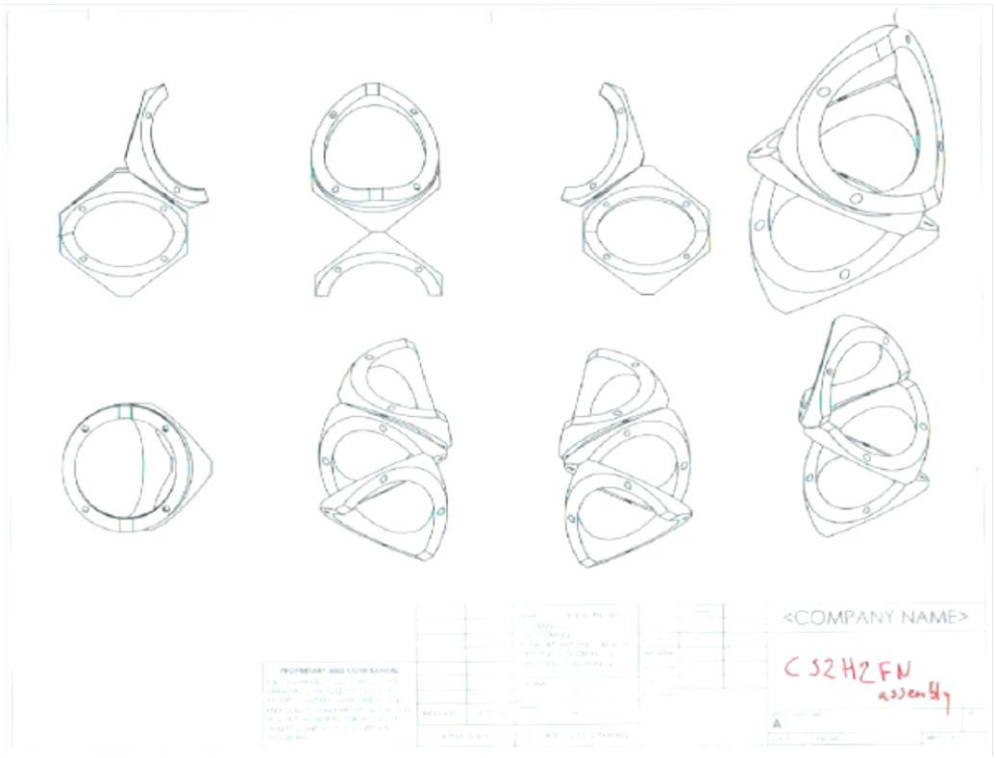


Figure 64 - Segment CS2H2FN Assembly

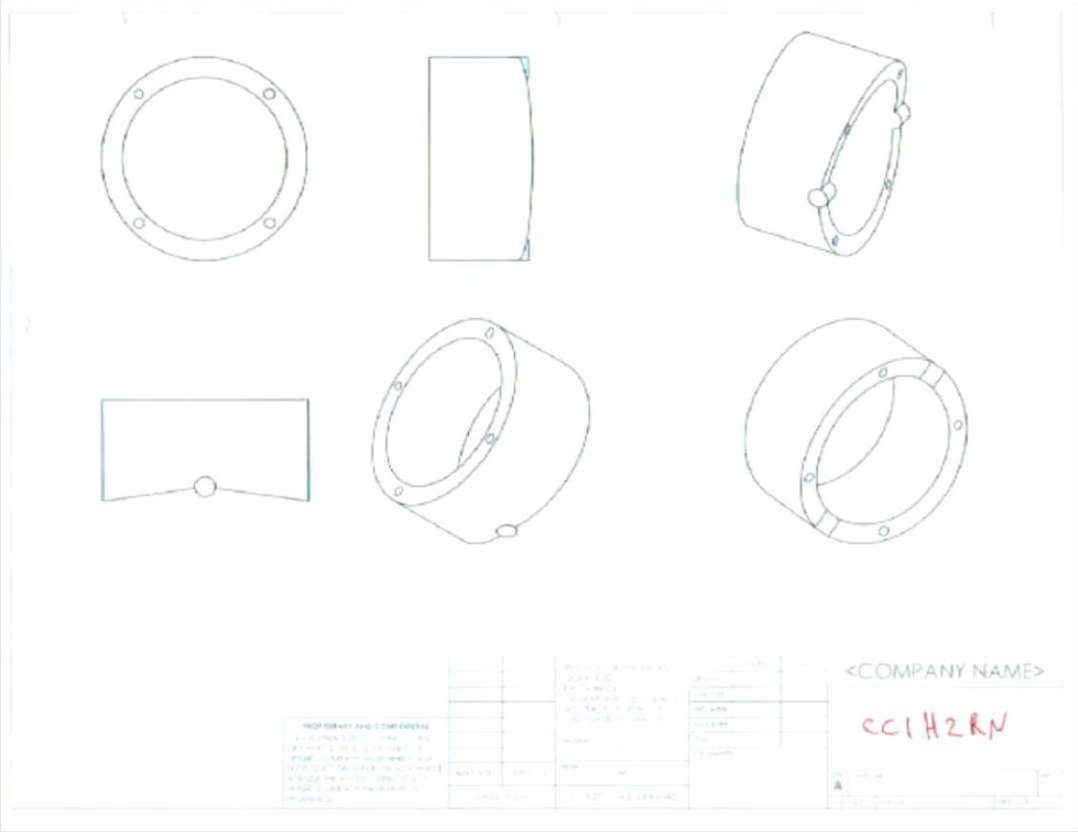


Figure 65 - Segment Design CC1H2RN

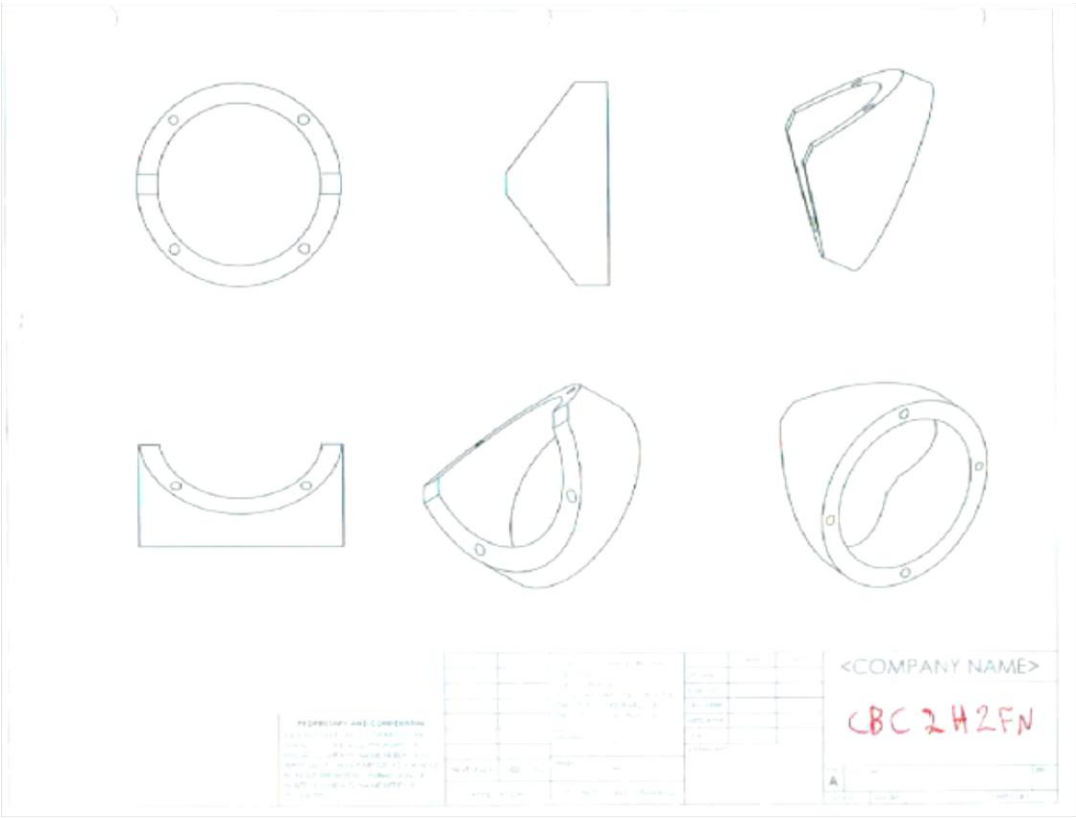


Figure 66 - Segment Design CBC2H2FN

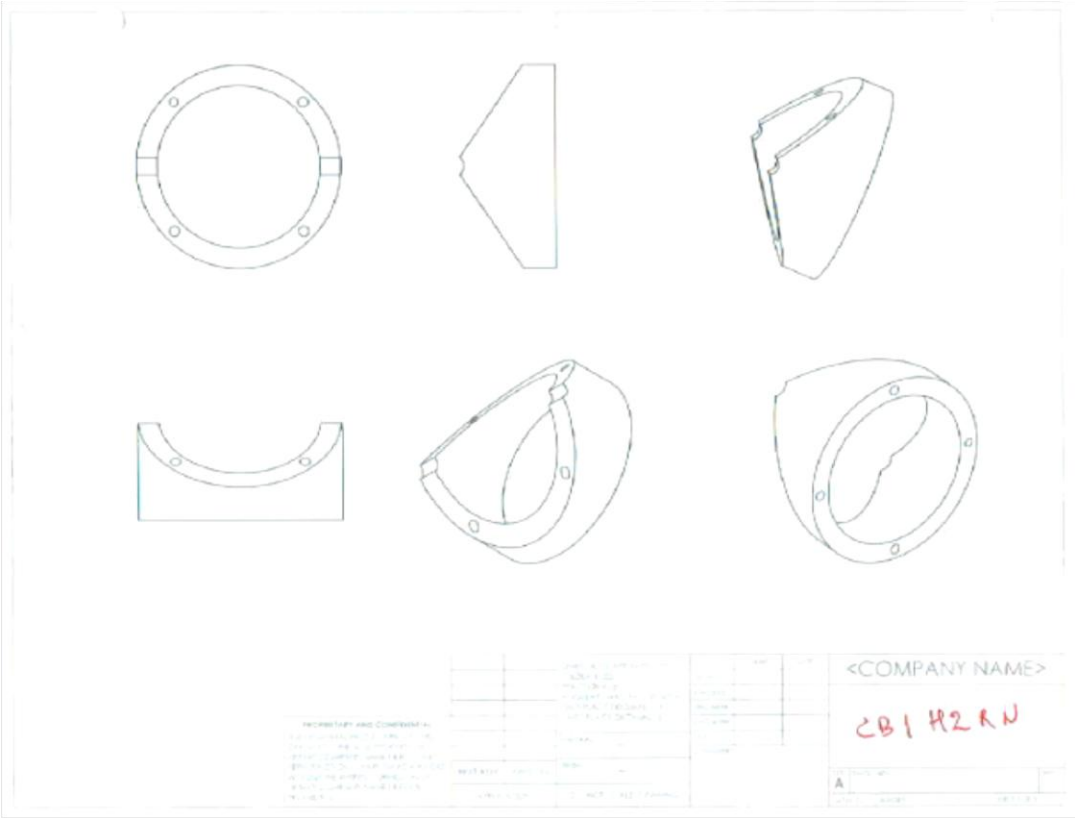


Figure 67 - Segment Design CB1H2RN

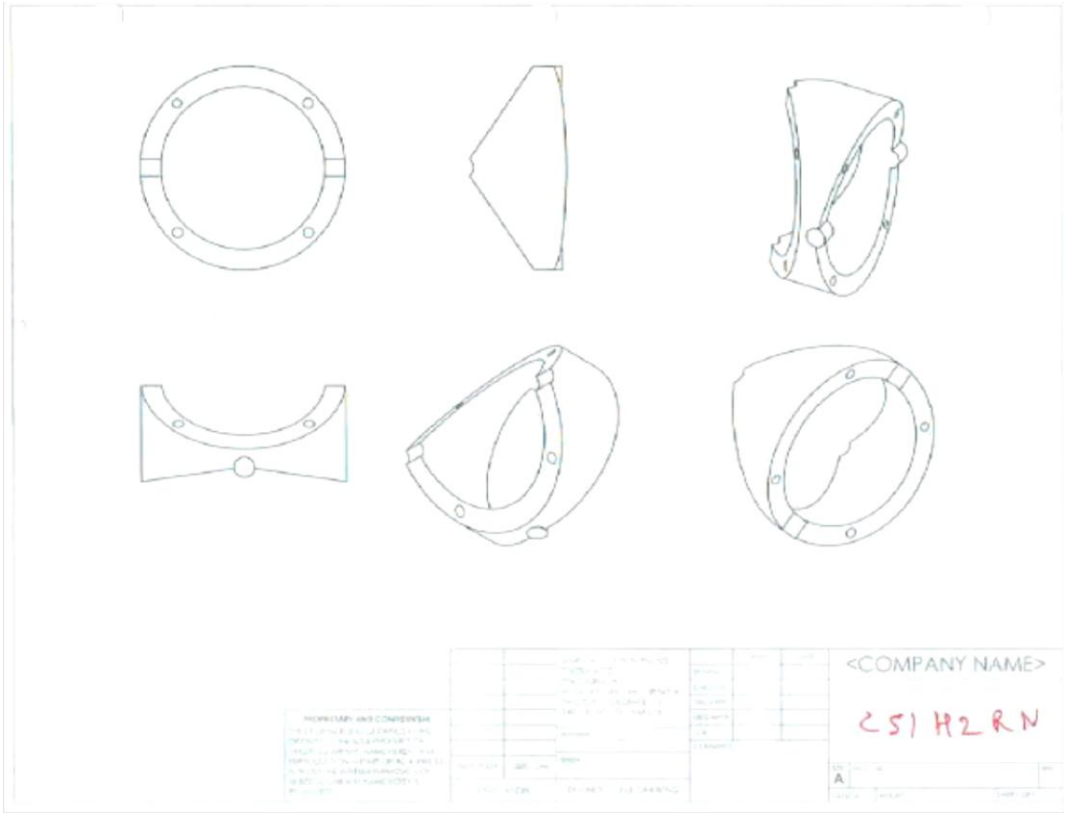


Figure 68 - Segment Design CS1H2RN

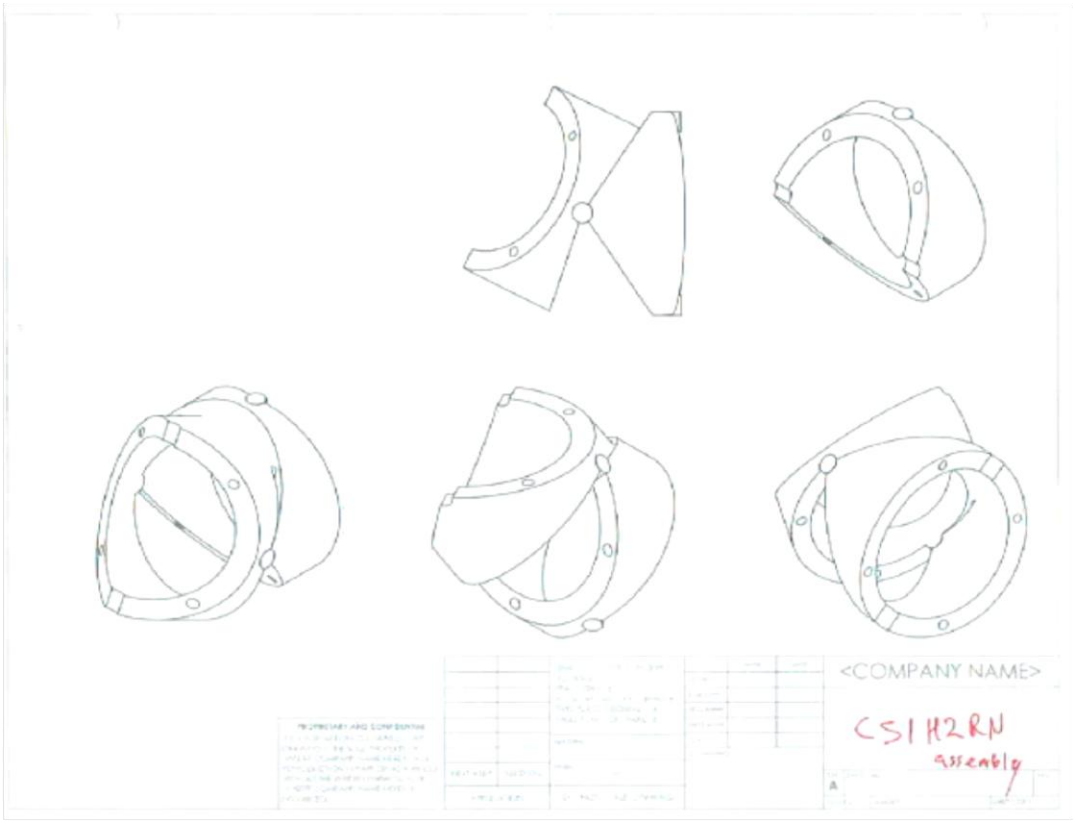


Figure 69 - Segment CS1H2RN Assembly

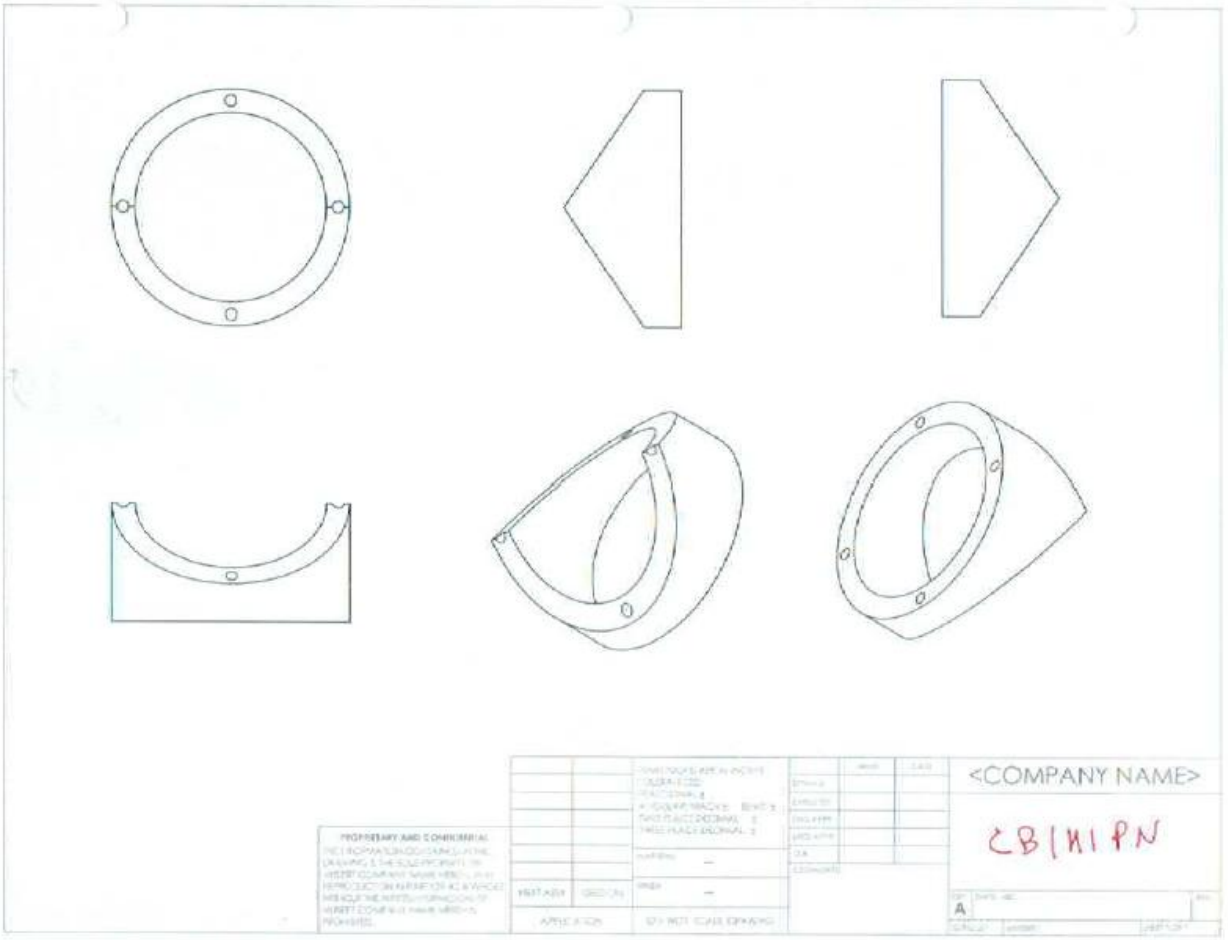


Figure 70 - Segment Design CB1H1PN

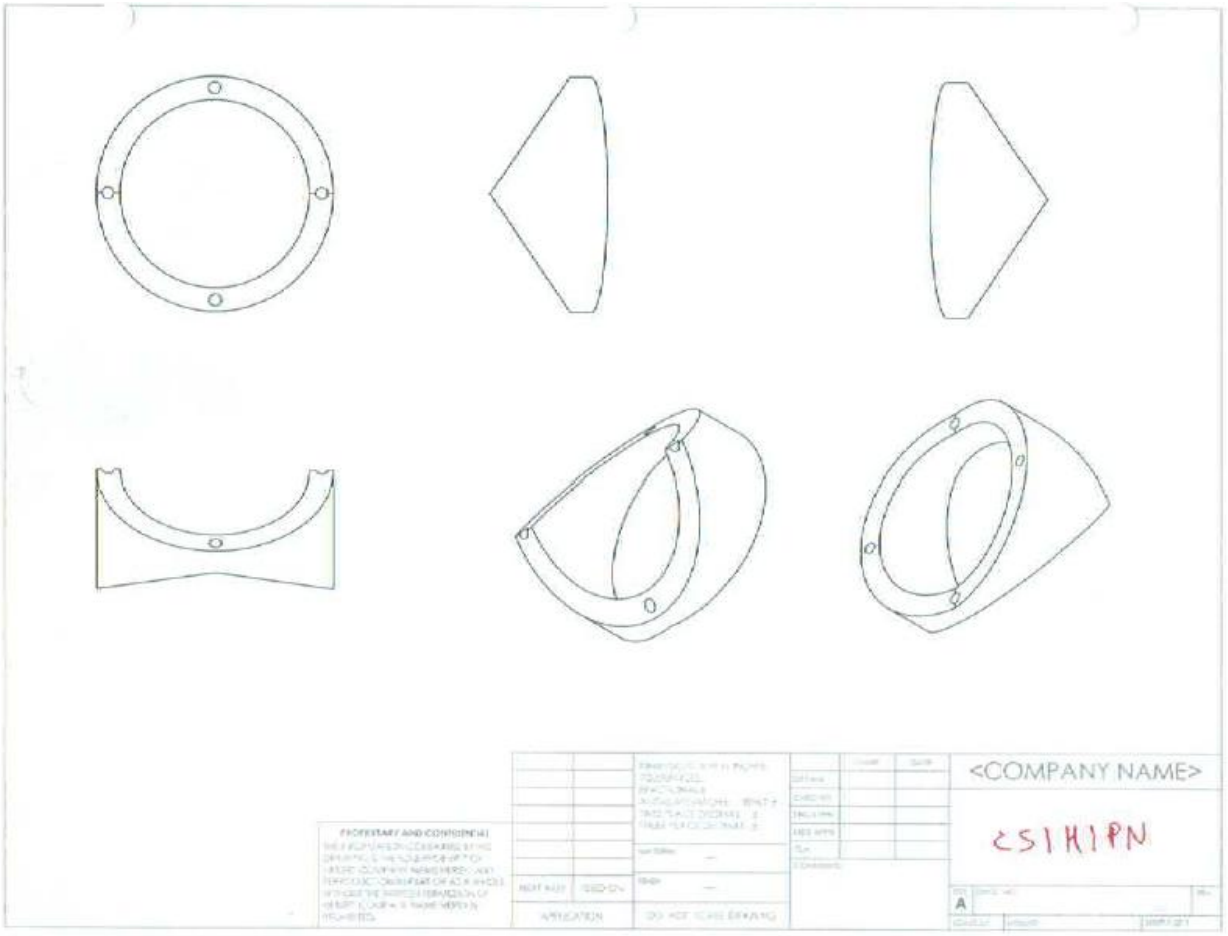


Figure 71 - Segment Design CS1H1PN

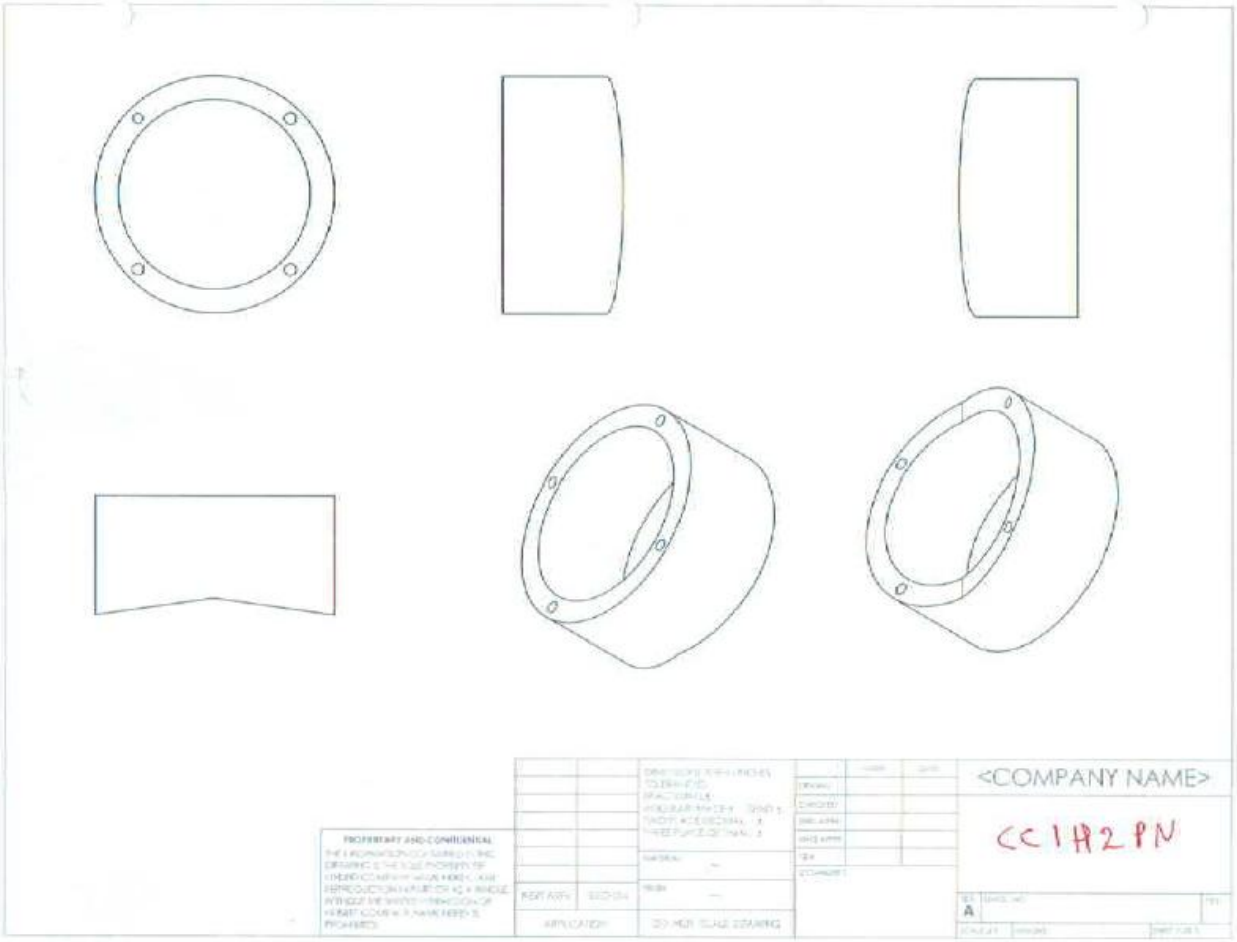


Figure 72 - Segment Design CC1H2PN

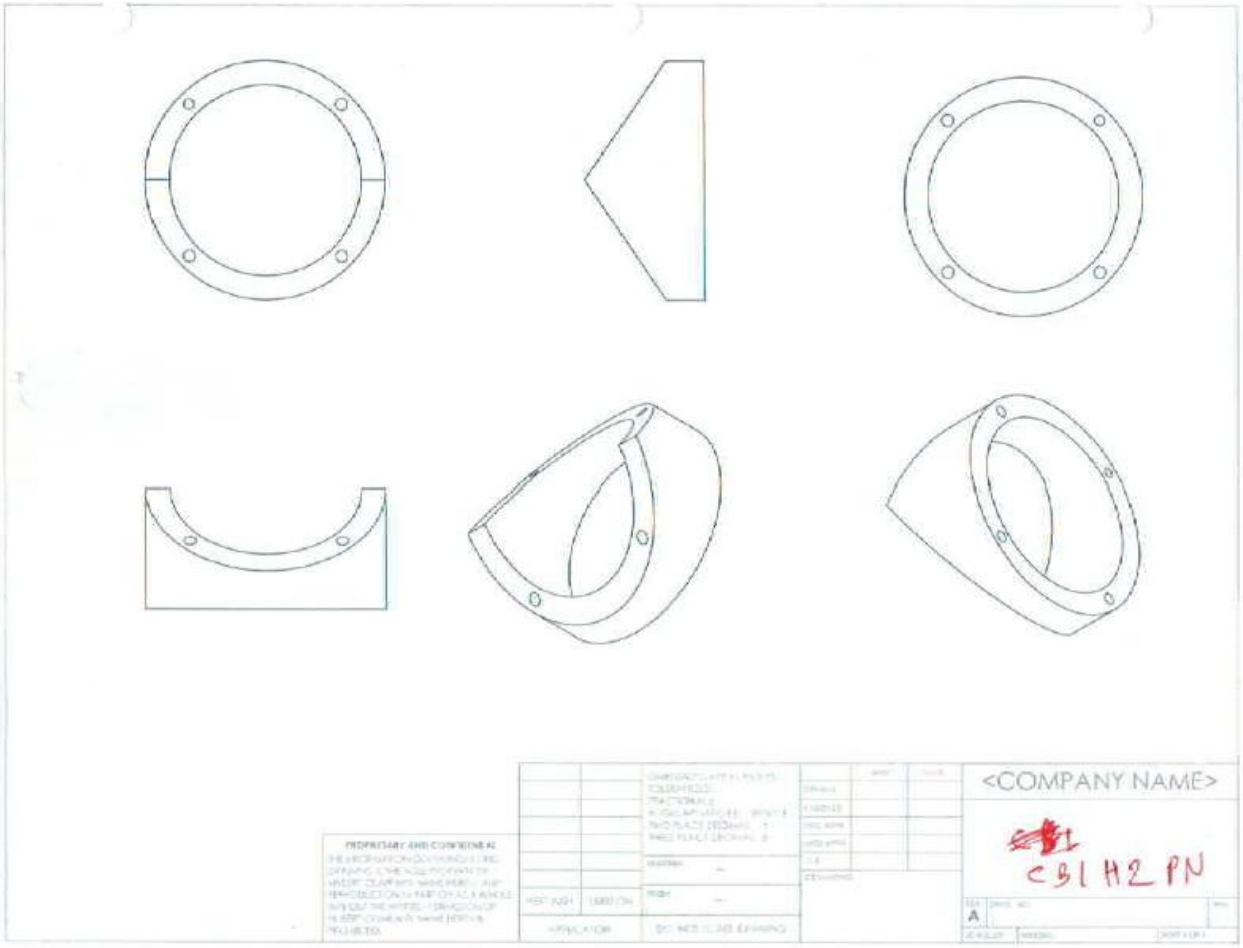


Figure 73 - Segment Design CB1H2PN

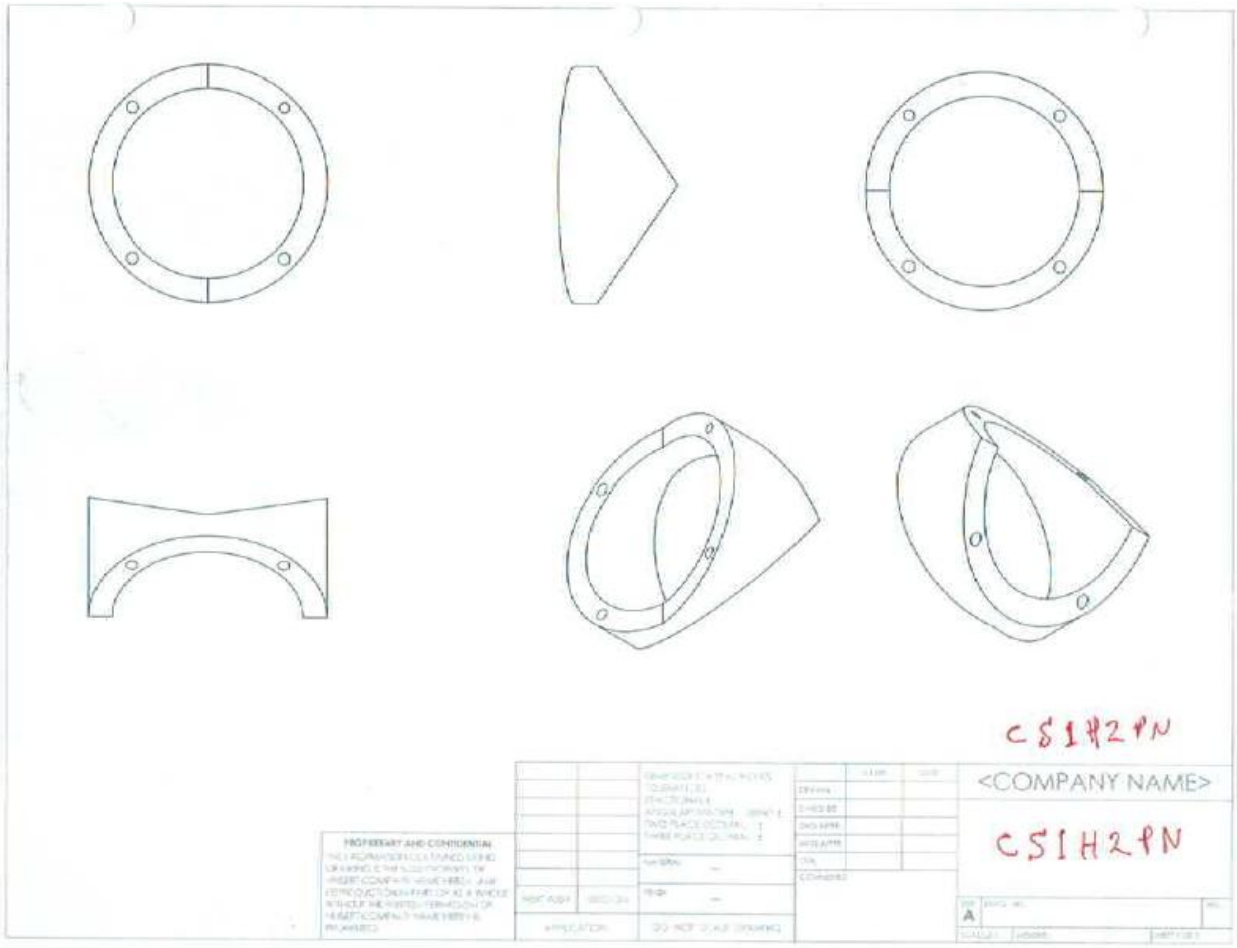


Figure 74 - Segment Design CS1H2PN

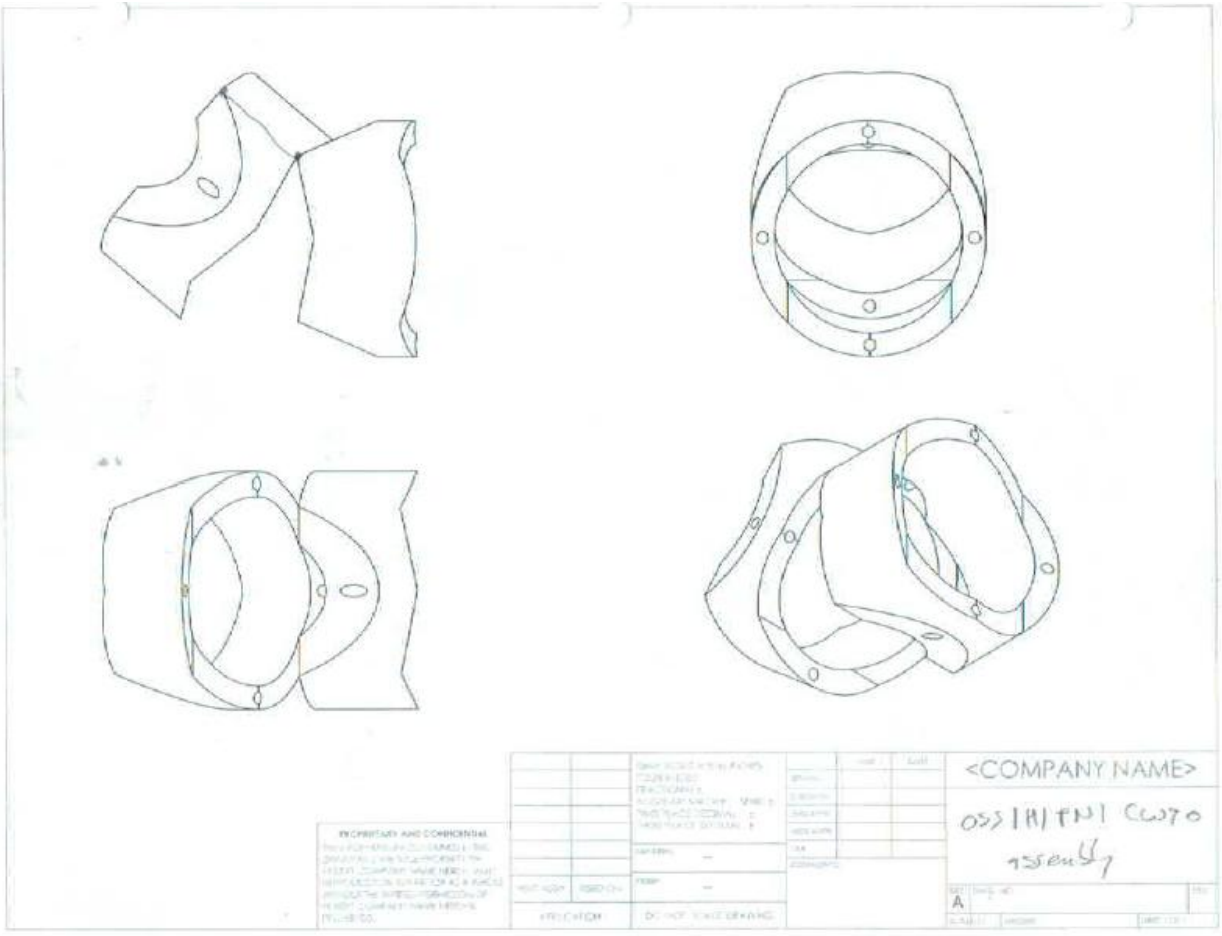


Figure 75 - Segment OSS1H1PN1CS90 Assembly 1

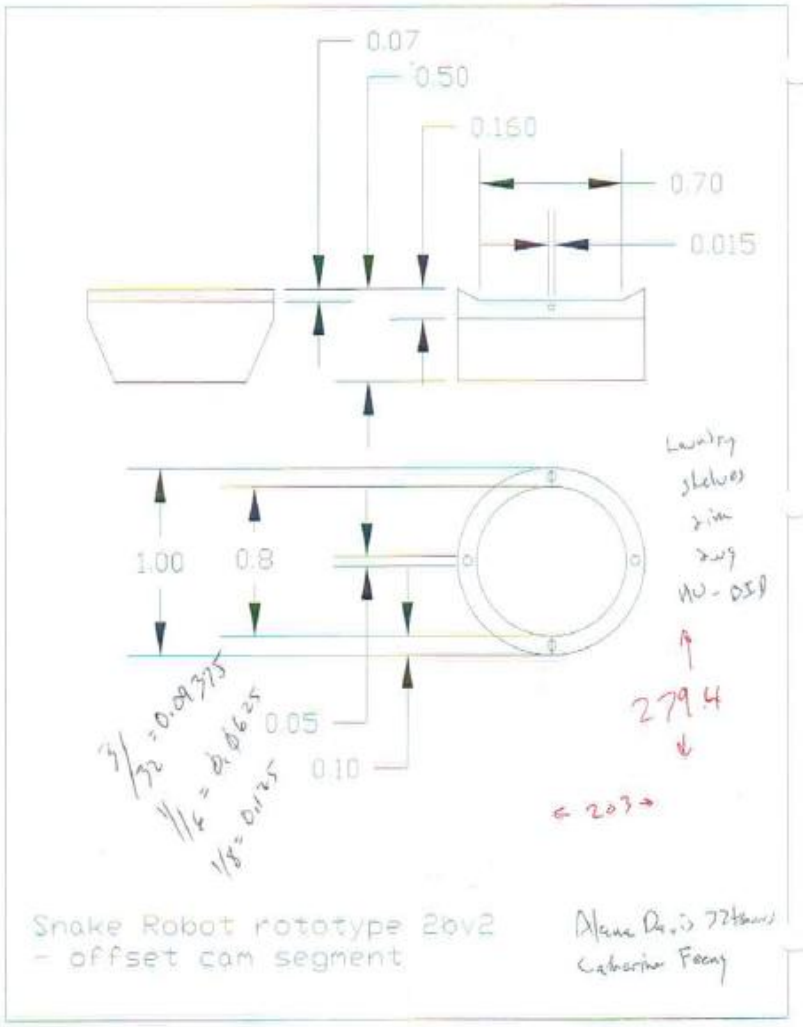


Figure 76 - Segment Design 2BV2

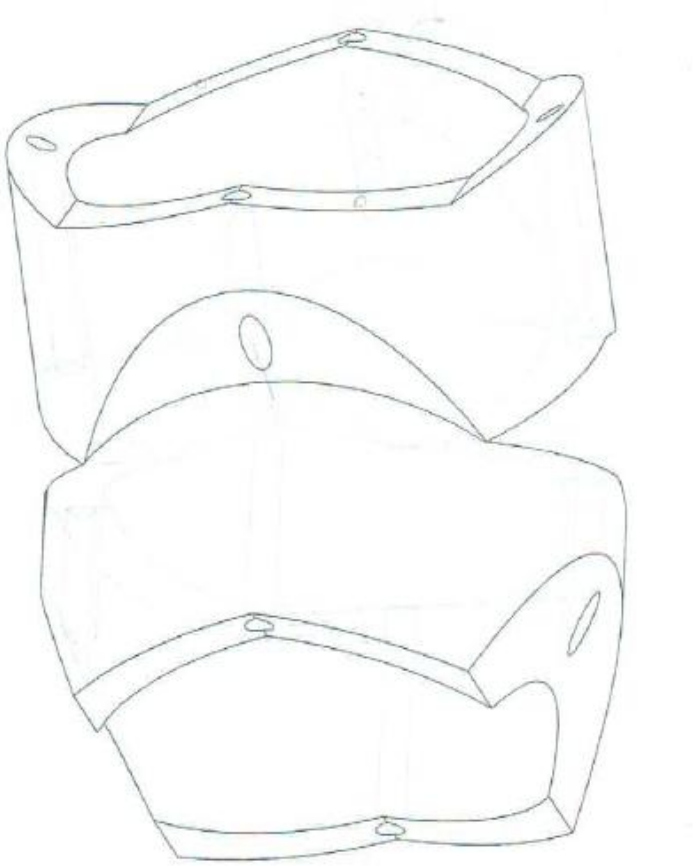


Figure 77- Segment OSS1H1PN1CS90 Assembly 2

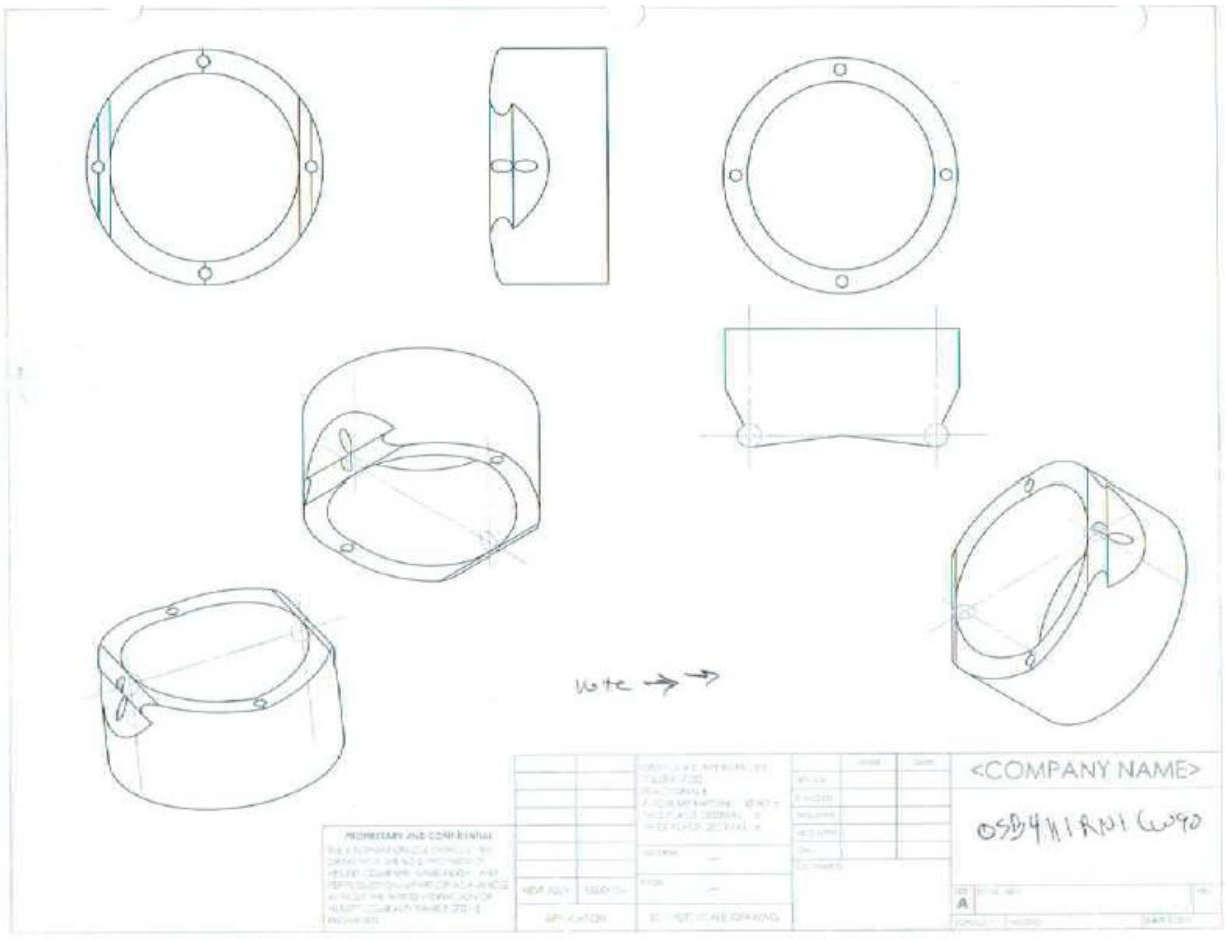


Figure 78 - Segment Design OSB4H1RN1CW90

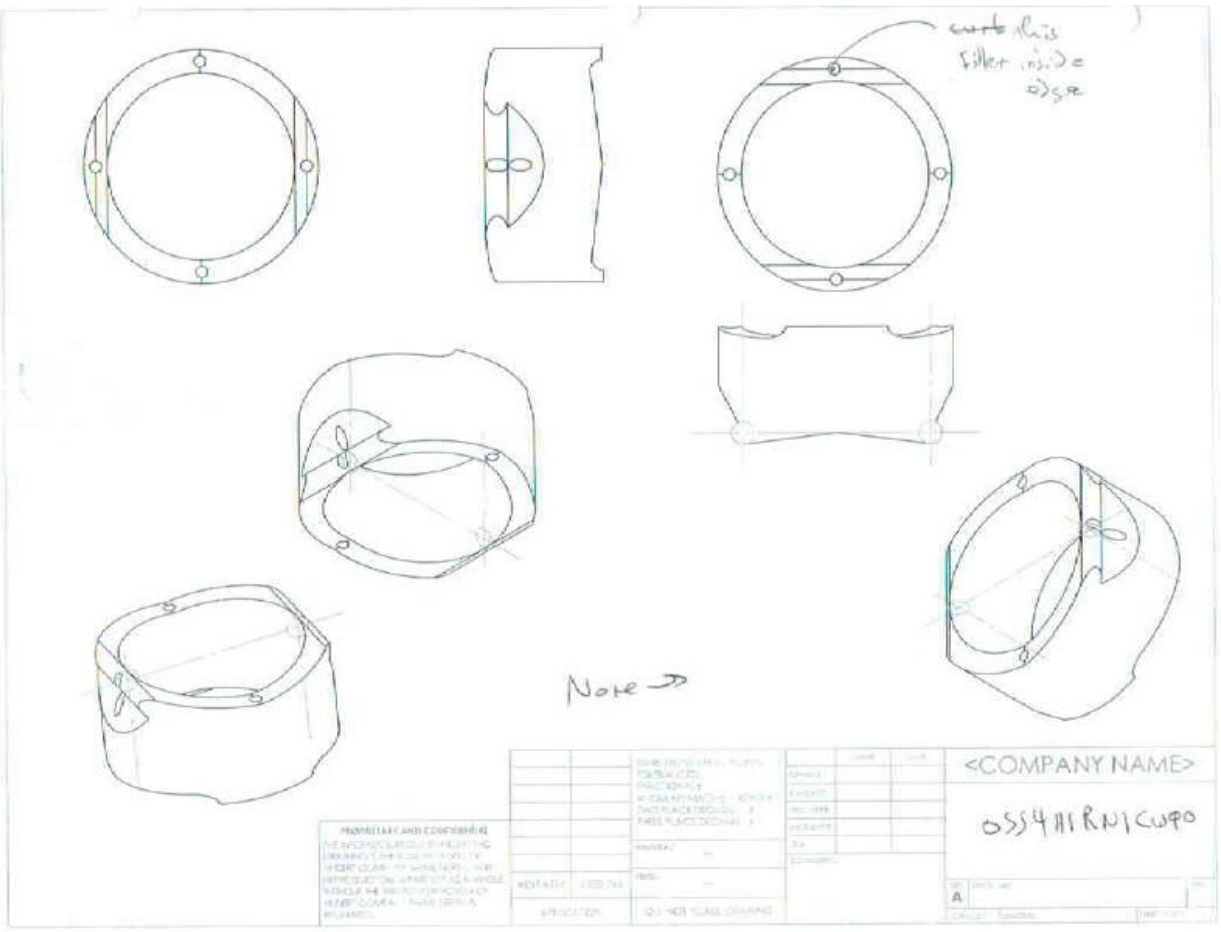


Figure 79 - Segment Design OSS4H1RN1CW90

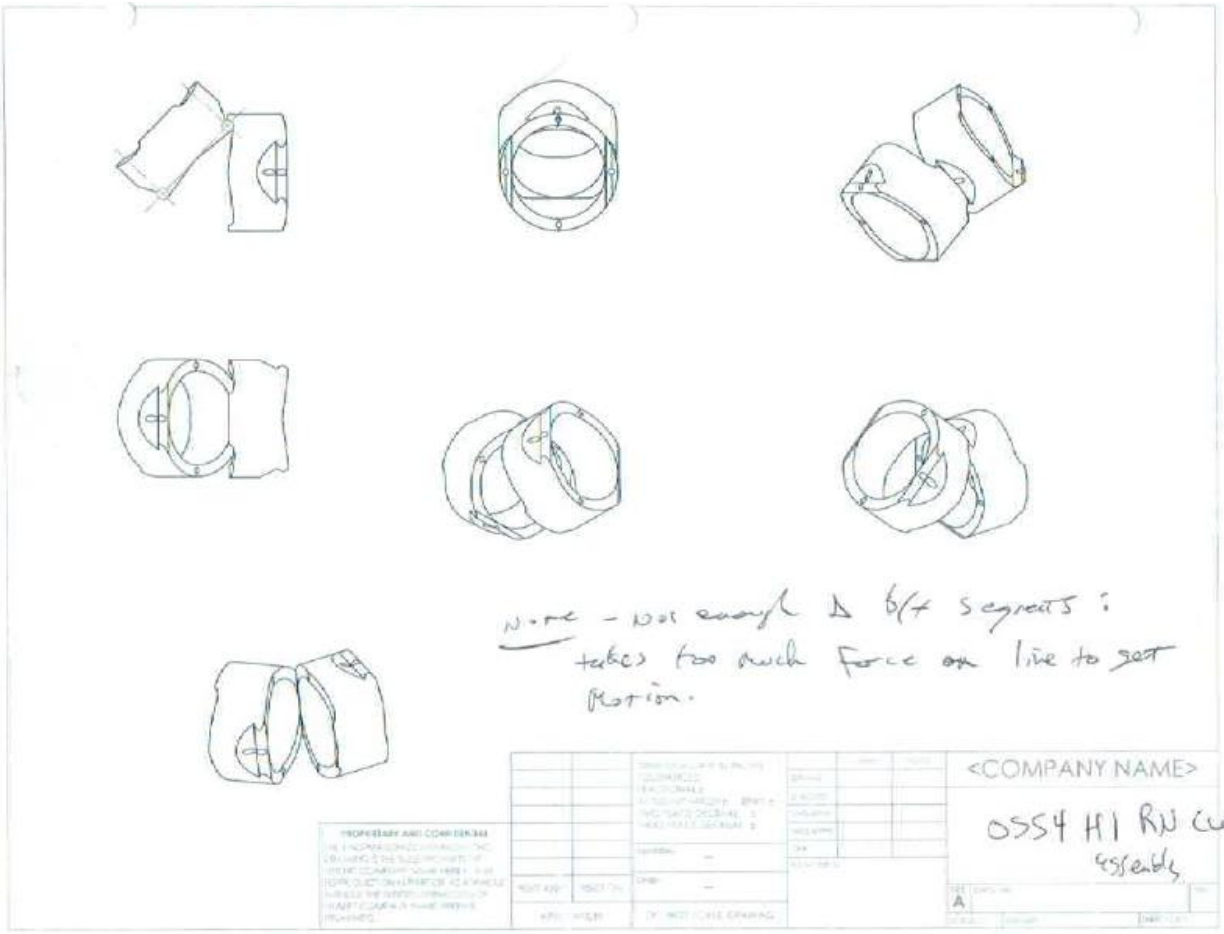


Figure 80 - Segment OSS4H1RNCW90 Assembly

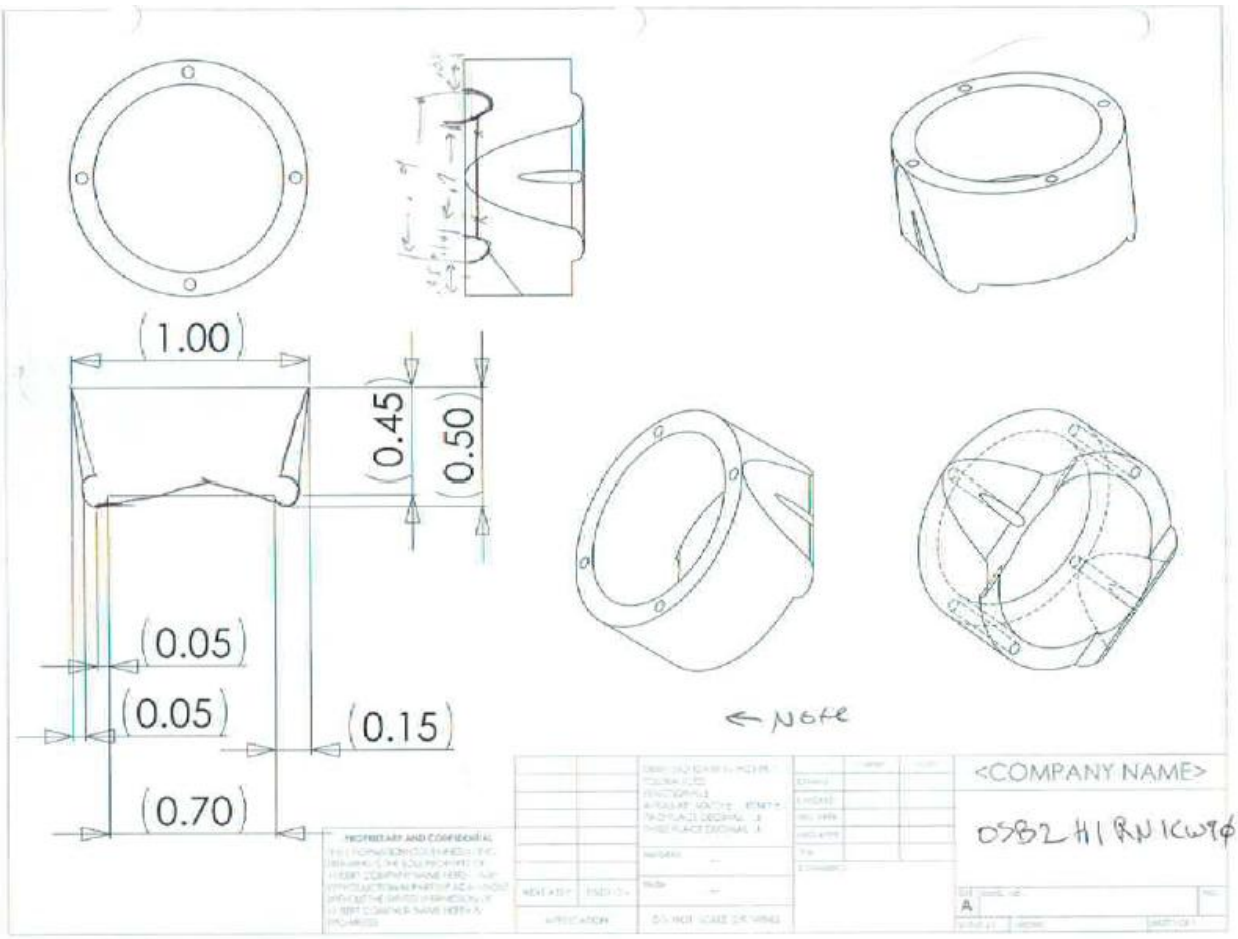


Figure 81 - Segment Design OSB2H1RN1CW90

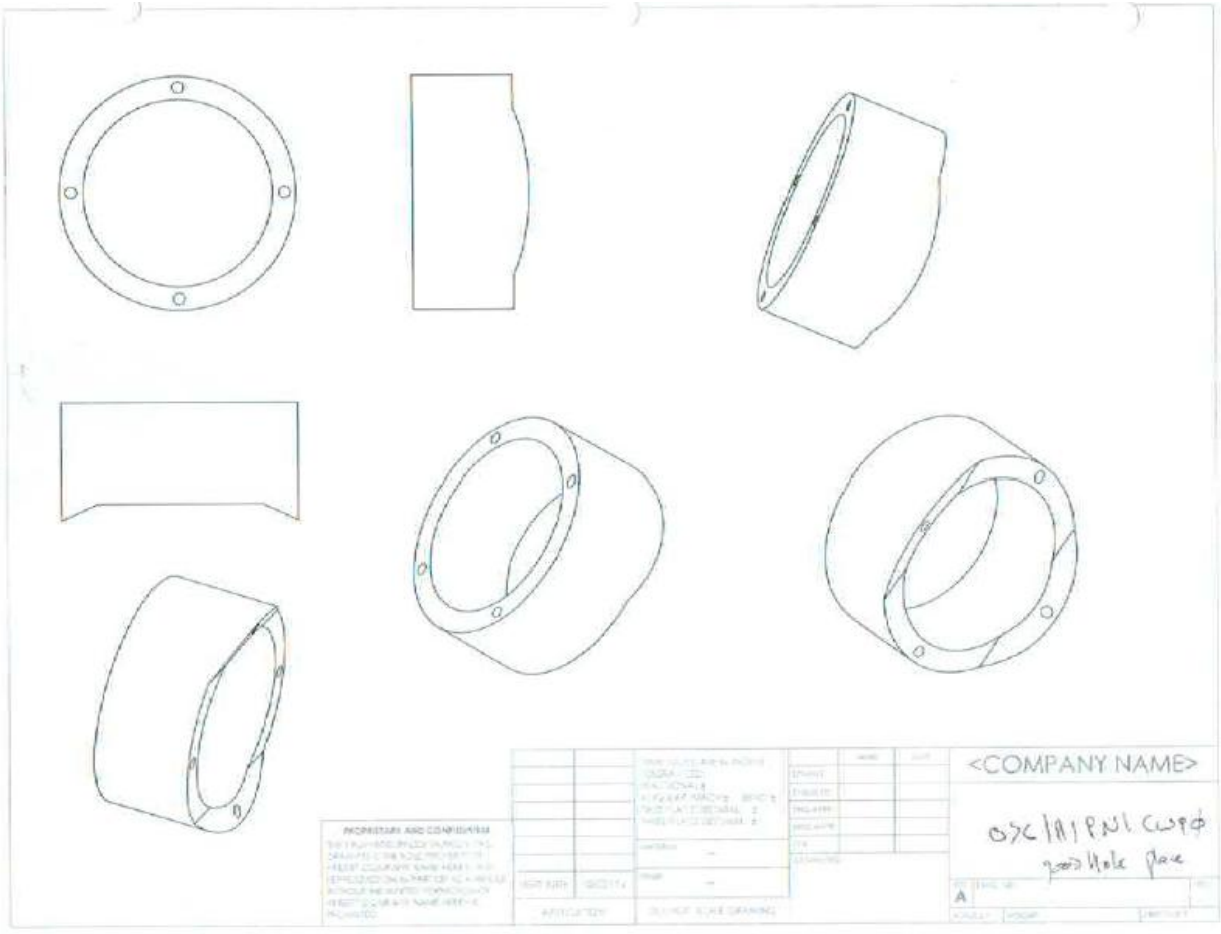


Figure 82 - Segment Design OSC1H1PN1CW90

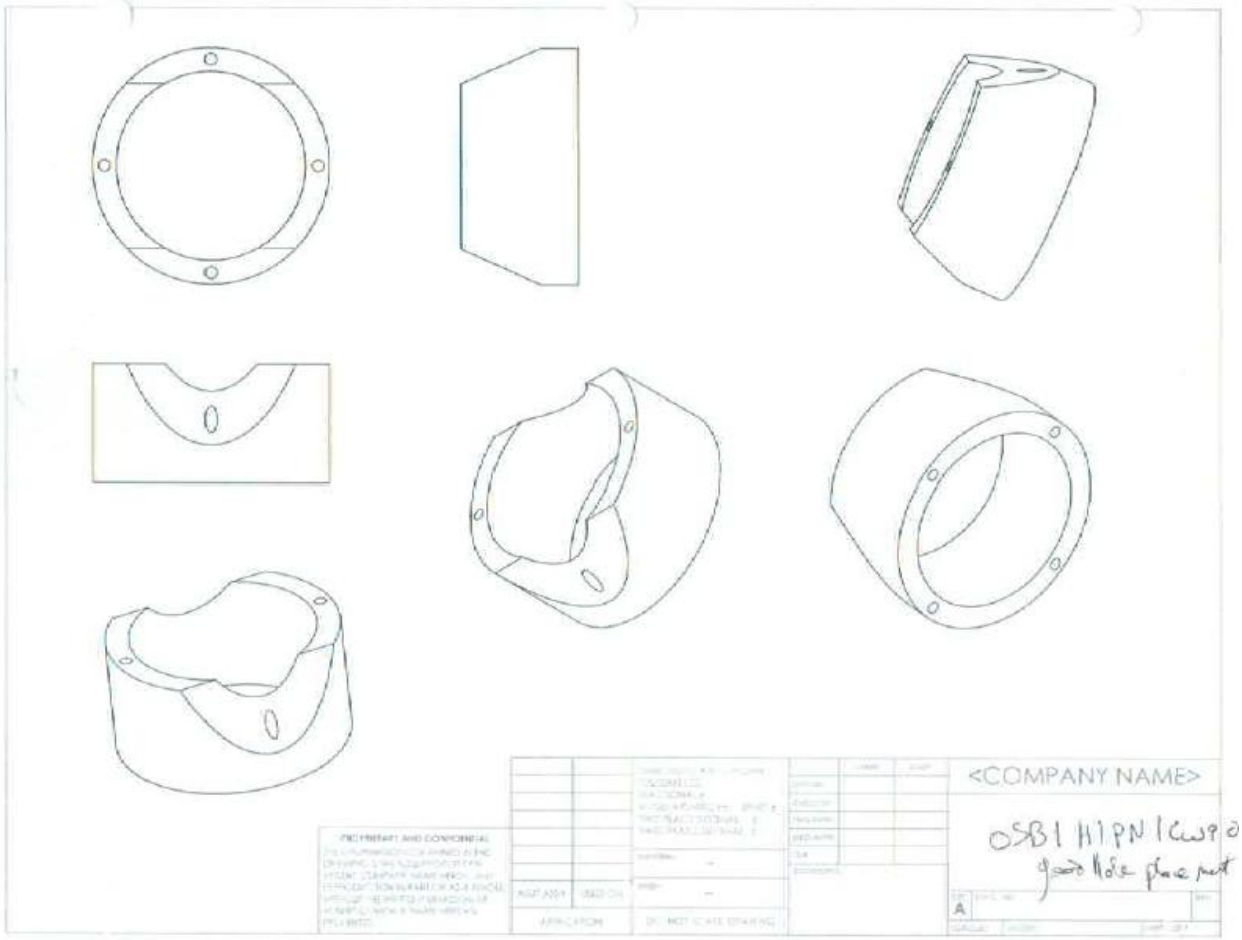


Figure 83 - Segment Design OSB1H1PN1CW90

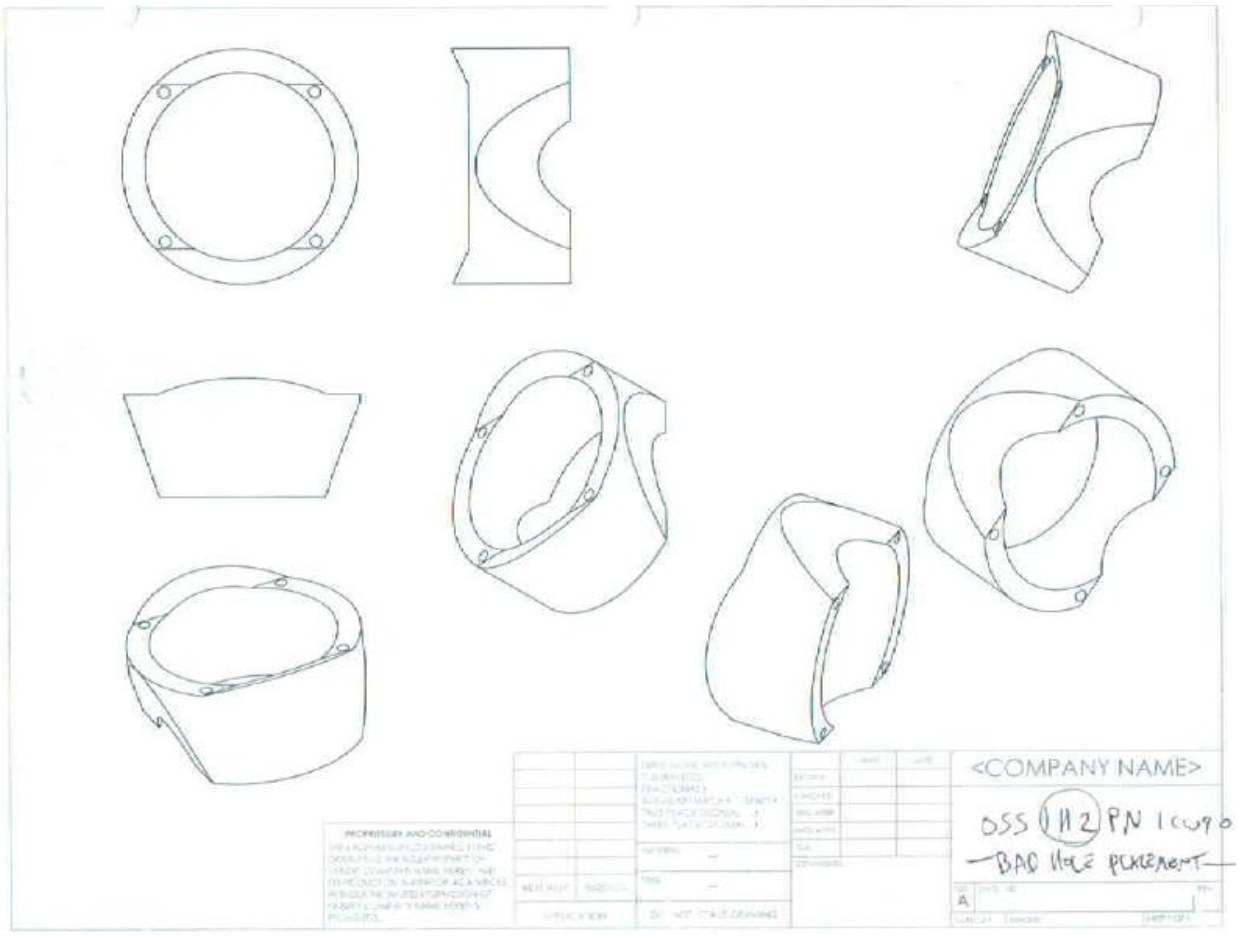


Figure 84 - Segment Design OSS1H2PN1CW90

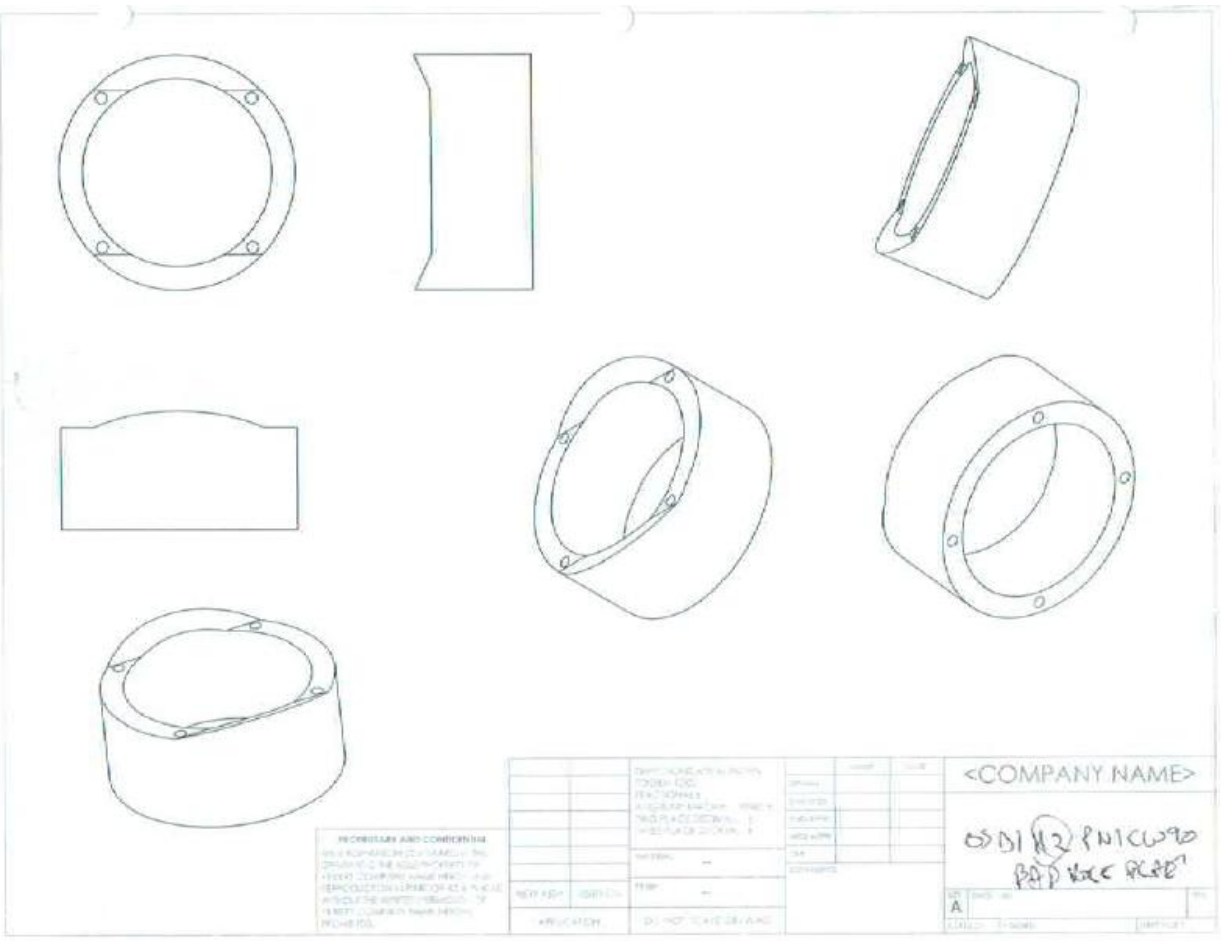


Figure 85 - Segment Design OSB1H2PN1CW90

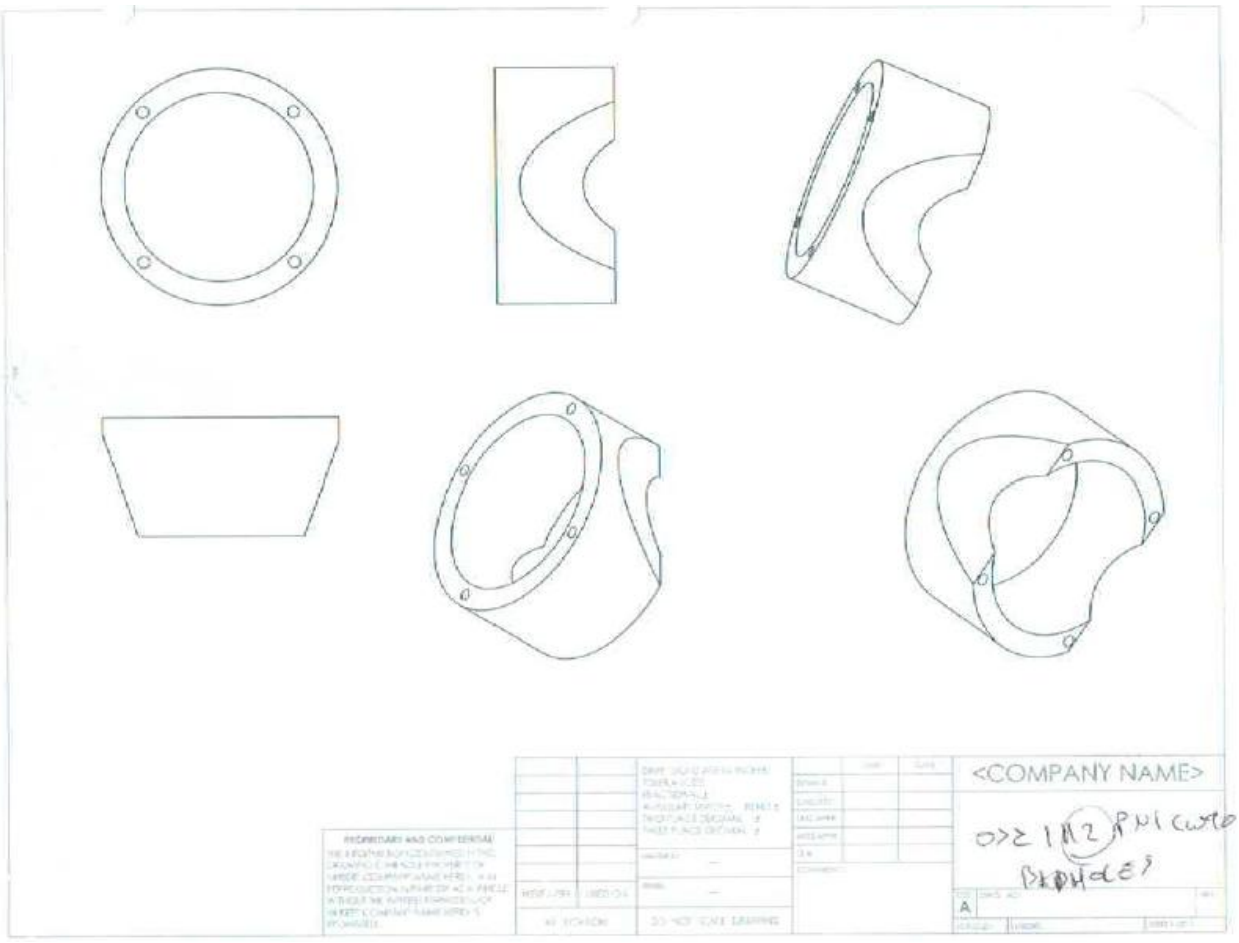


Figure 86 - Segment Design OSC1H2PN1CW90

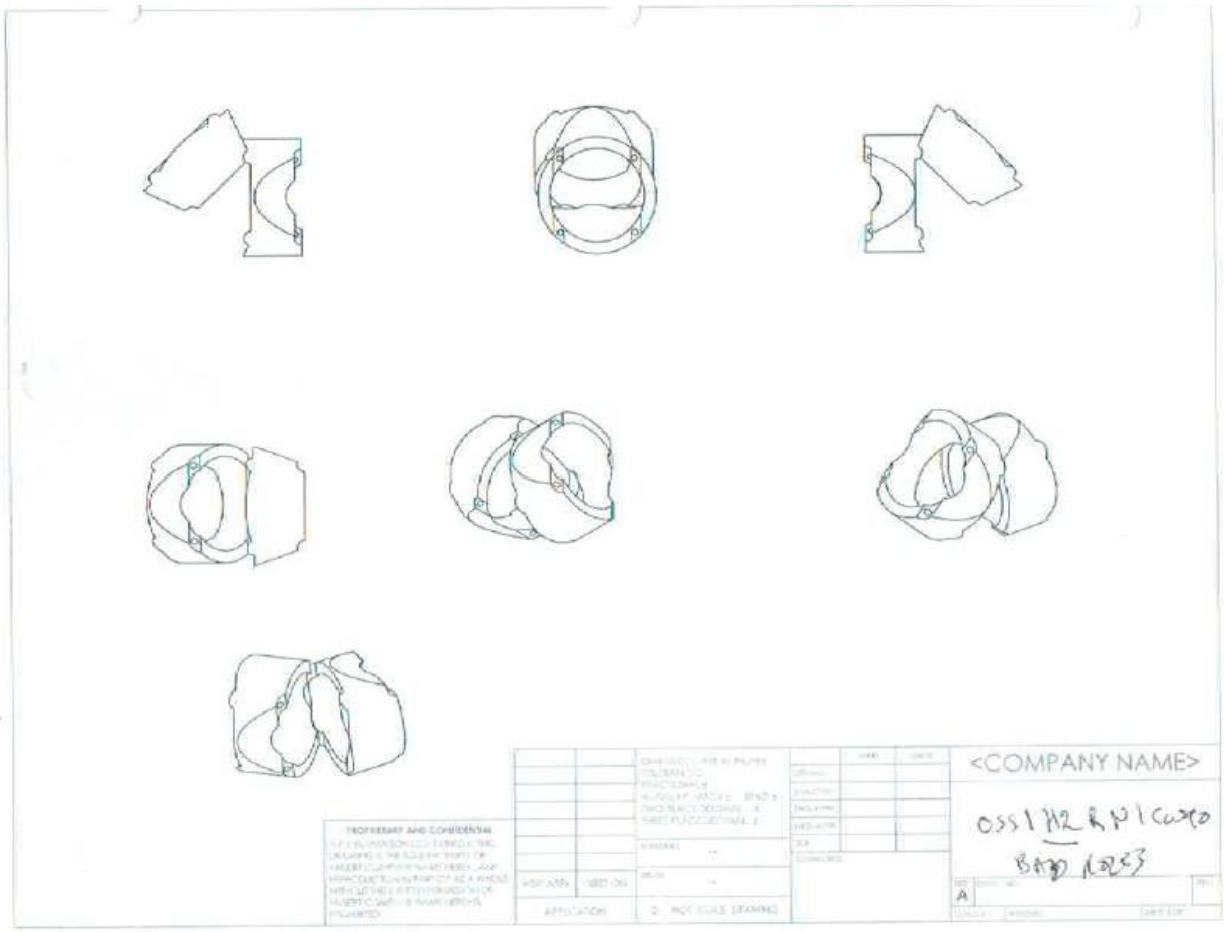


Figure 87 - Segment OSS1H2RN1CW90 Assembly

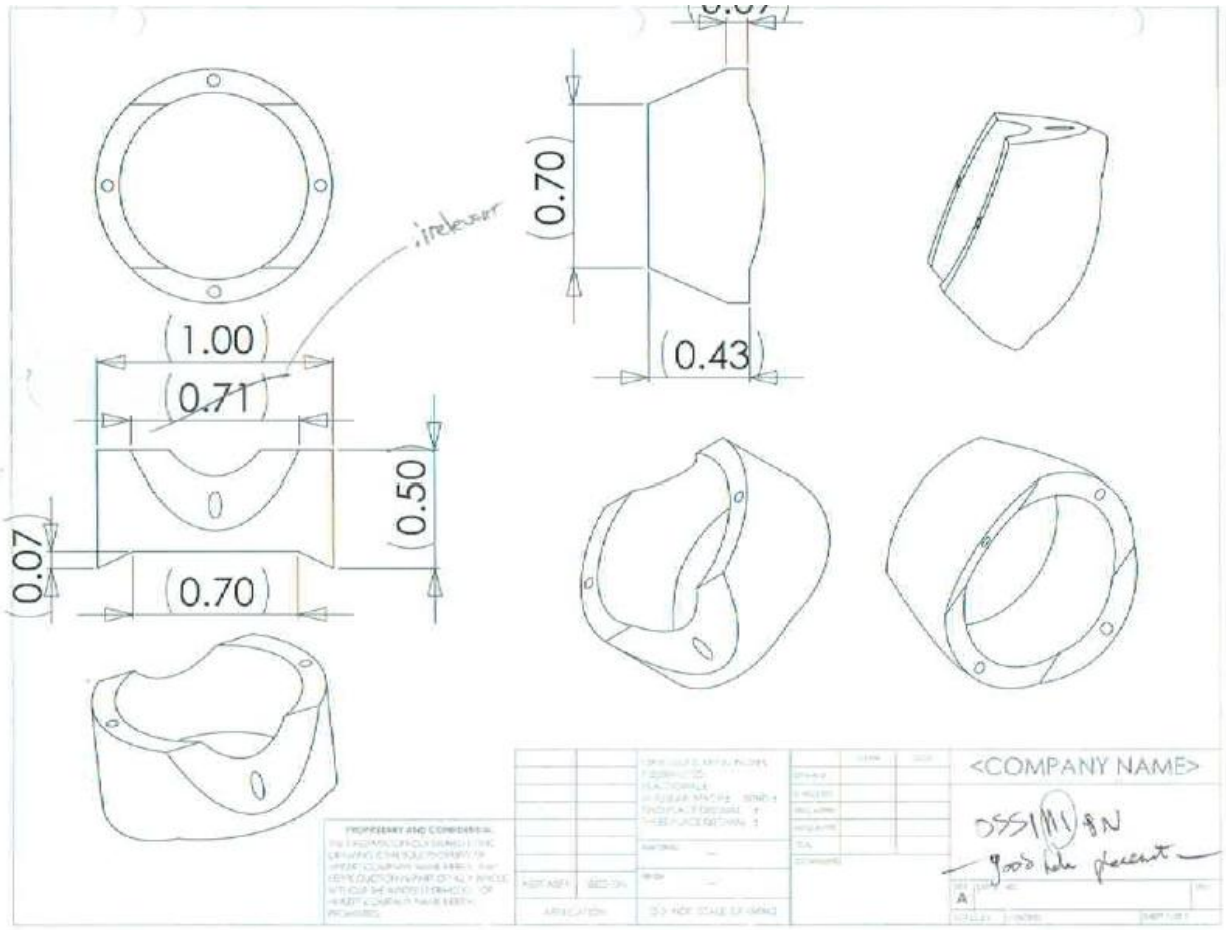


Figure 88 - Segment Design OSS1H1PN

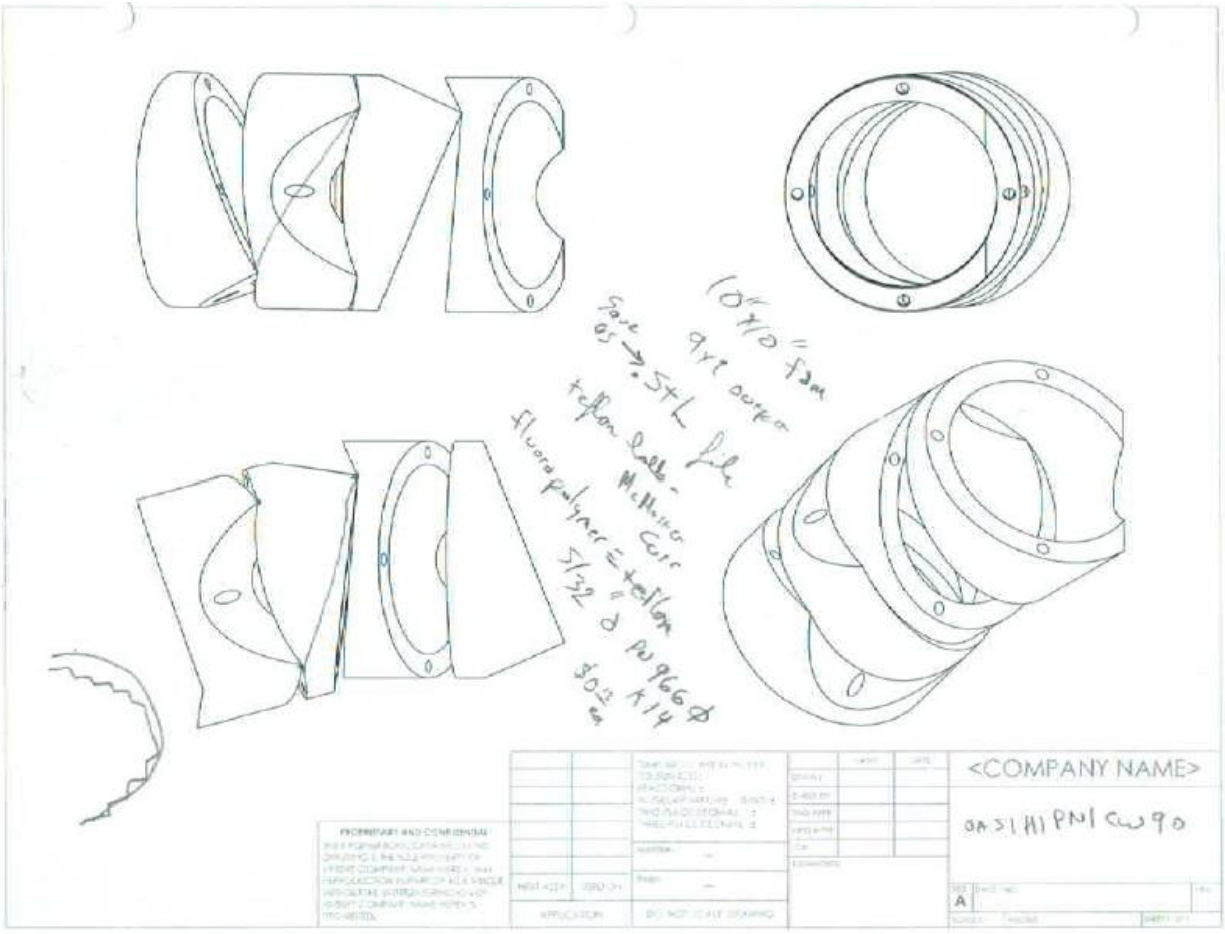


Figure 89 - Segment OAS1H1PN1CW90 Assembly

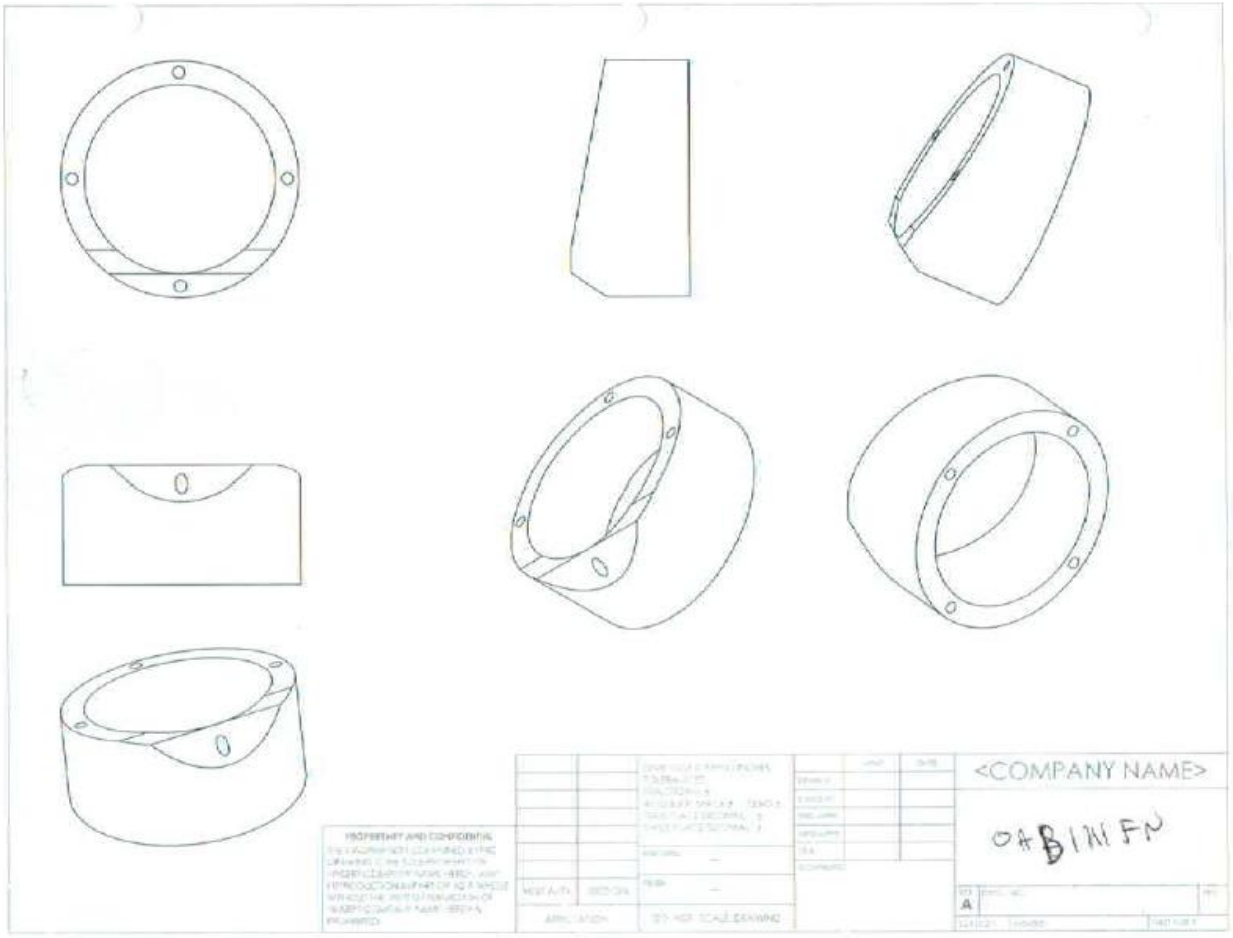


Figure 90 - Segment Design OAB1H1FN

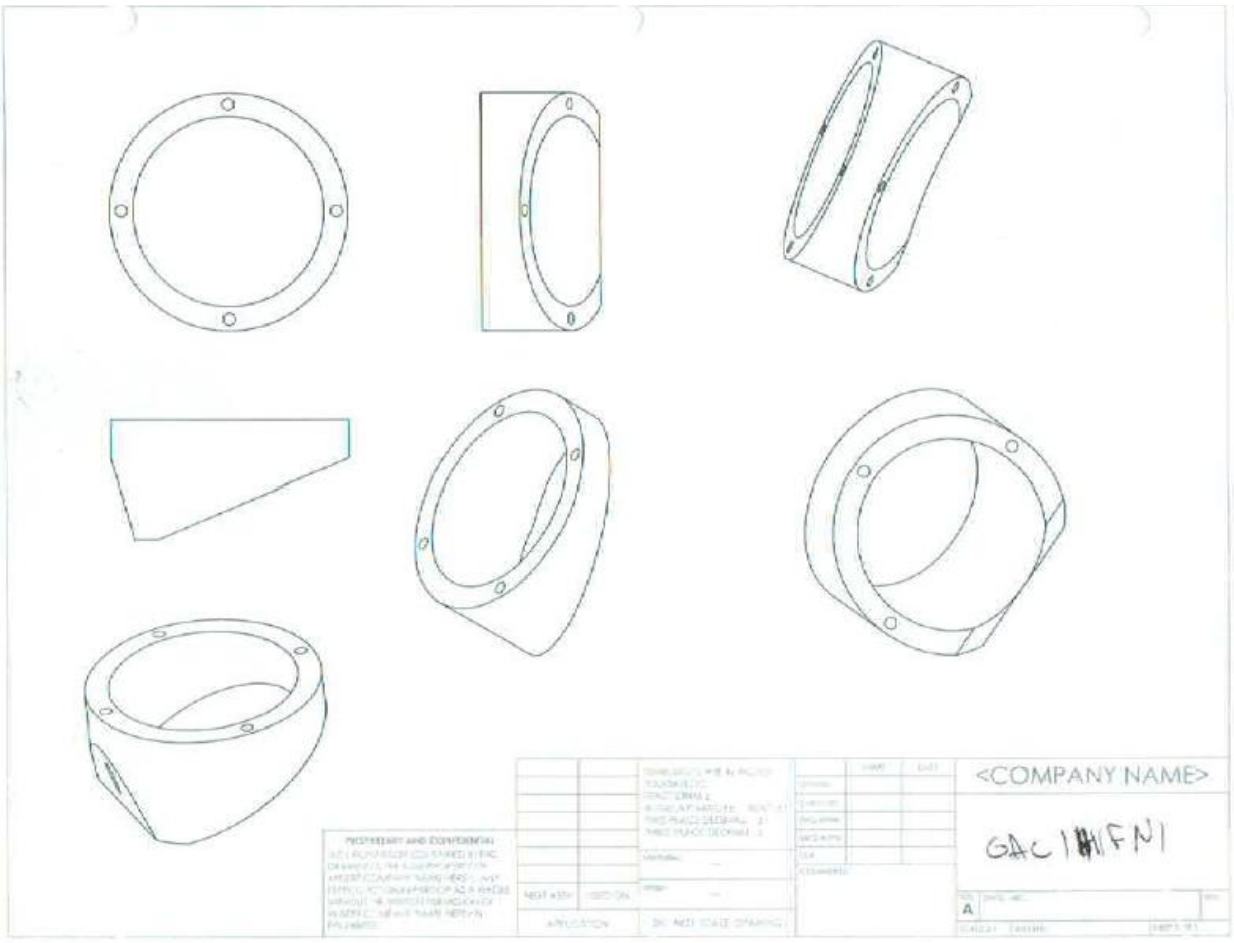


Figure 91 - Segment Design OAC1H1FN1

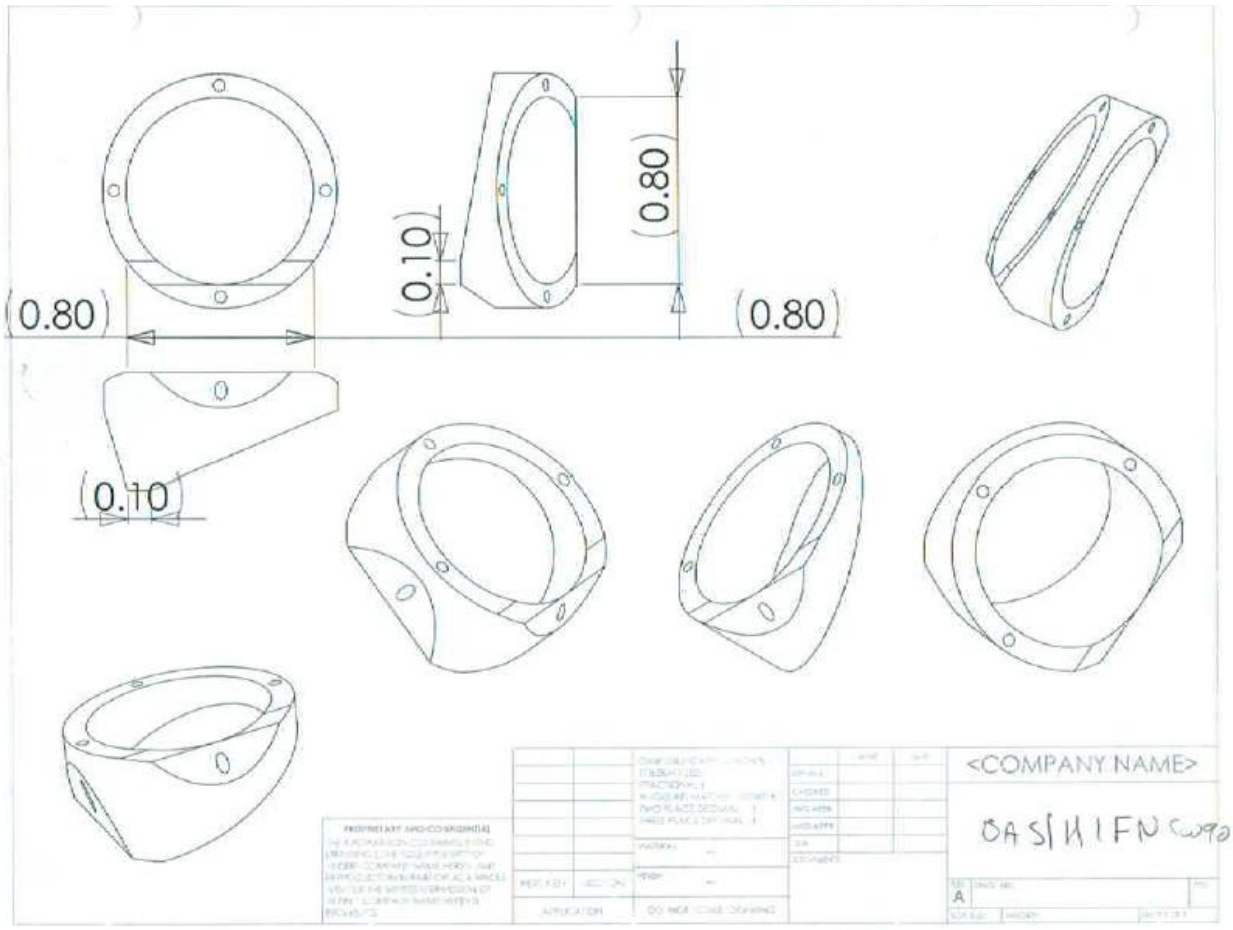


Figure 92 - Segment Design OAS1H1FN1CW90

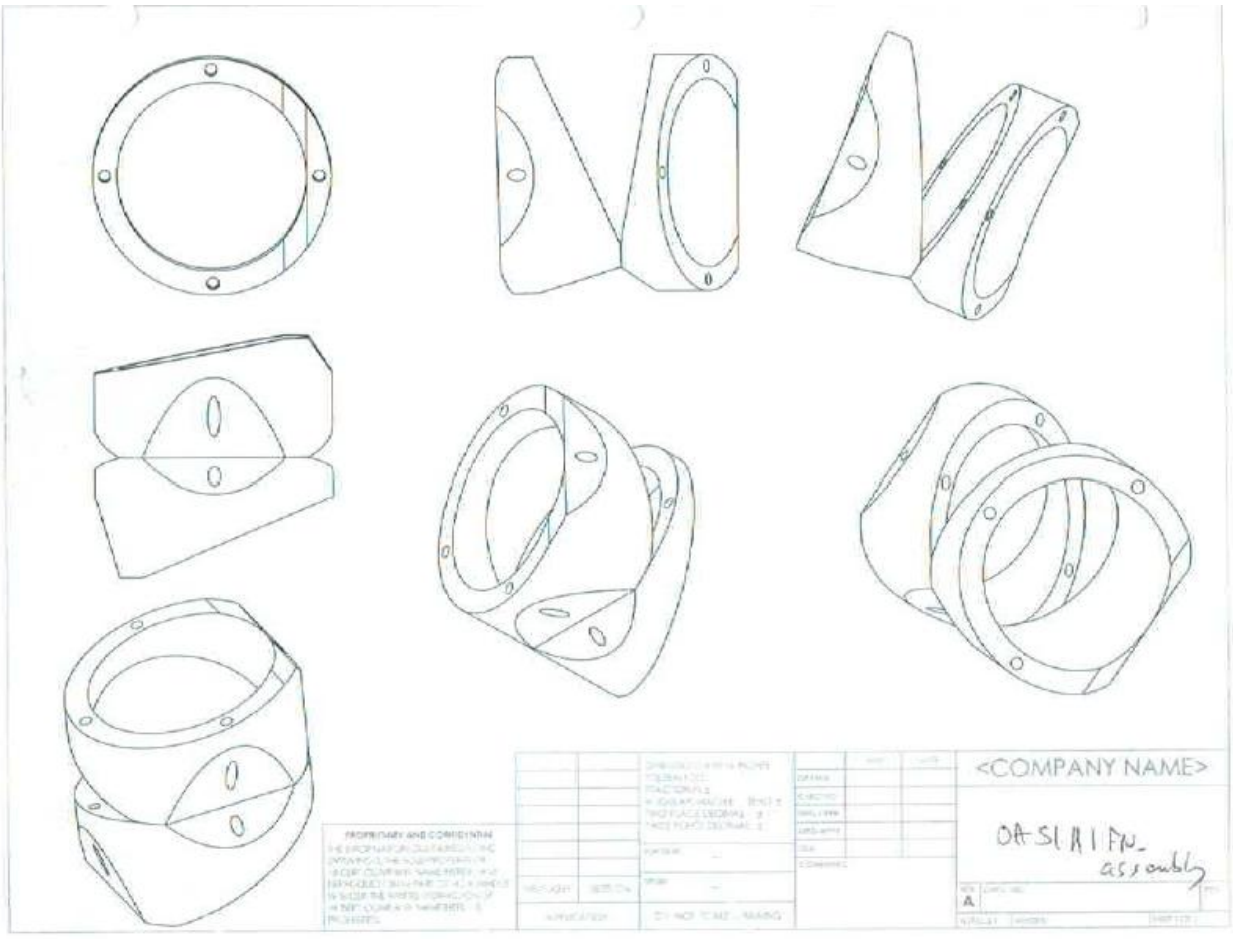


Figure 93 - Segment OAS1H1FN1CW90 Assembly

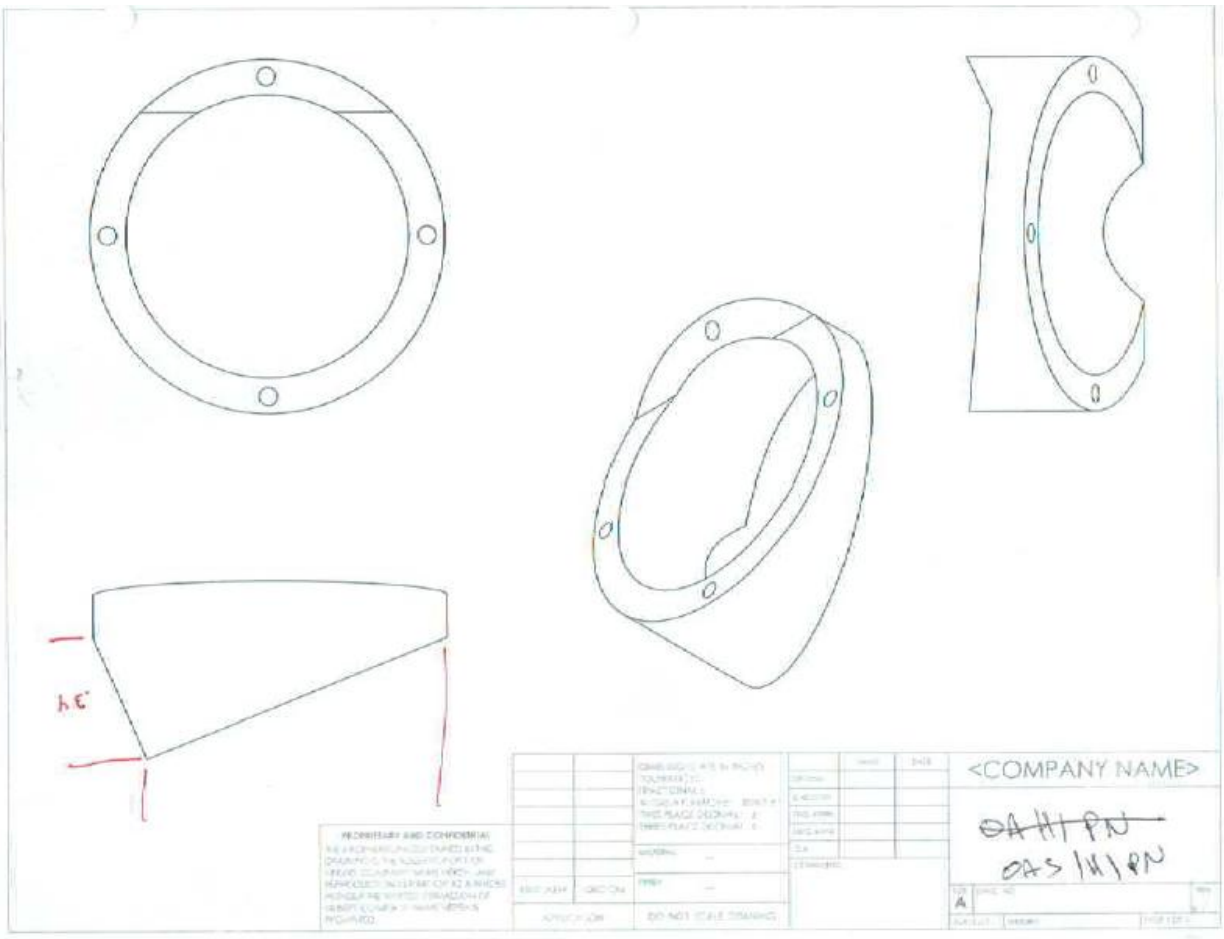


Figure 94 - Segment Design OAS1H1PN

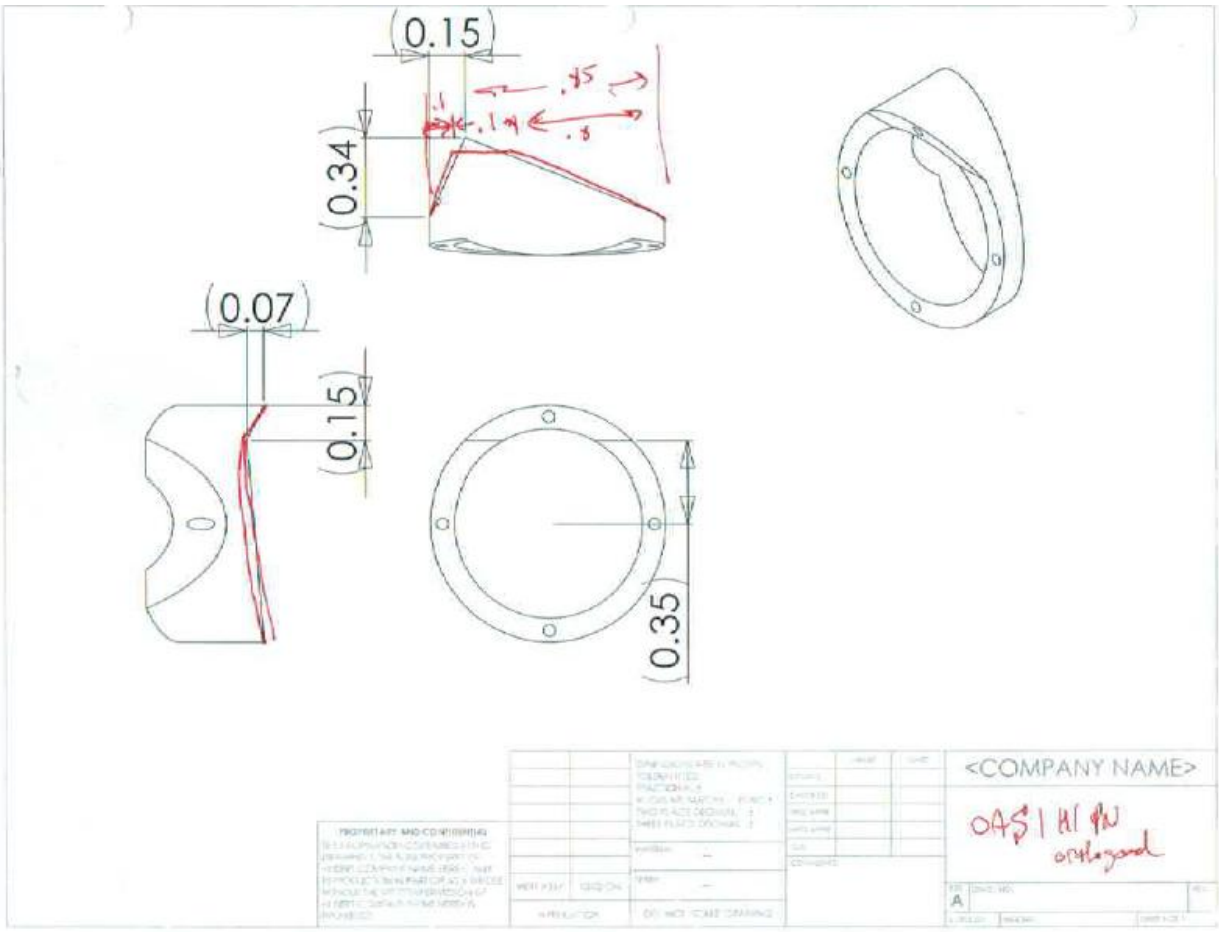


Figure 95 - Segment Design OAS1H1PN

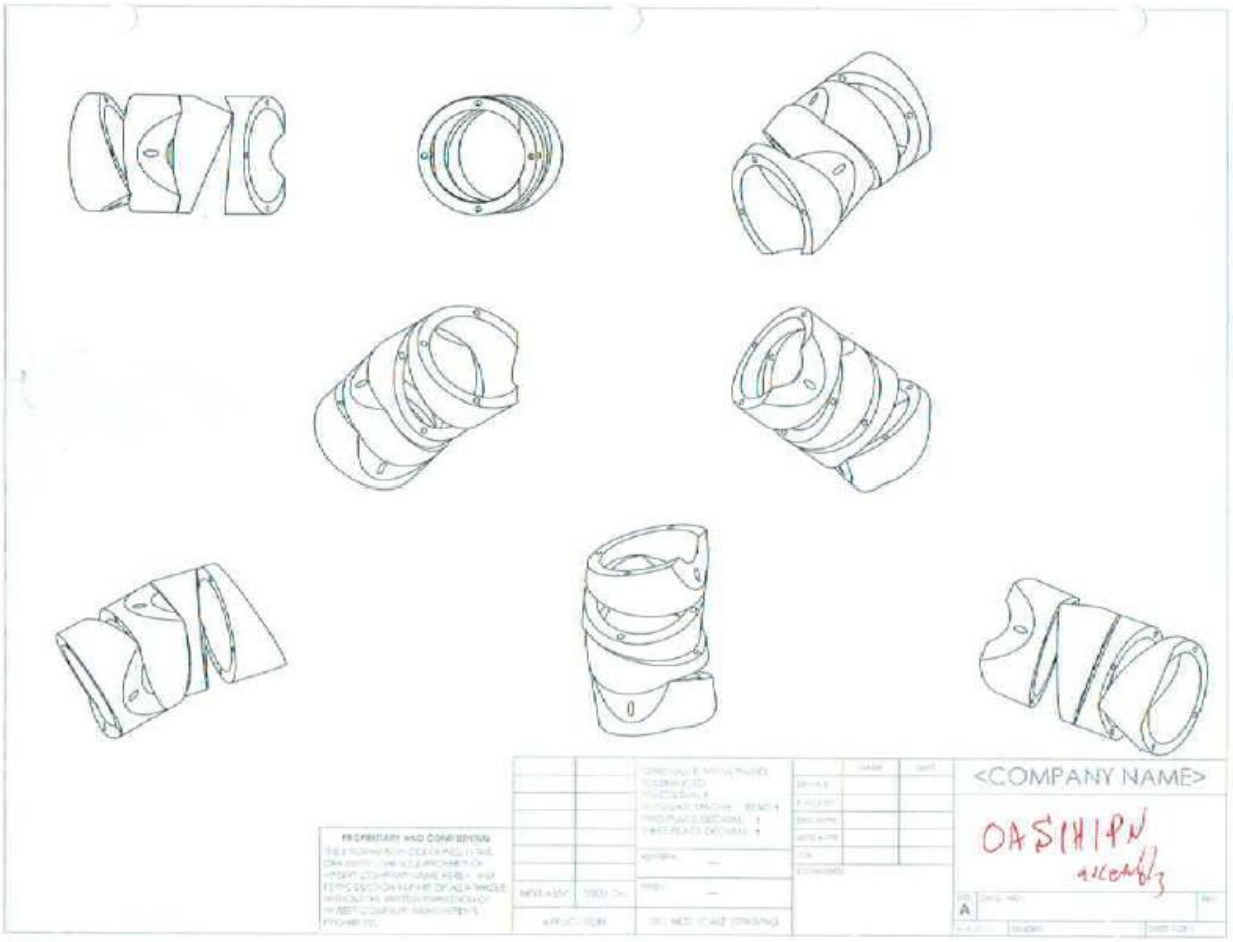


Figure 96 - Segment OAS1H1PN Assembly

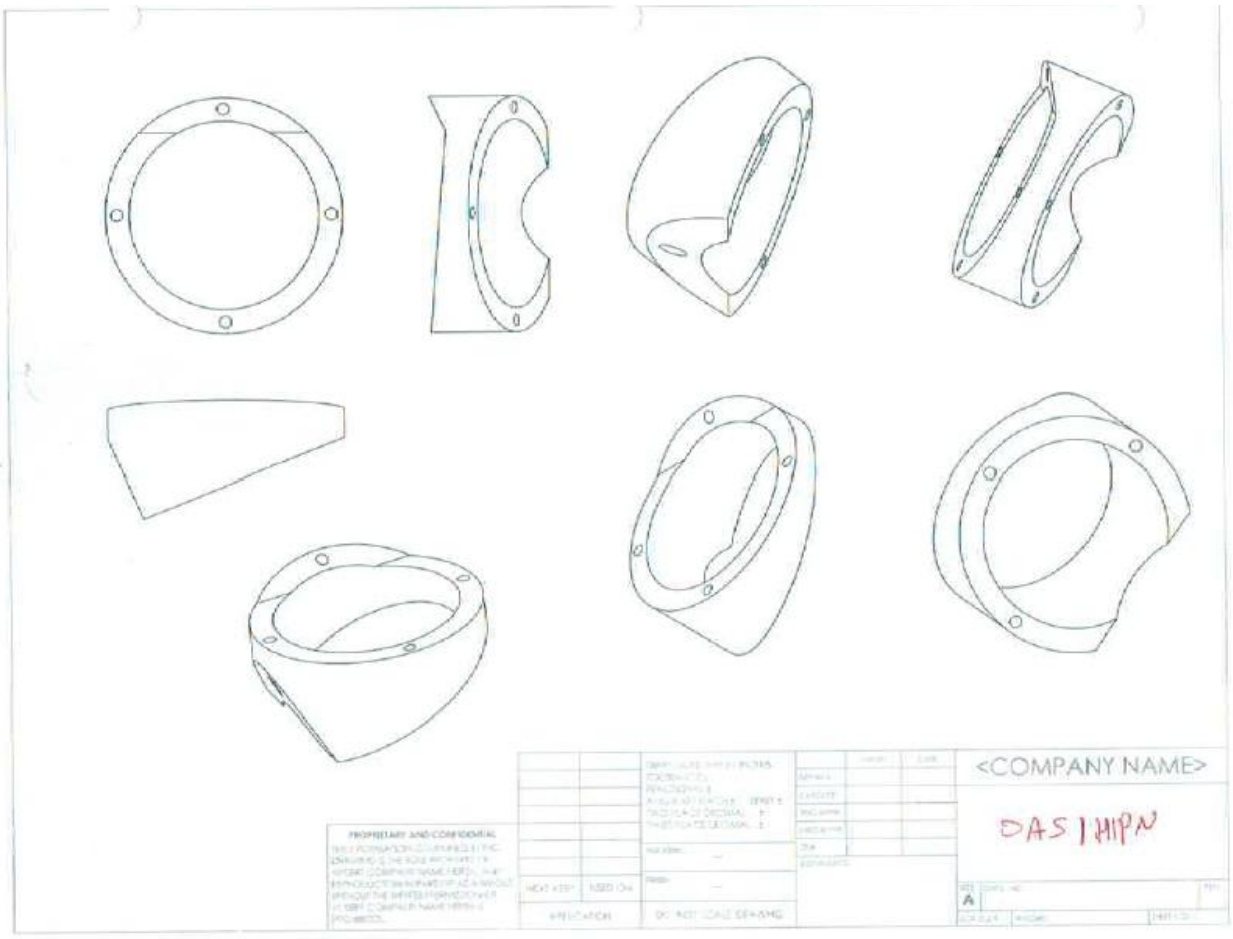


Figure 97 - Segment Design OAS1H1PN

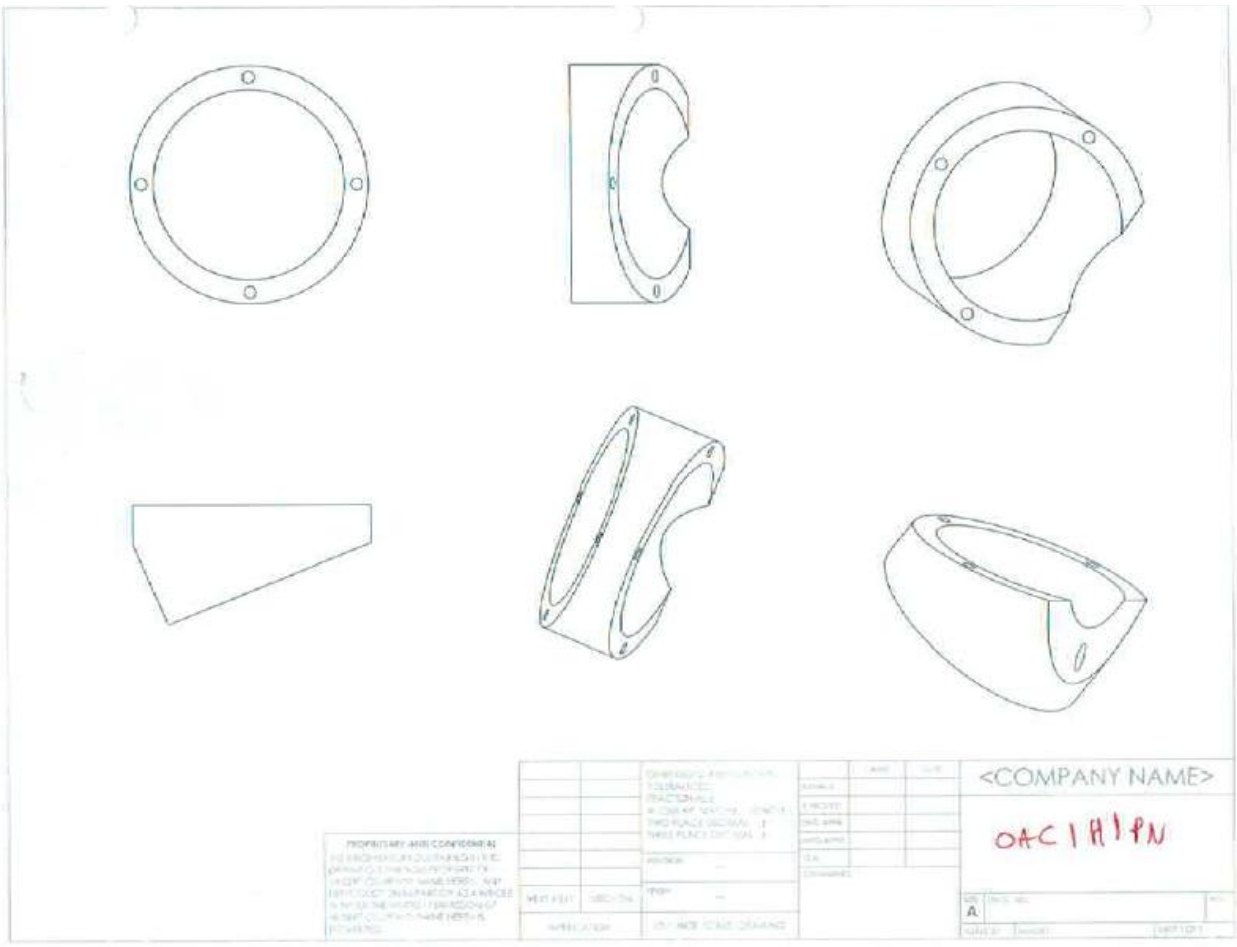


Figure 98 - Segment Design OAC1H1PN

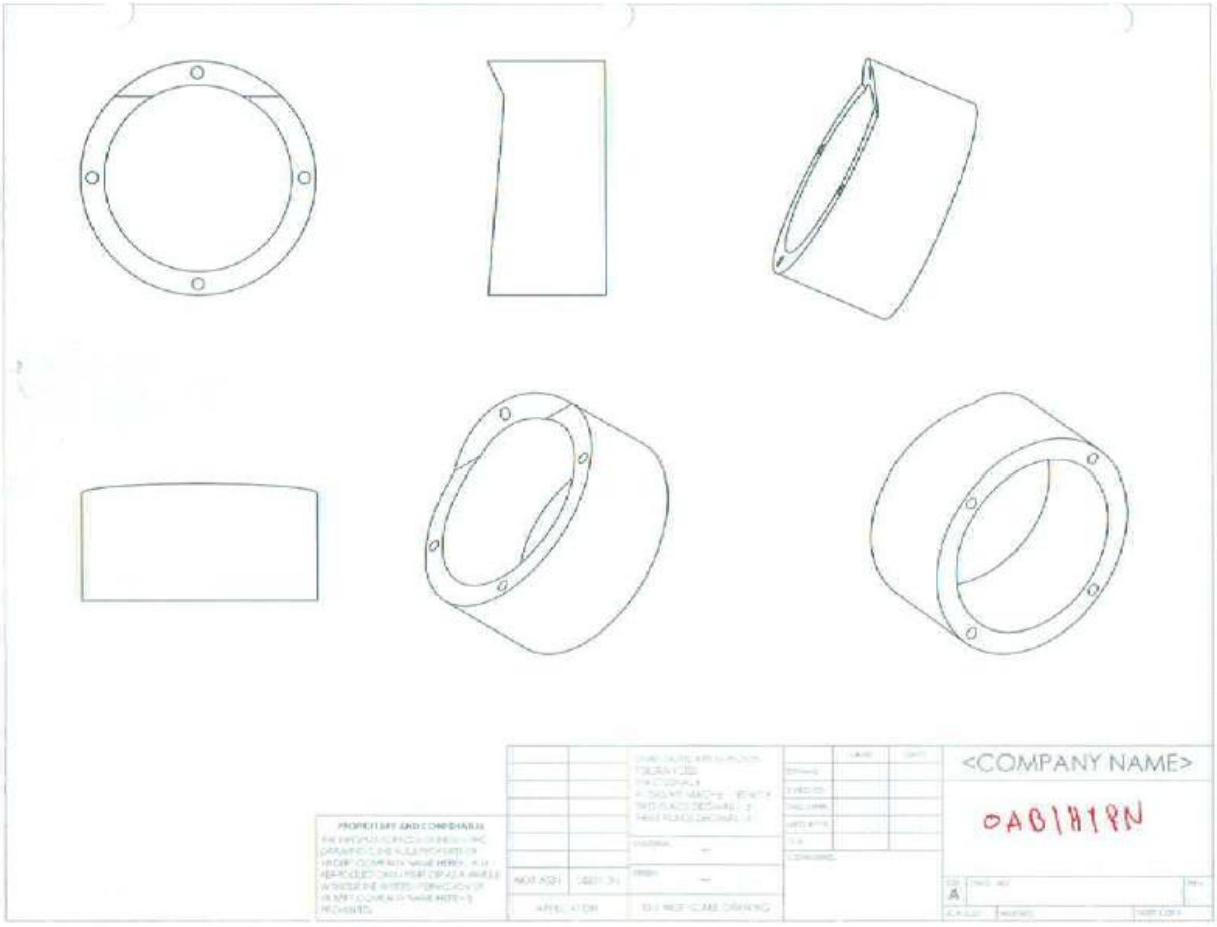


Figure 99 - Segment Design OAB1H1PN

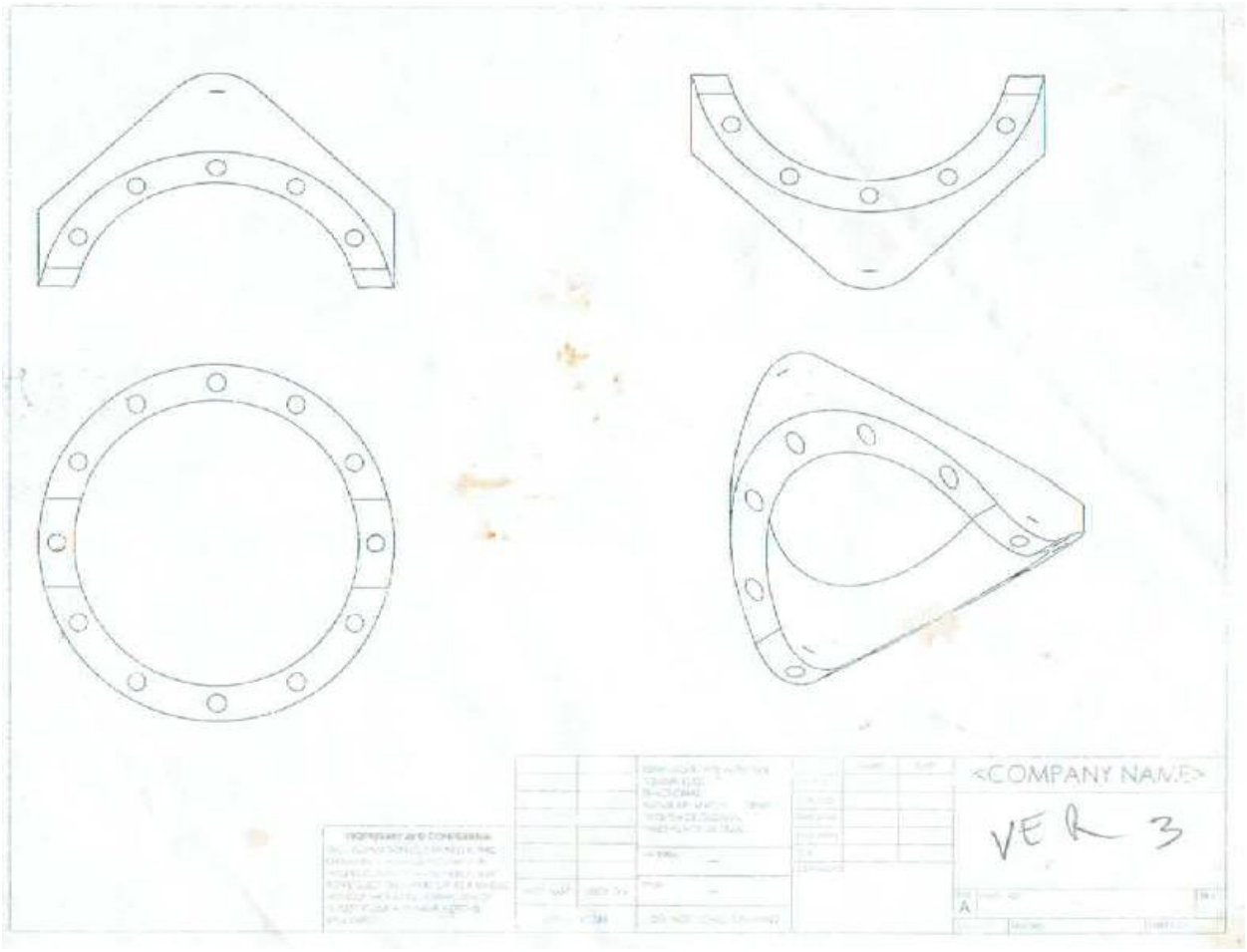


Figure 100 - Version 3 Segments

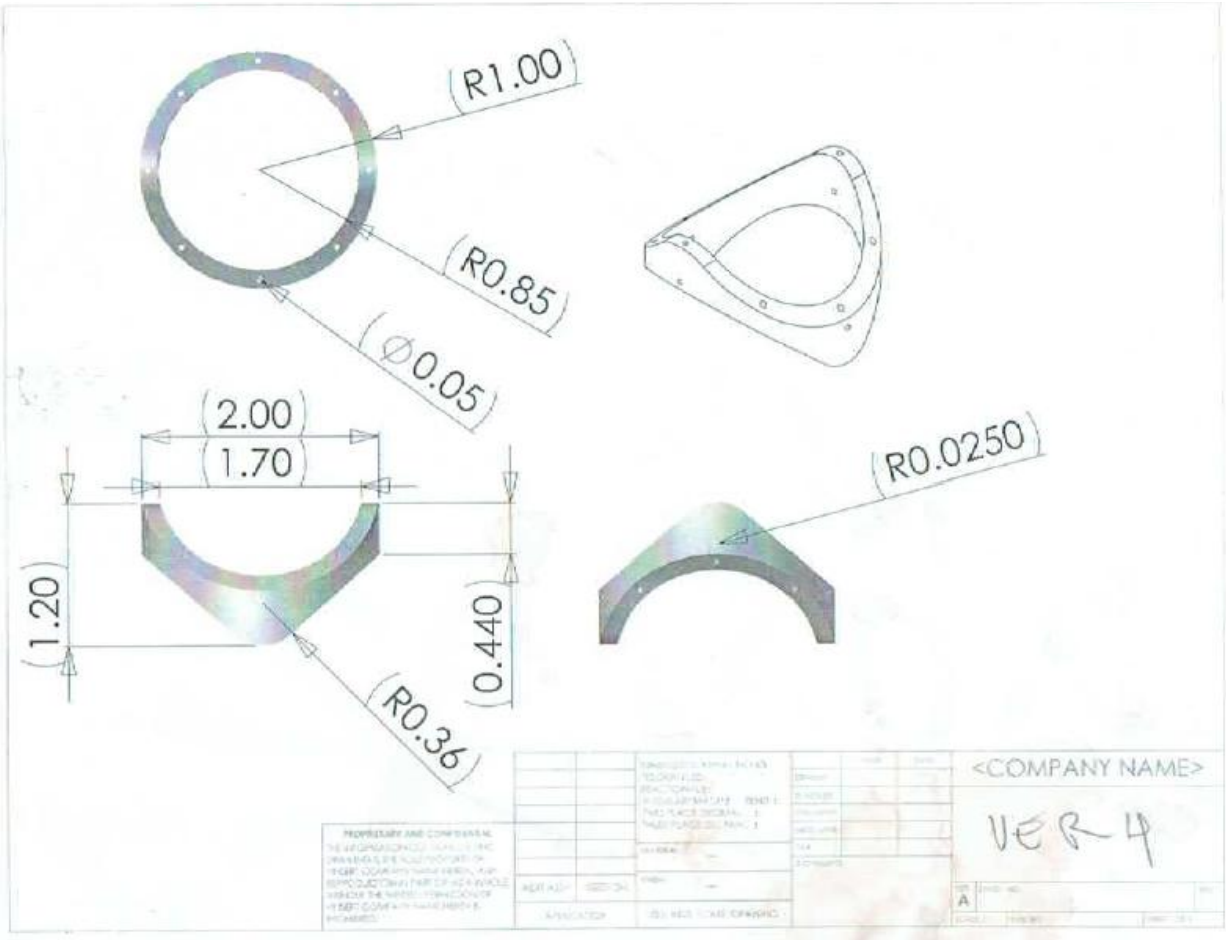


Figure 101 - Version 4 Segments

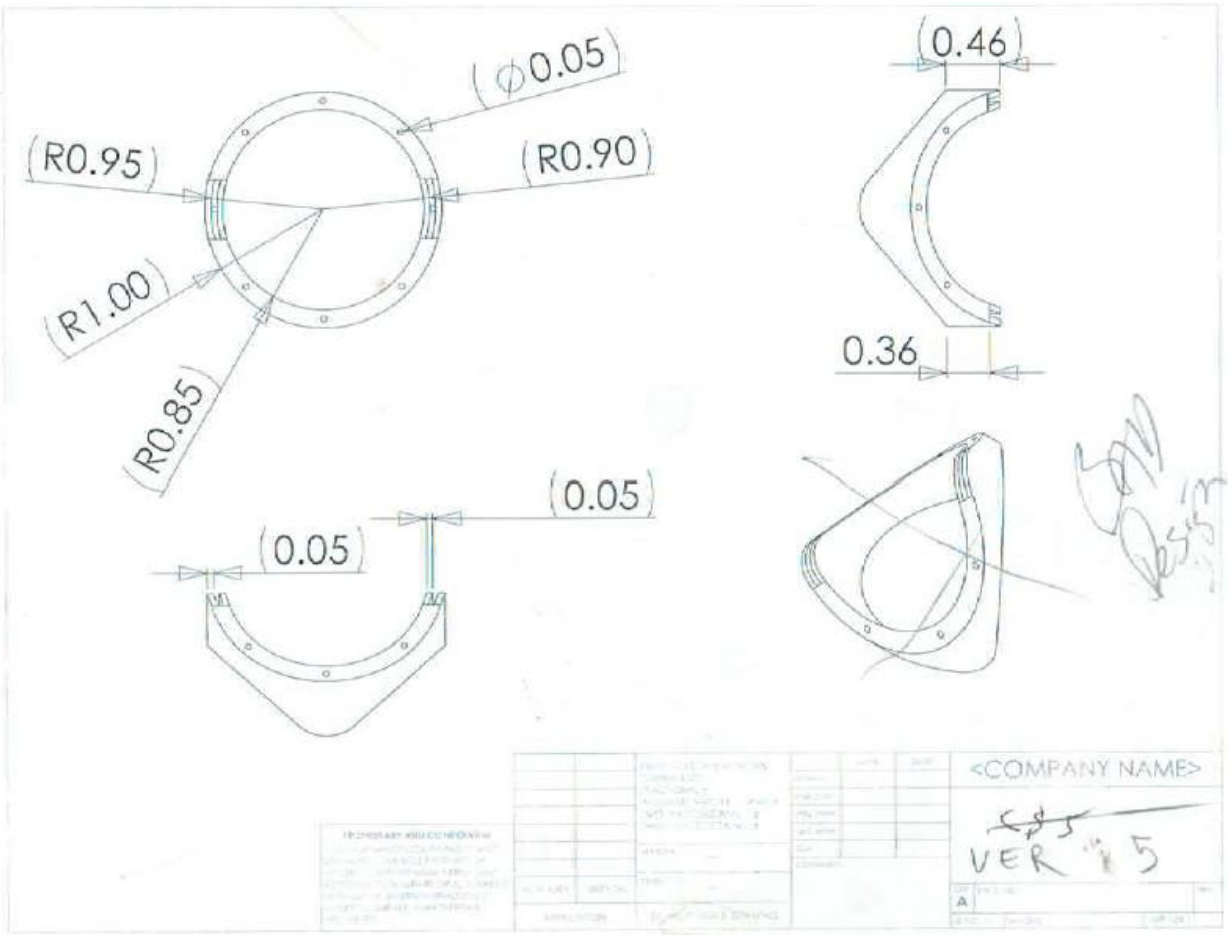


Figure 102 - Version 5 Segments

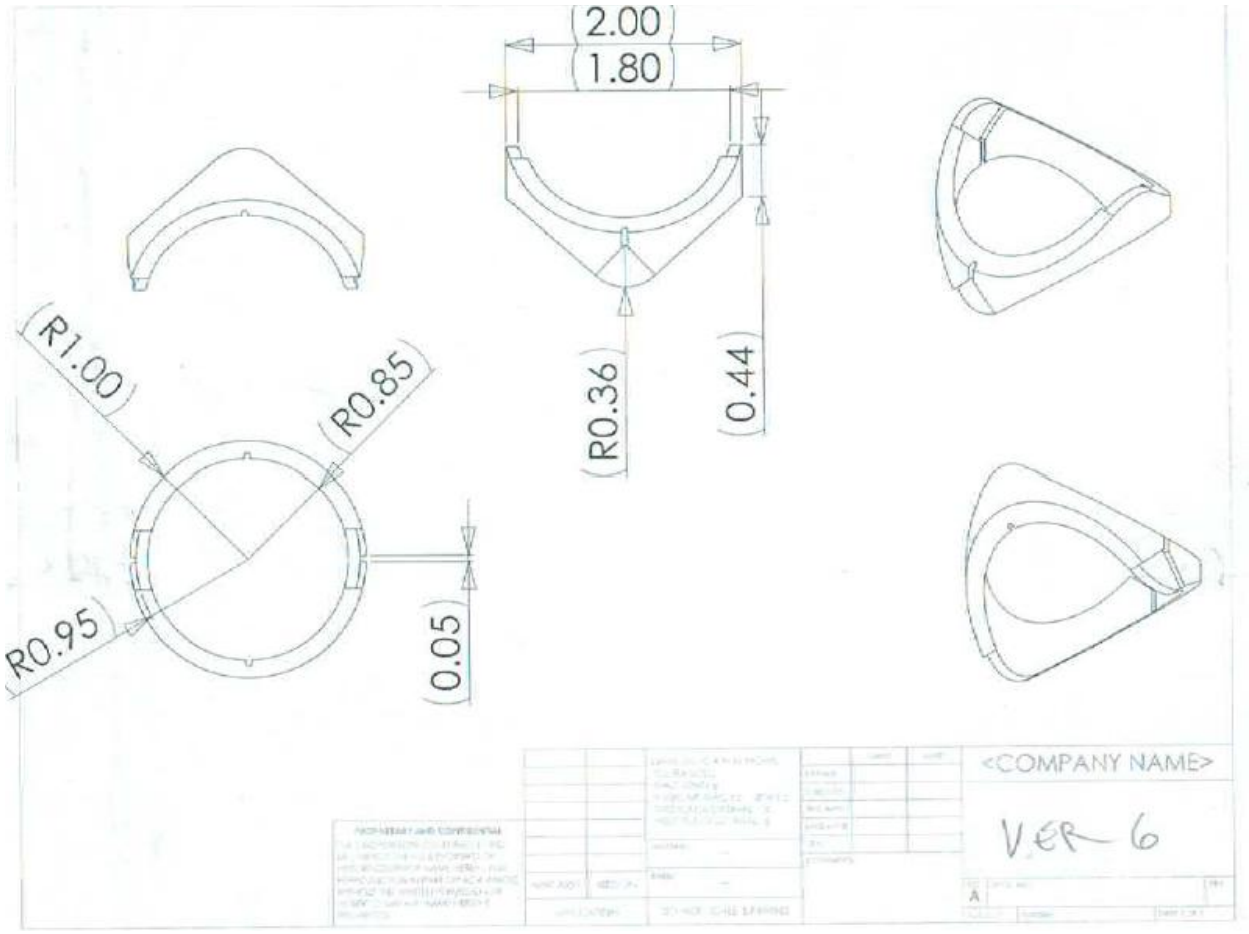


Figure 103 - Version 6 Segments

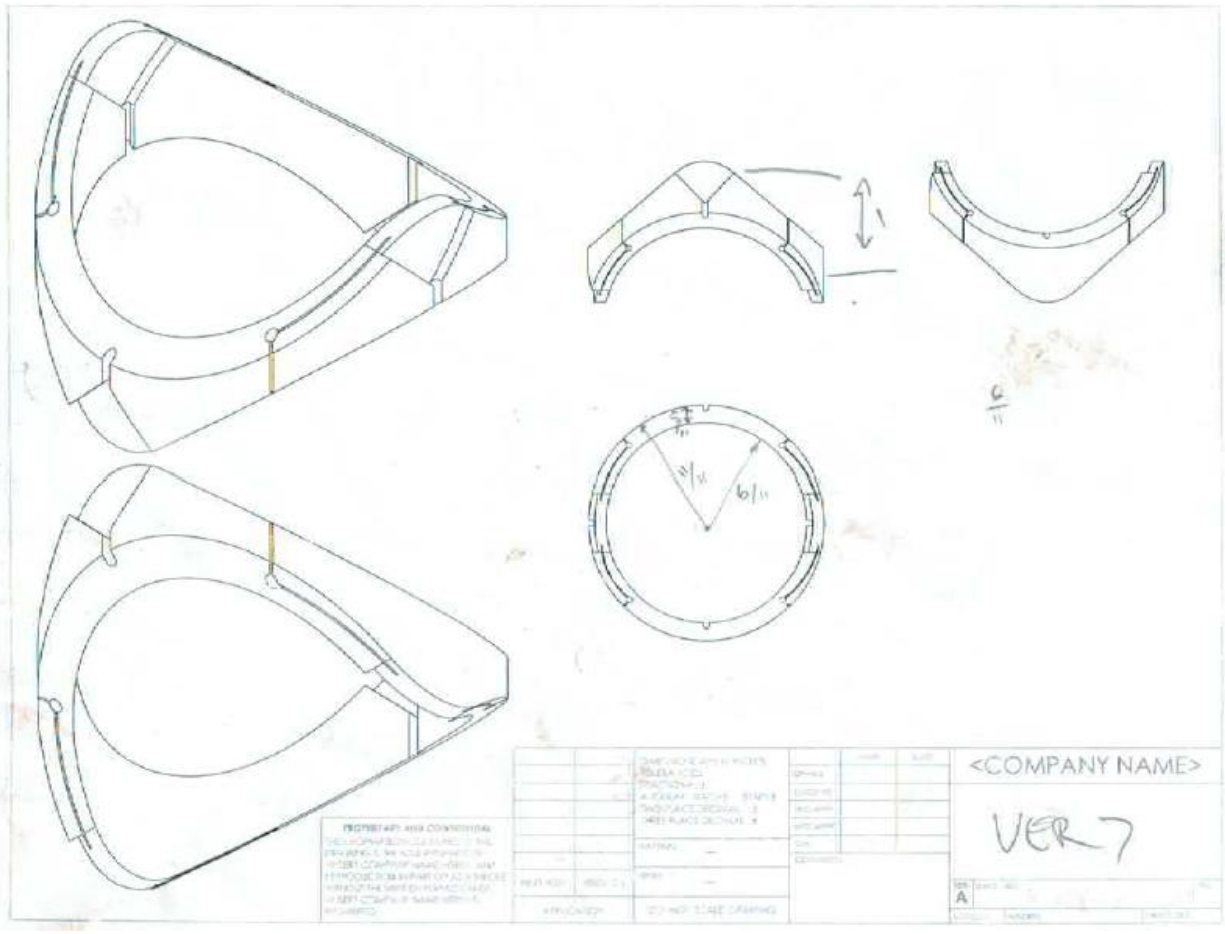


Figure 104 - Version 7 Segments

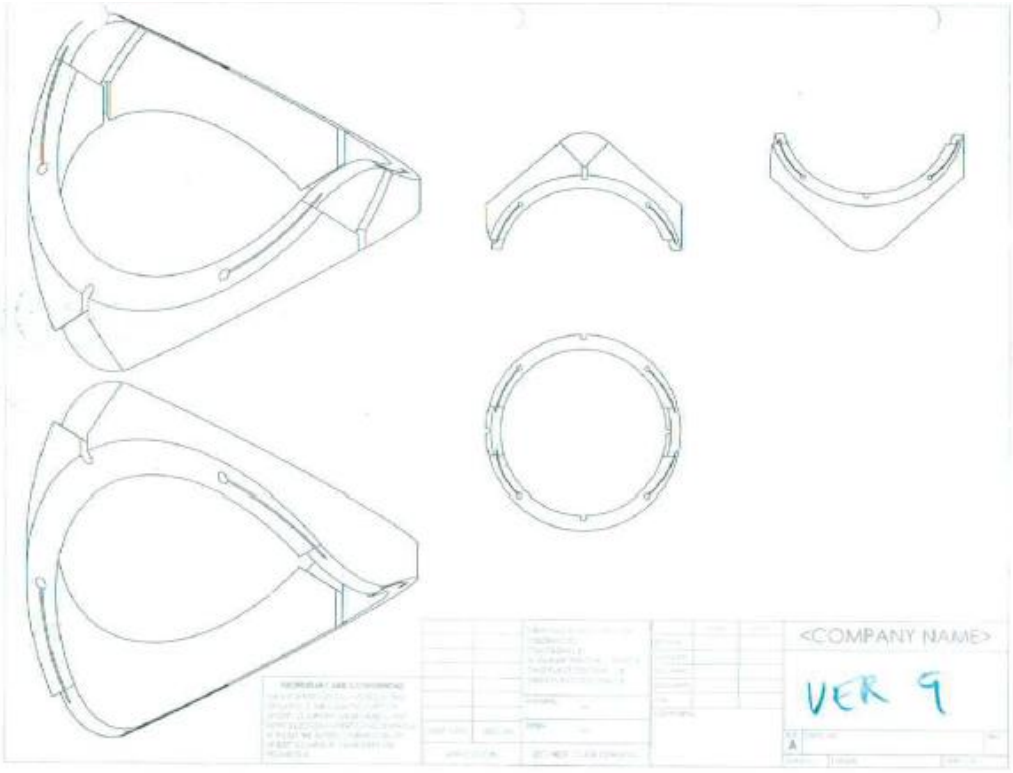


Figure 105 - Version 9 Segments

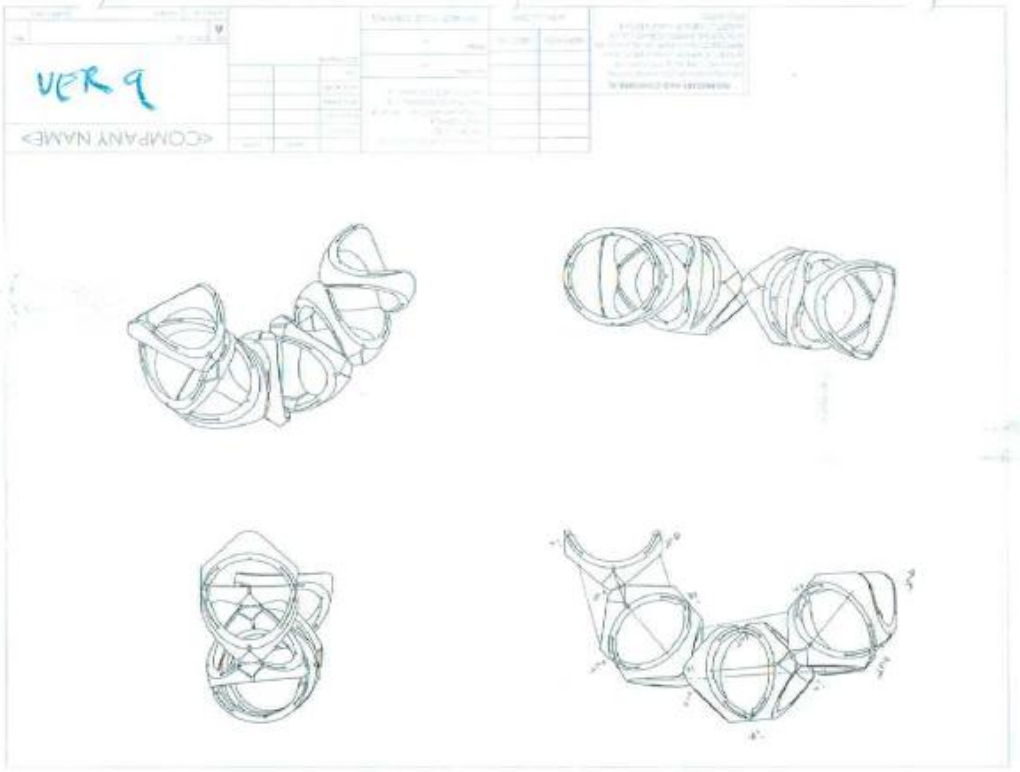


Figure 106 - Version 9 Assembly

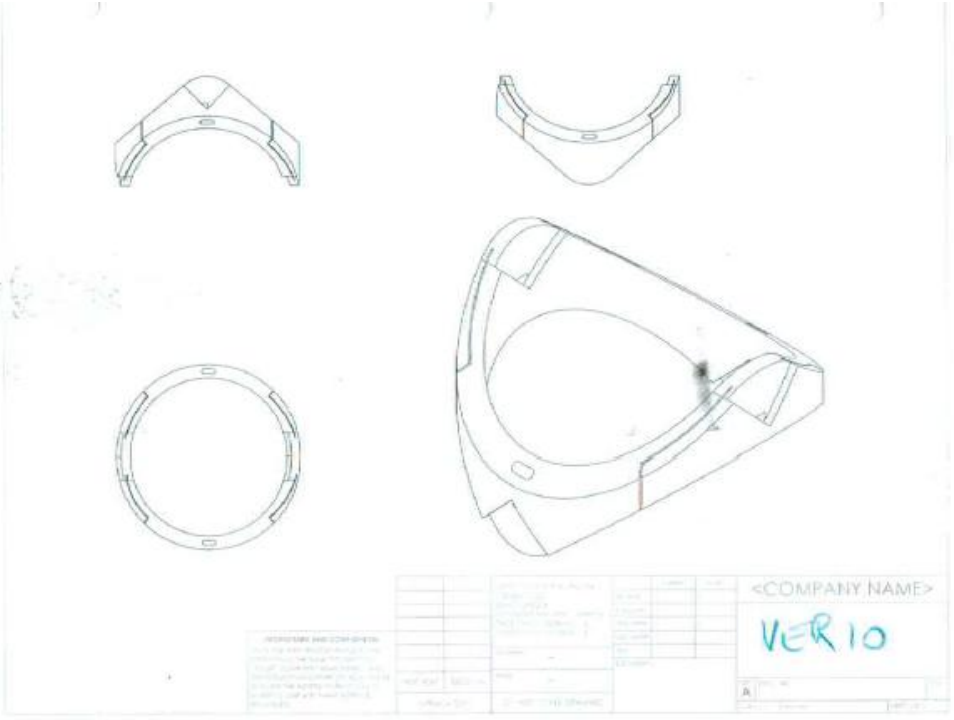


Figure 107 - Version 10 Segment

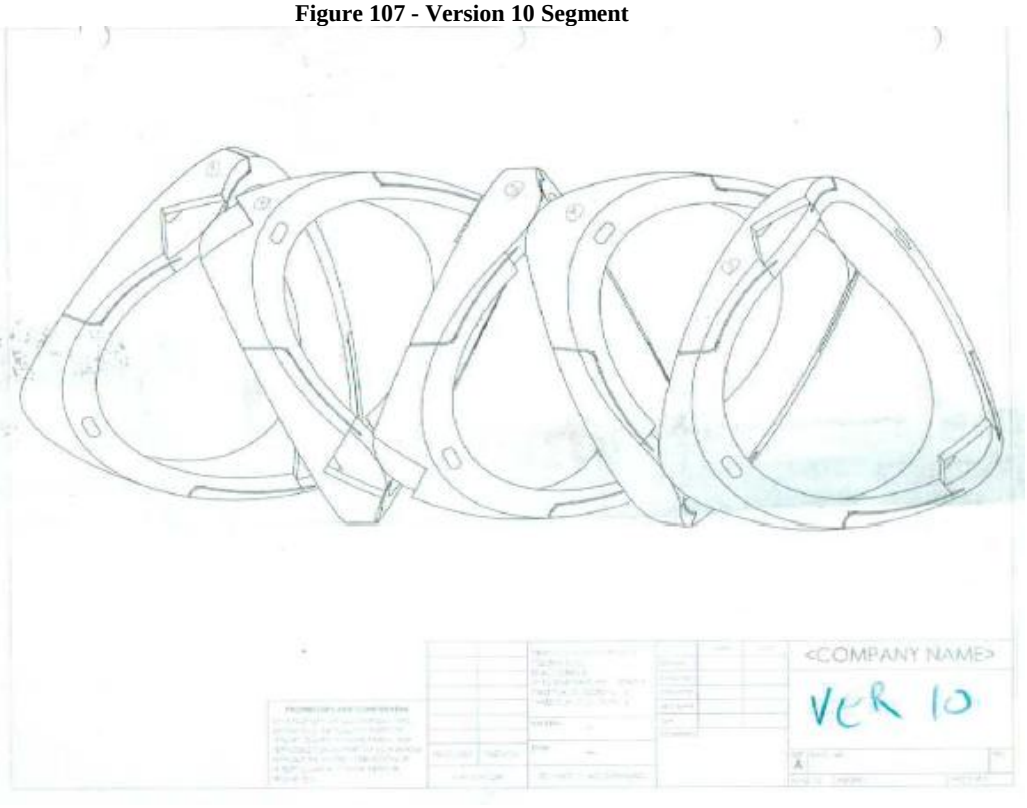


Figure 108 - Version 10 Assembly

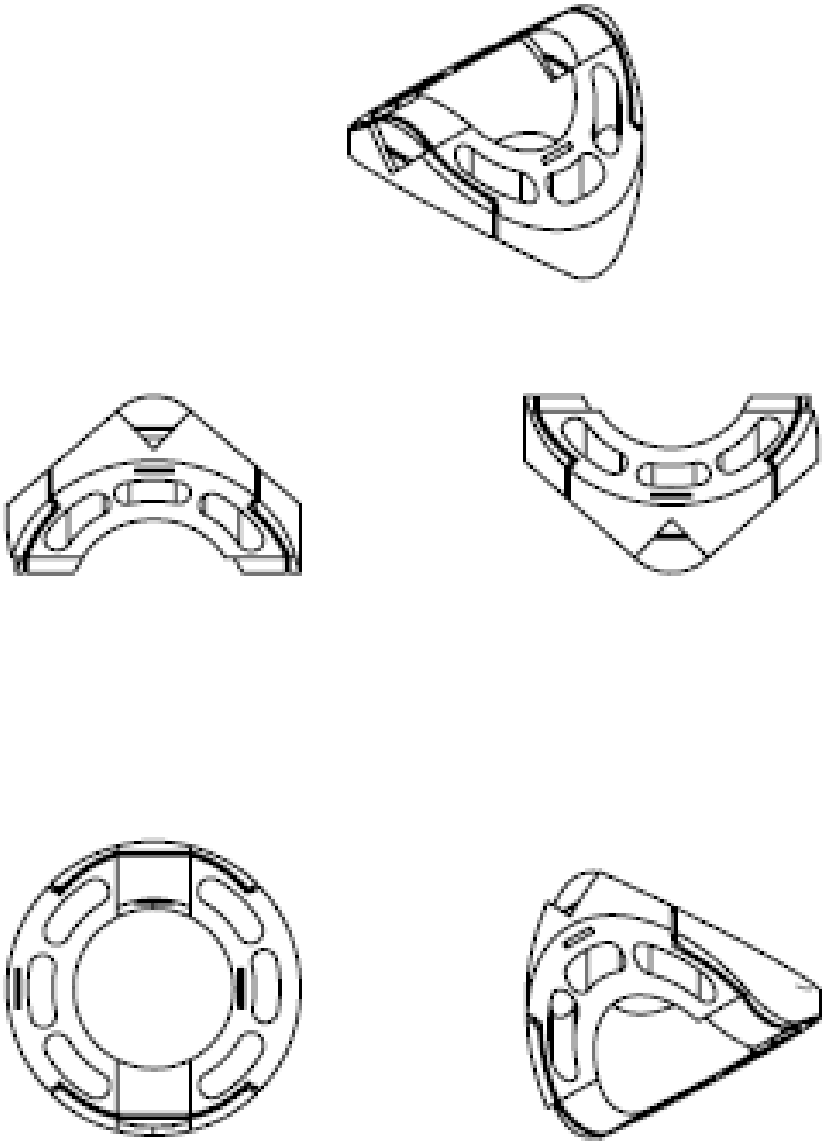


Figure 109 - Version 11

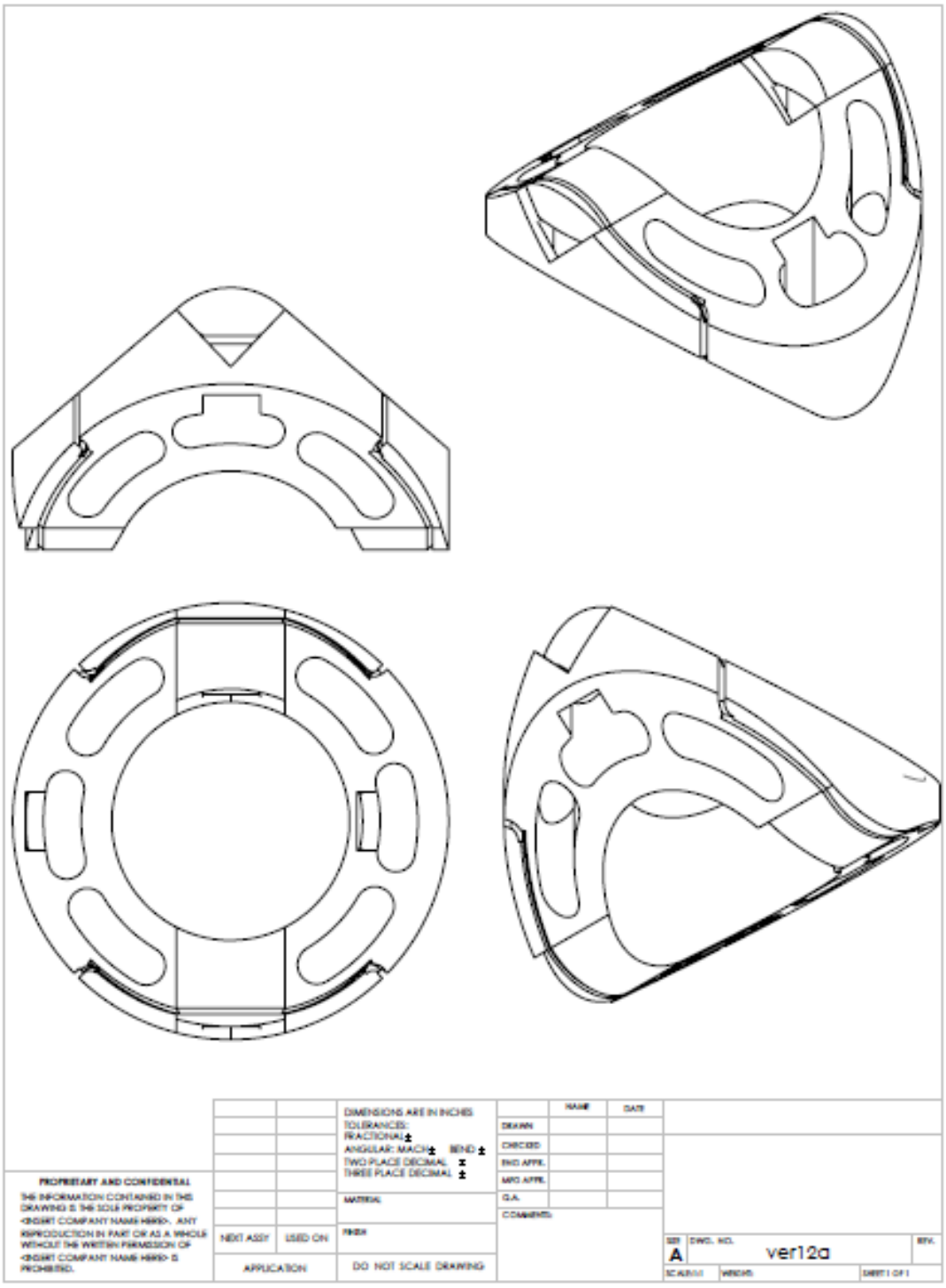


Figure 110 - Version 12a

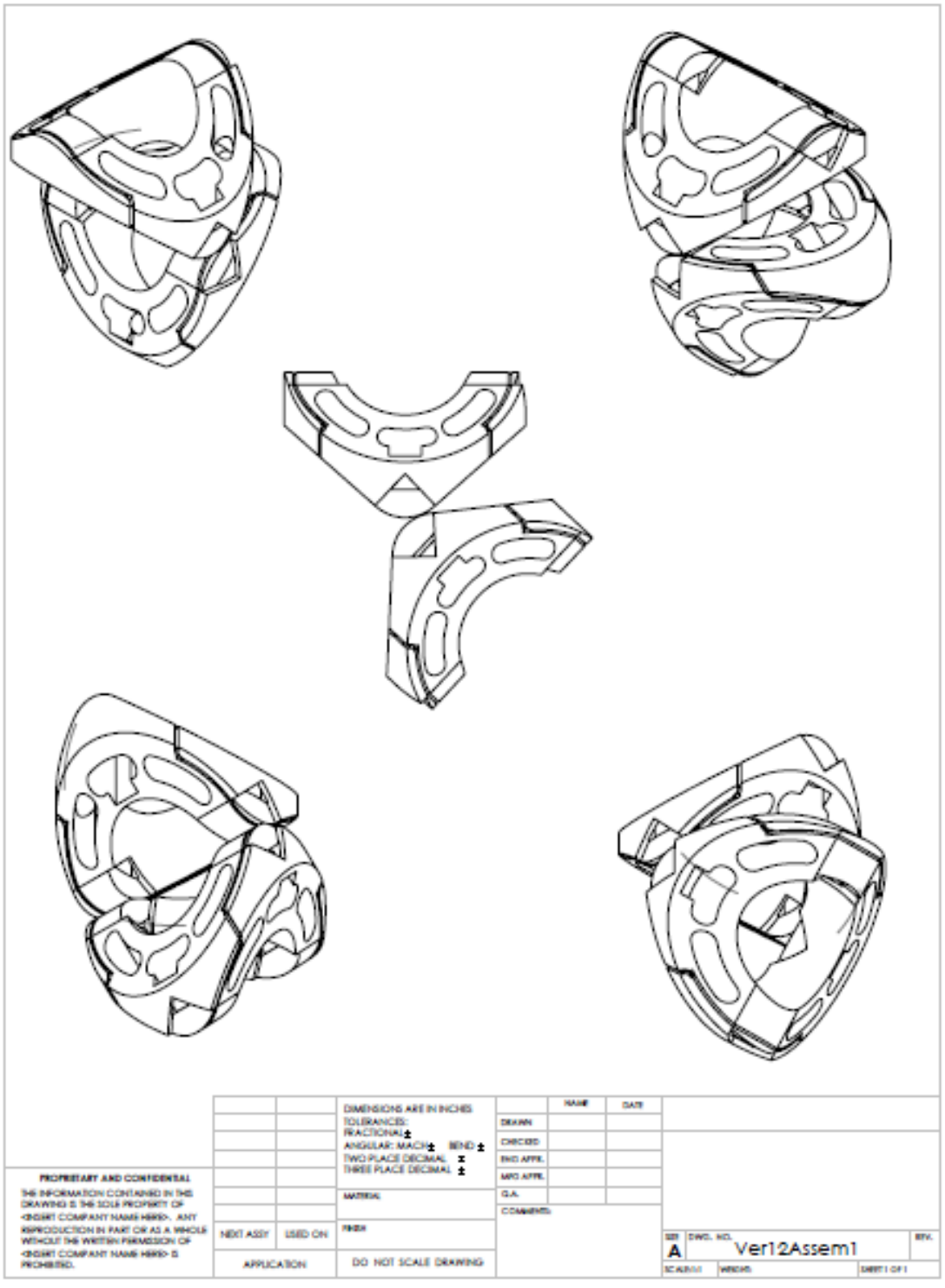


Figure 111 - Version 12a Assembly

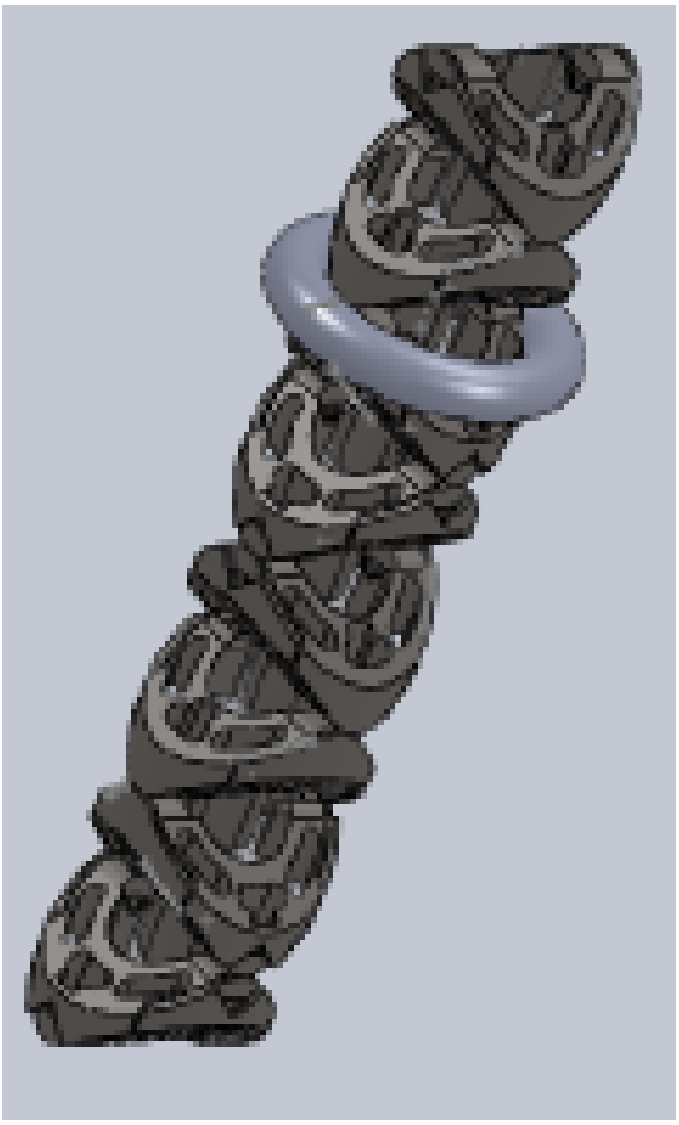


Figure 112 - Version 12a with stabilization balloon-collar

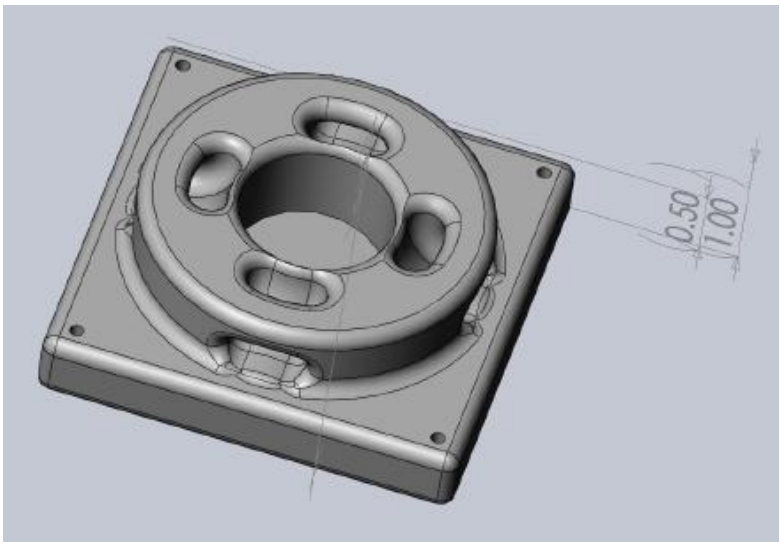


Figure 113 - Baseplate for Version 12a - nonmotorized snakebot

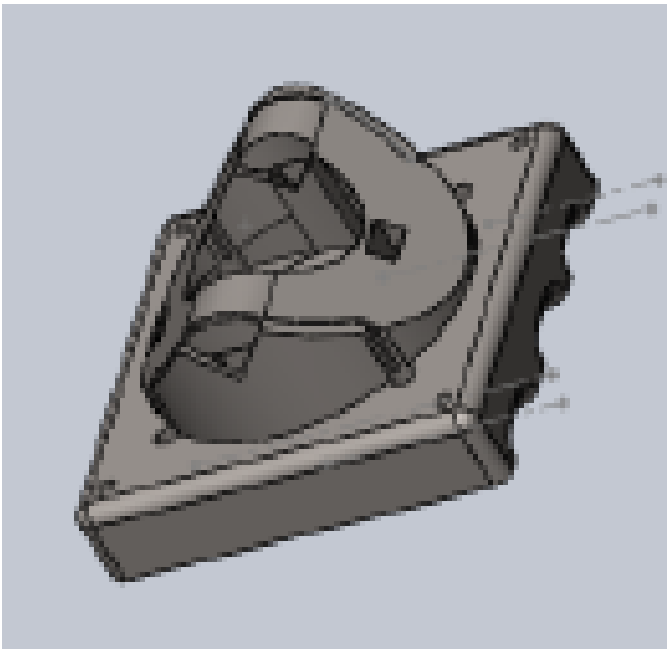


Figure 114 - Baseplate for Segment Version 12a - motorized snakebot

Figure 115 - Implementation of OSS4H1RN1CW90

Figure 116 - Implementation of CS1H2PN

Figure 117 - Implementation of CSS1PN1CW90

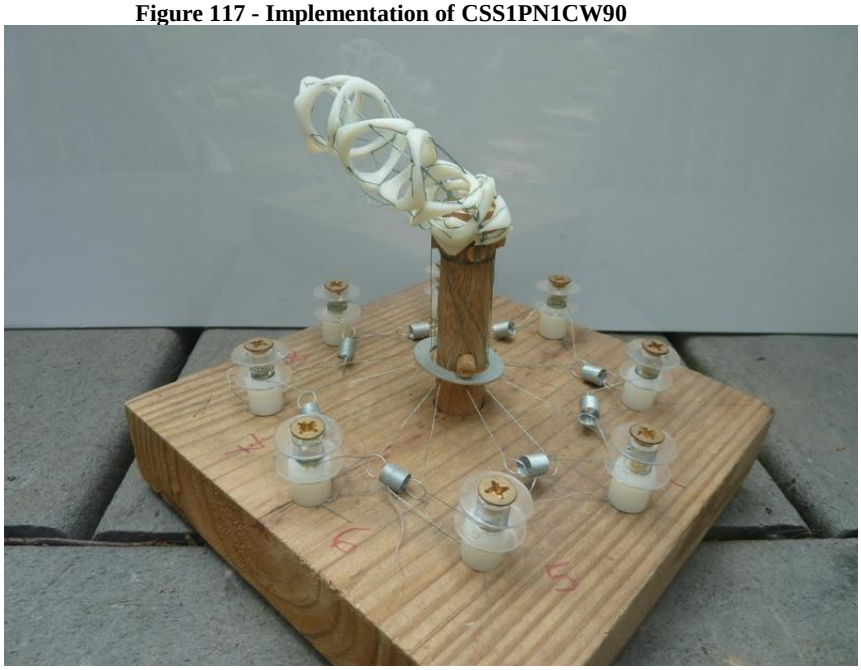


Figure 118 - Implementation of CS2H2FN

Figure 119 - Scrapbot - versions 3 through 10

Figure 120 - Segment 11 Snakebot - nonmotorized

Figure 121 - segment 12a Snakebot – motorized

Appendix 2

Software

Embedded Code for the 2560

Snakebot_2560.c:

```
/*--------------------------------------------------------------------

   snakebot_2560.c


   07 Dec 11  - copied from LEDtest2560.c
            changed message passing paradigm:
                  1 - make move commands to absolute axes' values
                                    2 - add move index to each axes' move commands
                                    3 - add move-command-echo to axes above axis being moved
                                       so upper axes can update limits
             4 - eliminate error checking and re-sending
   15 July 18 - copied from twi_test_2560.c as core of PC<->2560<->UNO
            This file runs on the 2560 dev board (Arduino Mega2560).
            It recieves commands from the PC via the USB port, parses
            these cmds, sends messages to UNO boards via the TWI
            comms, recieves and parses messages from the UNO boards
            via TWI, and returns responses to the PC via USB.

 *--------------------------------------------------------------------*/
#include <WProgram.h>
#include <stdlib.h>
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <HardwareSerial.h>
#include <print.h>
#include <wire.h>
#include <ctype.h>
#include <avr/io.h>
#include <avr/pgmspace.h>          // for RW var from FLASH mem
#include <avr/eeprom.h>           // for RW var to/from EEPROM
#include <lightweightRingBuff.h>    // TWI_ringBuf - ring buff routines

#include <snakebot_common.h>       // part of this executable's src
```

```
#include <snakebot_2560.h>
#include <snakebot_EEPROM2560.h>
#include <menu2560.h>              // help menu
#include <LED_2560.h>
#include <version.h>              // version numbers

                              /* un-comment enable testing LEDs */
//#define __DEBUG_TEST_LED_CONTROLS__

#define byte uint8_t

// ------------------------------------Global Variables ---------------//

byte TWI_ID_g = I2C_MASTER_00;              // TWI identity
RingBuff_t TWI_ringBuf;                    // Two Wire Rcv Buffer

per_axis_2560_t per_axis_2560_g[ NUMAXES ];

// ------------------------EEPROM Variables ----------------------------//

LL_axesConfig_EE_t LL_axesConfig_EE[ NUMCONFIGLIMIT ] EEMEM;

/*---------------------------------------------------------------------
 *-------------------------SETUP---------------------------------------
 *---------------------------------------------------------------------*/

/*---------------------------------------------------------------------
 *  setupAxesStructs
 *
 *    it is assumed that the snakebot is physically centered at boot
 *
 */
void setupAxesStructs( void ){

  for( byte i=0; i<NUMAXES; i++ ){
    per_axis_2560_g[i].axisNum        = i;
    per_axis_2560_g[i].TWI_ID         = TWI_ID_From_AxisNum( i );
    per_axis_2560_g[i].axisPos        = 0;
    per_axis_2560_g[i].forceThreshold = FORCE_LIMIT;
  }
}
/*---------------------------------------------------------------------
 *  EnablePowerPigtailControl
 *
 *    this function enables Port B bit 6 to send +5V activating a relay
 *    that supplies 115VAC to the power supply that drives the servos.
 *    Actually writing PB6 HI is done in the parsing
 *    routines below.
 */
int EnablePowerPigtailControl( void ){
  DDRB  |= (1<<DDB6);                 // make pin 9 an output
```

```
//PORTB |= (1<<PORTB6);              // set PB6 HI = +5VDC on pin 9,
                                    // activating relay.  Done in fcn
  return TRUE;                      // below, not here
}
/*-------------------------------------------------------------------------
 *  setup_2560
 */
void setup_2560( void ){

  Serial.begin(9600);
  Serial.print_P(PSTR("\r\n  LEDtest2560.c setup_2560"));

  setupTWI(  );
  setupTimer1( );
  setupAD( );
  setupOverforceLEDs( );
  setupUpperLimitLEDs( );
  setupLowerLimitLEDs( );
  setupAxesStructs( );
  EnablePowerPigtailControl();
  setupEEstruct();
  setupVernum( VERNUM_2560 );

  sei();

  Serial.print_P(PSTR("\r\n  setup complete  \r\n"));
}
/*-------------------------------------------------------------------
 * -----------INTERRUPT SERVICE ROUTINES--------------------------------
 *-------------------------------------------------------------------*/

/*-------------------------------------------------------------------------
 *  ADC_vect
 *
 *    This ISR implements a round-robin alternation of AD inputs to read,
 *    and stores the AD values to global variables which are processed
 *    externally to the ISR.  The CurrentADInput set in the  round-robin
 *    for the 'next' AD input
 *
 *    The registers that control the AD (and implement the round-robin)
 *    are written at the end of the ISR.  It is at the end (instead of
 *    beginning) to allow the assignment circuitry time to settle between
 *     AD readings
 *
 *    on the 2560, there are 16 AD inputs (Ports F and K...make your own
 *    acronym).  Port F will be used to read the 8 axes' force sensors (F
 *    for force...duh).
 */
ISR( ADC_vect, ISR_BLOCK ){

  static byte currentADInput = 0;                   // input-index
```

```
  per_axis_2560_g[ currentADInput ].currentForce = ADCH;// ADMUX ctrls
  if( ++currentADInput > 7 ) currentADInput = 0;       // chan 0->7
                                        // setup next AD
  ADMUX &= 0xf0;                              // clr RH 4 bits
  ADMUX |= currentADInput;                        // set LSN
                                        // time b/f nxt
}
          // for AD settle
/*---------------------------------------------------------------------
 *---------------------FUNCTIONS---------------------------------------
 *---------------------------------------------------------------------*/


/*---------------------------------------------------------------------
 *  sendMsgToAllAxes
 *       buf[0]     = 'M' or 'J'
 *       buf[1]     = '1' == axis #1 == [0->7]
 *       buf[2,3...]=  Release/Init
 */
static void sendMsgToAllAxes( byte * buf, byte numBytes ){
  for( byte i=0; i< NUMAXES; i++ ){
    buf[1] = i + '0';                 // convert num-to-ascii
          byte TWI_dest = TWI_ID_From_AxisNum( i );
          TWI_SendMessage( buf, TWI_dest, numBytes );
  }
}
/*---------------------------------------------------------------------
 *   parseMoveCommand( )
 *     args:  buf[] an array of byte ascii args from the USB or TWI
 *       buf[0]   = 'J' or 'M'
 *       buf[1]   = '1' == axis #1
 *       buf[2]   = ','
 *       buf[3,...] = destination axis count.  decimal-ascii
 */
byte parseMoveCommand( byte * buf, byte numBytes ){

  moveParms_t moveParms;

  if( numBytes < 4 ) return FALSE;

  if( !parseMoveParms( &moveParms, buf )) return FALSE;

  switch( moveParms.axisDest ){

    case INIT_AXES:     setupAxesStructs();
    case RELEASE_AXES:  sendMsgToAllAxes( buf, numBytes );    break;
          case ZERO_AXES:
      numBytes = createMoveCmd( "M", (char*)buf, moveParms.axisNum+'0', 0 );
    default: TWI_SendMessage( buf, moveParms.TWI_dest, numBytes );
  }
  return TRUE;
```

```
}
/*-----------------------------------------------------------------------
 *  printPerAxisStruct
 */
void printPerAxisStruct( byte axisNum ){

  per_axis_2560_t * ptr = &per_axis_2560_g[ axisNum ];

  Serial.print_P(PSTR("\r\n " ));
  Serial.print_P(PSTR("\r\n TWI_ID=" ));  Serial.print((int) ptr->TWI_ID,  DEC );
  Serial.print_P(PSTR("\r\n axisNum=" )); Serial.print((int) ptr->axisNum, DEC );
  Serial.print_P(PSTR("\r\n axisPos=" )); Serial.print((int) ptr->axisPos, DEC );
  Serial.print_P(PSTR("\r\n prevPos=" )); Serial.print((int) ptr->prevPos, DEC );
}
/*-----------------------------------------------------------------------
 *  printCurrentAxesPositions
 */
inline void printCurrentAxesPositions( void ){

  Serial.print_P(PSTR("\r\n current axes' positions as known by 'master':  " ));

  for( byte i=0; i<NUMAXES; i++ ){
    per_axis_2560_t * ptr = &per_axis_2560_g[ i ];
    Serial.print_P(PSTR("\r\n axisNum=" )); Serial.print((int) ptr->axisNum, DEC );
    Serial.print_P(PSTR(   ", axisPos=" )); Serial.print((int) ptr->axisPos, DEC );
  }
}
/*-----------------------------------------------------------------------
 *  updatePerAxisRAM
 */
void inline updatePerAxisRAM( axisStatus_t * axisStatus_p ){

  byte i = axisStatus_p->axisNum;
  if( i < NUMAXES ) {
    per_axis_2560_g[ i ].prevPos  = axisStatus_p->prevPos;
    per_axis_2560_g[ i ].axisPos  = axisStatus_p->axisPos;
  //printPerAxisStruct( i );
  }
}
/*-----------------------------------------------------------------------
 *  axisStatusConvertAndStash
 */
void inline axisStatusConvertAndStash( byte * buf, axisStatus_t * ptr ){
  convertAsciiAxisStatusToDec((char*) buf, ptr );
  updatePerAxisRAM(  ptr );
//printAxisStatus_t( ptr );
}
/*-----------------------------------------------------------------------
 * parseAxesStatusReq
 *
 *   [0] = 'P'
```

```
 *   [1] = 'R' or 'A' or 'C'
 *         'R' = request - from TWI_Master to TWI_dest slave
 *         'A' = axis answer  - from TWI_dest slave to TWI_master (this)
 *         'C' = current complete...print what the 2560 has as pos data
 *   [2] = axis num ascii
 */
byte parseAxesStatusReq( byte * buf, byte numBytes ){

  if( buf[0] != 'P' ) return FALSE;
  if( numBytes < 3  ) return FALSE;
  byte axisNum, TWI_dest;
  axisStatus_t axisStatus;

  switch( buf[1] ){

    case 'R':
            axisNum = buf[2] - '0';
      TWI_dest = TWI_ID_From_AxisNum( axisNum );
      if( TWI_dest == I2C_FAILURE ) return FALSE;
      TWI_SendMessage( buf, TWI_dest, numBytes );
    break;
    case 'A':
            axisStatusConvertAndStash( &buf[2], &axisStatus );
      printAxisStatus_t( &axisStatus );
          break;
          case 'C':
            printCurrentAxesPositions( );
          break;
          default:  return FALSE;
  }
  return TRUE;
}
/*---------------------------------------------------------------------------
 */
byte parseAxesLimUpdate( byte * buf, byte numBytes ){

  if( numBytes < 3 ) return FALSE;
  if(( buf[0] != 'U' )) return FALSE;

  axisStatus_t movedAxisStatus;
  axisStatusConvertAndStash( &buf[1], &movedAxisStatus );
  return TRUE;
}
/*---------------------------------------------------------------------------
 *  parseCurrentForceReq
 *    args:  pointer to ascii-array, decimal number-of-bytes-in-array
 *          buf[1,2] will contain a ascii-hex bitmap of which axes report
 *    rtrn:  TRUE if successful, FALSE if not
 *      ex:  F03  for axis 1 and 2
 *          ascii out the USB port containing requested data
 *          F2F  for axes 6, 4, 3, 2, and 1
```

```
 */
byte parseCurrentForceReq( byte * buf, byte numBytes ){

  if( numBytes < 3 ) return FALSE;
  byte i, bitMapOfAxes = convert2bAsciiHexToHex( &buf[1] );

  for( i=0; i< NUMAXES; i++ ){
    if(( 1<< i ) & bitMapOfAxes ){
      Serial.print_P(PSTR("\r\n  Axis[ "));
      Serial.print( i, DEC );
      Serial.print_P(PSTR(" ].currentForce = "));
      Serial.print( per_axis_2560_g[i].currentForce,  DEC );
    }
  }
  return TRUE;
}
/*---------------------------------------------------------------------------
 *    isOverforce
 *      args:     none
 *      returns:  byte containing bit-map of axes overforce, 0 if none
 */
byte isOverforce( void ) {
  byte i, retval=NO_OVERFORCE;
  for( i=0; i< NUMAXES; i++ ){
    if( per_axis_2560_g[i].currentForce >
        per_axis_2560_g[i].forceThreshold ){
          retval |= (1<<i);
    }
  }
  return retval;
}
/*---------------------------------------------------------------------------
 *  checkOverforce
 */
void checkOverforce( void ){

  byte whichAxesAreOverforce = isOverforce();

  turnOnOverforceLEDs( whichAxesAreOverforce );  // will turn off if none
}
/*---------------------------------------------------------------------------
 *  parsePowerPigtailOnOff
 *
 *    this function sends +5V out PB6, activating a relay that supplies
 *    115VAC to the power supply that drives the servos
 *
 *    if buf[1] is 0, turn off PB6.
 */
int parsePowerPigtailOnOff( byte * buf, byte numBytes ){

  if( numBytes < 2 )  return FALSE;
```

```
  if( buf[0] != 'G' ) return FALSE;

  if(      buf[1] == '1' ) PORTB |=  (1<<PORTB6);
  else if( buf[1] == '0' ) PORTB &= ~(1<<PORTB6);
  else return FALSE;

  return TRUE;
}
/*---------------------------------------------------------------------
 *  parsePingTWI
 *
 *    this function acts as a two-way relay of ping:  first,
 *    [2] == 'P'  relay the 'ping' to TWI_id that controls
 *               the axis number of buf[1]
 *          'R'  relay the response to the PC via USB
 *
 *    eg T3P  ping controller of axis #3
 *       T7R  response from controller of axis #7
 */
byte parsePingTWI( byte * buf, byte numBytes ){

  if( numBytes < 3  ) return FALSE;
  if( buf[0] != 'T' ) return FALSE;

  switch( buf[2] ){
    case 'R':
      echoToUSB( "rcvd ping rsp", buf, numBytes, __LINE__ );
      break;
    case 'P':                          // ping UNO
    { byte axisNum = buf[1] - '0';       // convert ascii->decimal
      byte TWI_dest = TWI_ID_From_AxisNum( axisNum );
      if( TWI_dest == I2C_FAILURE ) return FALSE;
      TWI_SendMessage( buf, TWI_dest, numBytes );}
    break;
    default:   return FALSE;
  }

  return TRUE;
}
/*---------------------------------------------------------------------
 *  parseCommand
 *
 *     if the first letter of the array 'buf' contains a valid command-type
 *     and the command parses and executes in subsequent subroutines
 *     correctly, this function returns TRUE.  If not, this function
 *     echoes the argument-string 'buf' to the USB port and returns FALSE
 */
byte parseCommand( byte * buf, byte numBytes ){

    if( numBytes < 1 ) return FALSE;
```

```
//  echoToUSB( "parse cmd: ", buf, numBytes, __LINE__ );

   switch( buf[0] ){

      case 'M':
      case 'J':
        if( parseMoveCommand(       buf, numBytes )) return TRUE;
        break;
      case 'U':
        if( parseAxesLimUpdate(     buf, numBytes )) return TRUE;
        break;
      case 'L':
        if( parseLEDcommand(        buf, numBytes )) return TRUE;
        break;
      case 'T':
        if( parsePingTWI(           buf, numBytes )) return TRUE;
        break;
      case 'P':
        if( parseAxesStatusReq(     buf, numBytes )) return TRUE;
        break;
      case 'F':
        if( parseCurrentForceReq(   buf, numBytes )) return TRUE;
        break;
      case 'G':
        if( parsePowerPigtailOnOff( buf, numBytes )) return TRUE;
        break;
      case 'E':
        if( parseEEPROMdataReq(     buf, numBytes )) return TRUE;
        break;
      case 'H':
      default:
        if( printHelpMenu(          buf, numBytes )) return TRUE;
        break;
          }
          return FALSE;
}
/*---------------------------------------------------------------------------
 *   main
 *
 *    this is a loop where command-messages are parsed and executed
 *    TWI messages are parsed first so pending move responses are handled
 *    and axes' values updated.  TWI is first so action pending updated
 *    axes coordinates aren't kept waiting.
 *    SelfSend messages are parsed second.  These include move commands
 *    from the PC that are held pending previous moves, configuration move
 *    commands pending completion of previous moves, etc.
 */

int main( void )
{
  byte numBytes, temp[ BUFFSIZE ];
```

```
  setup_2560( );

  while(1){

   if( RingBuffer_GetCount( &TWI_ringBuf )){   // check TWI message

     numBytes = readTWImsg( temp );
     parseCommand( temp, numBytes );
   }
   else if( Serial.available()){              // check for USB comms

            numBytes = readUSBmsg( temp );
     parseCommand( temp, numBytes );
   }
   else{

     heartbeatDots(  );                      // USB-out heartbeat dots
     _delay_ms( 250 );                        // NEED this delay to
          }
                                    // provide time for comms
   #ifdef __DEBUG_TEST_LED_CONTROLS__
      numBytes = makeLEDcycleCmd( temp );      // create a command
      parseCommand( temp, numBytes );          // execute created cmd
   #endif

   checkOverforce( );
  }                                  // while(1) <<-- above
}
```

Snakebot_2560.h

```
/*  snakebot_2560.h
 */
#ifndef __SNAKEBOT_2560_H__
#define __SNAKEBOT_2560_H__

#include <WProgram.h>          // necessary for definition of 'uint8_t' type
#include <inttypes.h>

/*-------------------------------------------------------------------------------------
 *                          Prototypes
 *-------------------------------------------------------------------------------------*/



/*-------------------------------------------------------------------------------------
 *                          DEFINES
 *-------------------------------------------------------------------------------------*/
                          /* none of the force detectors indicate force>threshold */
#define NO_OVERFORCE 0
#define FORCE_LIMIT 100        /* arbitrary..tbd */

#define NUMCONFIGLIMIT 150      /* array size of LL_EEPROM moves */
#define CONFIG_ALL_CENTERED 255

#define ILLEGAL_NUMMOVES 255
```

```
#define MOVE_SUCCESSFUL 1
#define MOVE_CAUSED_OVERFORCE 2
#define MOVE_FAILED_OTHER 3
#define MOVE_BACKED_OUT_OF 4
#define MOVE_NOT_YET_TAKEN 255

#define BUFFSIZE 32          /* buffer size in bytes for both USB and TWI comms */


/*-------------------------------------------------------------------------------------
 *                          STRUCTS
 *-------------------------------------------------------------------------------------*/

typedef struct per_axis_2560_t{
                        // first five uint8_ts will not be overwritten
  uint8_t TWI_ID;           // there will be <=2 axes per TWI_ID
  uint8_t axisNum;          // this should also be the index of array
  int16_t axisPos;
  int16_t prevPos;
  uint8_t currentForce;     // updated by force AD ISR
  uint8_t forceThreshold;   // ?needed?
}per_axis_2560_t;

typedef struct LL_axesConfig_EE_t{
```

```
  uint8_t configIndex;
  uint8_t IndexPrevConfig;
  uint8_t IndexNextConfig;
  int16_t axisPos[ NUMAXES ];
}LL_axesConfig_EE_t;



#endif
```

Menu2560.c

```
/*
 *
 *   menu2560.c
 */
#include <WProgram.h>
#include <avr/io.h>       //This contains definitions for all the registers locations and some other things, must always be included
#include <util/delay.h>   //Contains some delay functions that will generate accurate delays of ms and us
#include <avr/interrupt.h>
#include <HardwareSerial.h>
#include <print.h>
#include <wire.h>
#include <ctype.h>                // for recognizing hexidecimal

#include <avr/pgmspace.h>         // for reading variables/strings from FLASH mem
#include <avr/eeprom.h>           // for reading/writing variables to/from EEPROM
#include <lightweightRingBuff.h>   // for TWI_ringBuf - ring buffer routines

#include <snakebot_common.h>
#include <snakebot_2560.h>
#include <snakebot_EEPROM2560.h>
```

```
/*-------------------------------------------------------------------
 *  printEEmenu
 *
 *  [0] = 'E'
 *  [1] = 'M'
 *  [2] = 'T' - print top level menu
 *        'B' - print Begin New Config menu
 *        'E' -
 */
byte printEEmenu(  byte * buf, byte numBytes ){

  byte submenu = 'M';
  if( numBytes > 2 ) submenu = buf[2];

  switch( submenu ){

    case 'A':
Serial.print_P(PSTR("\r\n    [0] = 'E' - read/write/goto axes configuration in EEPROM "));
Serial.print_P(PSTR("\r\n    [1] = 'A' - direct entry of axes' value"));
Serial.print_P(PSTR("\r\n    [2,3]       axis value to write to axis 0"));
Serial.print_P(PSTR("\r\n    [4,5]       axis value to write to axis 1"));
Serial.print_P(PSTR("\r\n    [6,7]       axis value to write to axis 2"));
Serial.print_P(PSTR("\r\n    [8,9]       axis value to write to axis 3"));
Serial.print_P(PSTR("\r\n    [A,B]       axis value to write to axis 4"));
Serial.print_P(PSTR("\r\n    [C,D]       axis value to write to axis 5"));
```

```
Serial.print_P(PSTR("\r\n    [E,F]       axis value to write to axis 6"));
Serial.print_P(PSTR("\r\n    [G,H]       axis value to write to axis 7"));
Serial.print_P(PSTR("\r\n    [I,J]       configuration index number"));
   break;

   case 'B':
Serial.print_P(PSTR("\r\n    [0] = 'E' - read/write/goto axes configuration in EEPROM "));
Serial.print_P(PSTR("\r\n    [1] = 'B' - begin new stored sequence at index [2,3]"));
Serial.print_P(PSTR("\r\n    [2,3]       index at which to begin new sequence"));
   break;

   case 'E':
Serial.print_P(PSTR("\r\n    [0] = 'E' - read/write/goto axes configuration in EEPROM "));
Serial.print_P(PSTR("\r\n    [1] = 'E' - add new config data to end of LL"));
Serial.print_P(PSTR("\r\n    [2,3]       index contained in LL to add to end of"));
   break;

   case 'I':
Serial.print_P(PSTR("\r\n    [0] = 'E' - read/write/goto axes configuration in EEPROM "));
Serial.print_P(PSTR("\r\n    [1] = 'I' - insert new config data into LL at index [2,3]"));
Serial.print_P(PSTR("\r\n    [2,3]       index after which new config is added"));
   break;

   case 'U':
Serial.print_P(PSTR("\r\n    [0] = 'E' - read/write/goto axes configuration in EEPROM "));
Serial.print_P(PSTR("\r\n    [1] = 'U' - update current axes config at index [2,3]"));
```

```
Serial.print_P(PSTR("\r\n    [2,3]       index of config to update"));
Serial.print_P(PSTR("\r\n                uses current axes' values to update EE"));
   break;

   case 'D':
Serial.print_P(PSTR("\r\n    [0] = 'E' - read/write/goto axes configuration in EEPROM "));
Serial.print_P(PSTR("\r\n    [1] = 'D' - remove config from LL"));
Serial.print_P(PSTR("\r\n    [2,3]       index of config to delete"));
   break;

   case 'X':
Serial.print_P(PSTR("\r\n    [0] = 'E' - read/write/goto axes configuration in EEPROM "));
Serial.print_P(PSTR("\r\n    [1] = 'X' - remove LL containing index [2,3]"));
Serial.print_P(PSTR("\r\n    [2,3]       index of config contained in LL to delete"));
   break;

   case 'Z':
Serial.print_P(PSTR("\r\n    [0] = 'E' - read/write/goto axes configuration in EEPROM "));
Serial.print_P(PSTR("\r\n    [1] = 'Z' - clear all EEPROM config data"));
   break;

   case 'L':
Serial.print_P(PSTR("\r\n    [0] = 'E' - read/write/goto axes configuration in EEPROM "));
Serial.print_P(PSTR("\r\n    [1] = 'L' - print List starting at Index"));
Serial.print_P(PSTR("\r\n    [2,3]       index of config contained in LL to list"));
   break;
```

```
   case 'N':
Serial.print_P(PSTR("\r\n    [0] = 'E' - read/write/goto axes configuration in EEPROM "));
Serial.print_P(PSTR("\r\n    [1] = 'N' - goto next config in LL "));
Serial.print_P(PSTR("\r\n              must have been preceded by a EGnn command"));
   break;

   case 'P':
Serial.print_P(PSTR("\r\n    [0] = 'E' - read/write/goto axes configuration in EEPROM "));
Serial.print_P(PSTR("\r\n    [1] = 'P' - goto prev config in LL "));
Serial.print_P(PSTR("\r\n              must have been preceded by a EGnn command"));
   break;

   case 'G':
Serial.print_P(PSTR("\r\n    [0] = 'E' - read/write/goto axes configuration in EEPROM "));
Serial.print_P(PSTR("\r\n    [1] = 'G' - goto configuration [2,3] "));
Serial.print_P(PSTR("\r\n    [2,3]      index of config to go to"));
   break;

   case 'C': break;
Serial.print_P(PSTR("\r\n    [0] = 'E' - read/write/goto axes configuration in EEPROM "));
Serial.print_P(PSTR("\r\n    [1] = 'C' - cycle between configs [2,3] and [4,5] for"));
Serial.print_P(PSTR("\r\n    [2,3]      config index of lower end of cycle"));
Serial.print_P(PSTR("\r\n    [4,5]      config index of upper end of cycle"));
Serial.print_P(PSTR("\r\n    [6,7]      number of iterations"));
Serial.print_P(PSTR("\r\n    [8]    'A'  alternate UP then DOWN cycle"));
```

```
Serial.print_P(PSTR("\r\n    [8]    'U'  cycle low->hi, goto LO, repeat  "));
Serial.print_P(PSTR("\r\n    [8]    'D'  cycle hi->low, goto HI, repeat   "));
Serial.print_P(PSTR("\r\n              THIS COMMAND IS CURRENT BROKEN.  someday... "));
   break;


   case 'H':
Serial.print_P(PSTR("\r\n    [0] = 'E' - read/write/goto axes configuration in EEPROM "));
Serial.print_P(PSTR("\r\n    [1] = 'H' - halt axes config cycle"));
   break;


   case 'M':                        // Top level menu
   default:
Serial.print_P(PSTR("\r\n printEEmenu"));
Serial.print_P(PSTR("\r\n    [0] = 'E' - read/write/goto axes configuration in EEPROM "));
Serial.print_P(PSTR("\r\n      Commands to Add new Configuration Data to EE"));
Serial.print_P(PSTR("\r\n    [1] = 'B' - begin new stored sequence at index [2,3]"));
Serial.print_P(PSTR("\r\n         'E' - add new config data to end of LL"));
Serial.print_P(PSTR("\r\n         'I' - insert new config data into LL at index [2,3]"));
Serial.print_P(PSTR("\r\n         'U' - update current axes config at index [2,3]"));
Serial.print_P(PSTR("\r\n      Commands to delete config data from EE"));
Serial.print_P(PSTR("\r\n         'D' - remove config from LL"));
Serial.print_P(PSTR("\r\n         'X' - remove LL"));
Serial.print_P(PSTR("\r\n         'Z' - clear array of all LL data"));
Serial.print_P(PSTR("\r\n      Commands to use data in EE"));
Serial.print_P(PSTR("\r\n         'L' - print List starting at Index"));
Serial.print_P(PSTR("\r\n         'G' - goto configuration [2,3] "));
```

```
Serial.print_P(PSTR("\r\n         'N' - goto next config in LL "));
Serial.print_P(PSTR("\r\n         'P' - goto prev config in LL"));
Serial.print_P(PSTR("\r\n         'C' - cycle between configs [2,3] and [4,5] "));
Serial.print_P(PSTR("\r\n         'A' - direct entry of axes' value"));
Serial.print_P(PSTR("\r\n         'M' - screen (USB) print menu"));
   break;

  }
 return TRUE;
}

/*---------------------------------------------------------------------
 *  printHelpMenu
 *
 *  [0] = 'H'
 */
byte printHelpMenu(byte * buf, byte numBytes ){

  byte submenu = 'H';
  if( numBytes > 1 ) submenu = buf[1];

  switch( submenu ){

      case 'M':
      case 'J':
Serial.print_P(PSTR("\r\n    [0] = 'M' - move cmd from PC"));
```

```
Serial.print_P(PSTR("\r\n    [1]   '1'  ascii-axis number '0'-'7'"));
Serial.print_P(PSTR("\r\n    [2]   'P' - move in Positive direction"));
Serial.print_P(PSTR("\r\n          'N' - move in Negative direction"));
Serial.print_P(PSTR("\r\n  [3,4]         step size "));
Serial.print_P(PSTR("\r\n  [5,6]  'S1'   speed->1  -- OPTIONAL DATA"));
      break;

      case 'L':
Serial.print_P(PSTR("\r\n    [0] L - indicates an LED on/off message"));
Serial.print_P(PSTR("\r\n    [1]   - port name.  Valid values are "));
Serial.print_P(PSTR("\r\n         F - overforce LEDs"));
Serial.print_P(PSTR("\r\n         T - top-of-axis limit LEDs"));
Serial.print_P(PSTR("\r\n         B - bottom-of-axis limit LEDs"));
Serial.print_P(PSTR("\r\n         W - TWI led:  cntrls LEDs on UNO slave boards"));
Serial.print_P(PSTR("\r\n    [2] O - on or off - boolean 1 or 0"));
Serial.print_P(PSTR("\r\n  [3,4] M - ascii-HEX bit-map of LEDs to turn on/off (2 bytes)  "));
Serial.print_P(PSTR("\r\n           each HI (==1) bit means to turn that LED ON"));
Serial.print_P(PSTR("\r\n           NOTE:  LEDs are LO-actuated:  writing that bit LO"));
Serial.print_P(PSTR("\r\n           turns the LED on.  "));
      break;

      case 'T':
Serial.print_P(PSTR("\r\n    [0] T - ping UNO"));
Serial.print_P(PSTR("\r\n    [1]   - axis number to ping or from whom ping rcvd"));
Serial.print_P(PSTR("\r\n    [2] P - 'ping' TWI_id that controls axis number"));
Serial.print_P(PSTR("\r\n         R   relay ping response to the PC via USB"));
```

```
        break;

        case 'P':
Serial.print_P(PSTR("\r\n    [0] P - print current axes' positions"));
Serial.print_P(PSTR("\r\n  [1,2]   - ascii-hex bitmap of axes to print"));
Serial.print_P(PSTR("\r\n           ex:  P03  for axis 1 and 2"));
Serial.print_P(PSTR("\r\n           P2F  for axes 6, 4, 3, 2, and 1"));
        break;

        case 'F':
Serial.print_P(PSTR("\r\n    [0] F - print current axes' force sensor readings"));
Serial.print_P(PSTR("\r\n  [1,2]   - ascii-hex bitmap of axes to print"));
Serial.print_P(PSTR("\r\n           ex:  P03  for axis 1 and 2"));
Serial.print_P(PSTR("\r\n           P2F  for axes 6, 4, 3, 2, and 1"));
        break;

        case 'C':
Serial.print_P(PSTR("\r\n    [0] C - Completed move msg from TWI slave"));
Serial.print_P(PSTR("\r\n    [1] 3 - axis number in decimal"));
Serial.print_P(PSTR("\r\n    [2] S - Status of Move "));
Serial.print_P(PSTR("\r\n    [3]'1'- MOVE_SUCCESSFUL "));
Serial.print_P(PSTR("\r\n           not a user-useful cmd...internal only"));
        break;

        case 'G':
Serial.print_P(PSTR("\r\n    [0] G - turn power pigtail on/off"));
```

```
Serial.print_P(PSTR("\r\n    [1] 0   turn off pigtail.  1 turns on"));
        break;

        case 'A':
Serial.print_P(PSTR("\r\n    [0] A - axes init routines.  Deprecated"));
        break;

        case 'E':  printEEmenu( buf, 0 );  break;

        case 'D':
Serial.print_P(PSTR("\r\n      [0] - 'D' delay command "));
Serial.print_P(PSTR("\r\n      [1] = 'S'   set timer to [2,3,4,5] "));
Serial.print_P(PSTR("\r\n           'P'   print timer_g (32b) "));
Serial.print_P(PSTR("\r\n           'D'   set delay-target to timer_g+[2,3,4,5] "));
Serial.print_P(PSTR("\r\n[2,3,4,5]        4-bytes of ascii-hex coded timer value "));
        break;

        case 'H':
        default:
Serial.print_P(PSTR("\r\n  [0]   = 'H' - print help menu"));
Serial.print_P(PSTR("\r\n    [1] = 'M' - move cmd from PC"));
Serial.print_P(PSTR("\r\n         'J' - move cmd from Joystick"));
Serial.print_P(PSTR("\r\n         'L' - LED on/off"));
Serial.print_P(PSTR("\r\n         'T' - ping"));
Serial.print_P(PSTR("\r\n         'P' - print current axes positions"));
Serial.print_P(PSTR("\r\n         'F' - print current force sensor values"));
```

```
Serial.print_P(PSTR("\r\n         'C' - move completed cmd"));
Serial.print_P(PSTR("\r\n         'G' - power pigtail on/off"));
Serial.print_P(PSTR("\r\n         'A' - init axes - DEPRECATED"));
Serial.print_P(PSTR("\r\n         'E' - EEPROM"));
Serial.print_P(PSTR("\r\n         'D' - delay"));
      break;
  }
  return TRUE;

}
```

Menu_2560.h

```
byte printHelpMenu(byte * buf, byte numBytes );

byte printEEmenu(  byte * buf, byte numBytes );
```

Led_2560.c

```
/*
   LED_2560.c
 */

#include <WProgram.h>
#include <stdlib.h>
#include <avr/io.h>       //This contains definitions for all the registers locations and some other things, must always be included
#include <util/delay.h>   //Contains some delay functions that will generate accurate delays of ms and us
#include <avr/interrupt.h>
#include <HardwareSerial.h>
#include <print.h>
#include <wire.h>
#include <ctype.h>                // for recognizing hexidecimal


#include <avr/io.h>
#include <avr/pgmspace.h>         // for reading variables/strings from FLASH mem
#include <avr/eeprom.h>           // for reading/writing variables to/from EEPROM

#include <lightweightRingBuff.h>   // for TWI_ringBuf - ring buffer routines

#include <snakebot_common.h>
```

```
#include <snakebot_2560.h>

/*-----------------------------------------------------------------------
 *  setupOverforceLEDs
 *
 *    this function sets up the over-force-indication LEDs as outputs then
 *    tests that they work by turning then on then off (blink)
 */
int setupOverforceLEDs( ){
  DDRA = 0xff;
  for( byte i=0; i<8; i++){
    PORTA = ~(1<<i);
    _delay_ms( 50 );
  }
  PORTA = 0xff;
  return TRUE;
}

/*-----------------------------------------------------------------------
 *  setupUpperLimitLEDs
 *
 *    this function sets up the axis-upper-limit-indication LEDs as outputs
 *    then tests that they work by turning then on then off (blink)
 */
int setupUpperLimitLEDs( ){
  DDRC = 0xff;
  for( byte i=0; i<8; i++){
    PORTC = ~(1<<i);
```

```
    _delay_ms( 50 );
  }
  PORTC = 0xff;
  return TRUE;
}
/*---------------------------------------------------------------------
 *  setupLowerLimitLEDs
 *
 *    this function sets up the axis-lower-limit-indication LEDs as outputs
 *    then tests that they work by turning then on then off (blink)
 */
int setupLowerLimitLEDs( ){
  DDRL = 0xff;
  for( byte i=0; i<8; i++){
    PORTL = ~(1<<i);
    _delay_ms( 50 );
  }
  PORTL = 0xff;
  return TRUE;
}
/*---------------------------------------------------------------------
 *  sendTWI_LED_msg
 *   args
 *     buf[0] - ascii-bool '1' == ON, '0' == OFF
 *     buf[1] - axis number
 *     buf[2] - UNO PortB pin number (can be 2 or 5)
```

```
*
*  sent_data:
*          ex: L21
*          [0] L - indicates an LED on/off message
*          [1]   - PortB bit number (2 and 5 are legal values)
*          [2]   - on or off:  ascii-boolean 1 or 0

*/
byte sendTWI_LED_msg( byte * buf, byte numBytes ){

  if(( buf[2] != '2' )&&( buf[2] != '5' )) return FALSE;

  byte axisNum = buf[1] - '0';

  byte TWI_dest = TWI_ID_From_AxisNum( axisNum );
  if( TWI_dest == I2C_FAILURE ) return FALSE;

  byte TWI_XmtBuf[5];

  TWI_XmtBuf[0] = 'L';
  TWI_XmtBuf[1] = buf[2];      // UNO PortB pin
  TWI_XmtBuf[2] = buf[0];      // OnOff
  TWI_XmtBuf[3] = SPACE;

  TWI_SendMessage( TWI_XmtBuf, TWI_dest, 4 );
```

```
  return TRUE;
}
/*---------------------------------------------------------------------
 *  set328LEDs
 *
 *     buf[0] - ascii-bool '1' == ON, '0' == OFF
 *     buf[1] - axis number
 *     buf[2] - UNO PortB pin number (can be 2 or 5)
 */
void set328LEDs( byte Green, byte Red, byte axisNum ){

  byte buf[3];

  buf[0] = Green;
  buf[1] = axisNum;
  buf[2] = '2';
  sendTWI_LED_msg( buf, 3 );
//echoToUSB( "GreenLed", buf, 3 );

  buf[0] = Red;
//buf[1] = axisNum;      // unnecessary: is same as set in Green code
  buf[2] = '5';
  sendTWI_LED_msg( buf, 3 );
//echoToUSB( "RedLed", buf, 3 );
}
/*---------------------------------------------------------------------
```

```
*  parseLEDcommand
*    args:  pointer to ascii buffer
*           number of bytes in buffer usable.  Must be == 5
*    returns:  TRUE if successful, FALSE if not
*
*     ex LF0F3:
*           [0] L - indicates an LED on/off message
*           [1] P - port name.  Valid values are
*                 F - overforce LEDs
*                 T - top-of-axis limit reached LEDs
*                 B - bottom-of-axis limit reached LEDs
*                 W - TWI led
*           [2] O - on or off - boolean 1 or 0
*          [3,4] M - ascii-HEX bit-map of LEDs to turn on/off (2 bytes)
*                   each HI (==1) bit means to turn that LED ON
*                   NOTE:  LEDs are LO-actuated:  writing that bit LO
*                   turns the LED on.
*/
byte parseLEDcommand( byte * buf, byte numBytes ){
                              // first verify args are valid
  if( numBytes < 5 ) return FALSE;
  if( buf[0] != 'L') return FALSE;

  byte PortName = buf[1];
  byte  OnOrOff = buf[2];
```

```
if(( PortName != 'F' )&&
   ( PortName != 'T' )&&
   ( PortName != 'B' )&&
   ( PortName != 'W' )) return FALSE;

if((  OnOrOff != '0' )&&(  OnOrOff != '1' )) return FALSE;

byte bitMap = convert2bAsciiHexToHex( &buf[3] );

switch( PortName ){                          // ON|OFF of LEDs
  case 'F':                                  // overforce indicator
    if( OnOrOff == '0' ) PORTA |=  bitMap;       // turn off LED:  bit HI
    else               PORTA &= ~bitMap;       // LED on:  bit LO
    break;                                   // no visible effect
  case 'T':                                  // top-of-axis-limit
    if( OnOrOff == '0' ) PORTC |=  bitMap;
    else               PORTC &= ~bitMap;
    break;
  case 'B':                                  // bottom-of-axis
    if( OnOrOff == '0' ) PORTL |=  bitMap;
    else               PORTL &= ~bitMap;
    break;
  case 'W':                                   // send TWI msg on/off LED
    return sendTWI_LED_msg( &buf[2], numBytes-2 );
    break;
  default:                                   // was checked above too
```

```
    return FALSE;
  }
  return TRUE;
}
/*---------------------------------------------------------------------
 *   build2560LEDcmnd( )
 */
void build2560LEDcmnd( byte PortName, byte OnOrOff, byte axisNum, byte * ptr ){

  ptr[0] = 'L';
  ptr[1] = PortName;
  ptr[2] = OnOrOff;
  convertHexTo2bAsciiHex(( 1<<axisNum ), &ptr[3] );
  ptr[5] = SPACE;
//echoToUSB( "build2560LEDcmnd", ptr, 6 );
}
/*---------------------------------------------------------------------
 *   control2560LEDs( )
 */
void control2560LEDs( byte PortName, byte OnOrOff, byte axisNum ){

  byte buf[6];
  build2560LEDcmnd( PortName, OnOrOff, axisNum, buf );
  parseLEDcommand( buf, 6 );
}
```

```
/*---------------------------------------------------------------------
 *  makeLEDcycleCmd
 *
 *    this function builds a LED-command to cycle through all LEDs on/off
 */
#ifdef __DEBUG_TEST_LED_CONTROLS__

byte makeLEDcycleCmd( byte * ptr ){

  static byte PortName = 'B', OnOrOff = '1', cnt = 0;

  build2560LEDcmnd( PortName, OnOrOff, cnt, ptr );

  if( OnOrOff == '1' ) {
    OnOrOff = '0';
  }
  else{
    OnOrOff = '1';
    if(( ++cnt ) >= 8 ){
      cnt = 0;
      switch( PortName ){
        case 'F':   PortName = 'T';      break;
        case 'T':   PortName = 'B';      break;
        case 'B':
        default:    PortName = 'F';      break;
      }
```

```
    }
  }
  return 6;
}
#endif
```

LED_2560.h

```
/*
   LED_2560.h
 */


#ifndef __LED_2560_H__
#define __LED_2560_H__
/*---------------------------------------------------------------------
 *  setupOverforceLEDs
 *
 *    this function sets up the over-force-indication LEDs as outputs then
 *    tests that they work by turning then on then off (blink)
 */
int setupOverforceLEDs( void );
int setupUpperLimitLEDs( void );
int setupLowerLimitLEDs( void );
byte sendTWI_LED_msg( byte * buf, byte numBytes );
void set328LEDs( byte Green, byte Red, byte axisNum );
byte parseLEDcommand( byte * buf, byte numBytes );
void build2560LEDcmnd( byte PortName, byte OnOrOff, byte axisNum, byte * ptr );
void control2560LEDs( byte PortName, byte OnOrOff, byte axisNum );
inline void turnOnOverforceLEDs( byte whichAxesAreOverforce );
inline void turnOffOverforceLEDs( void );

#ifdef __DEBUG_TEST_LED_CONTROLS__
  byte makeLEDcycleCmd( byte * ptr );
```

```
#endif


/*---------------------------------------------------------------------
 *   turnOnOverforceLEDs
 *
 *     args:  byte-map of which axes' force sensors indicate overforce
 *            note this fcn is inline since it is small
 *            note also this fcn assumes ALL of portA is overforce LEDs
 *            Note:  LEDs are LO-actuated.
 *            The bitmap arg sent is HI->turnLEDon.  Ergo inversion.
 */
inline void turnOnOverforceLEDs( byte whichAxesAreOverforce ) {
          // The bitmap arg sent is HI->turnLEDon.  Ergo inversion.
   PORTA = ~whichAxesAreOverforce;
}
/*---------------------------------------------------------------------
 *   turnOffOverforceLEDs
 *     args:  none
 *            note this fcn is inline since it is small
 *            note also this fcn assumes ALL of portA is overforce LEDs
 */
inline void turnOffOverforceLEDs( void ) {
  PORTA = 0xff;
}
#endif
```

Snakebot_EEPROM2560.c

```
/*
 *  Snakebot_EEPROM2560.c
 */
#include <WProgram.h>
#include <avr/io.h>      //This contains definitions for all the registers locations and some other things, must always be included
#include <util/delay.h>  //Contains some delay functions that will generate accurate delays of ms and us
#include <avr/interrupt.h>
#include <HardwareSerial.h>
#include <print.h>
#include <wire.h>
#include <ctype.h>               // for recognizing hexidecimal

#include <avr/pgmspace.h>        // for reading variables/strings from FLASH mem
#include <avr/eeprom.h>          // for reading/writing variables to/from EEPROM
#include <lightweightRingBuff.h>  // for TWI_ringBuf - ring buffer routines

#include <snakebot_common.h>
#include <snakebot_2560.h>
#include <snakebot_EEPROM2560.h>
#include <menu2560.h>

// ------------------------------------Global Variables -------------------------//
```

```
extern per_axis_2560_t per_axis_2560_g[ NUMAXES ];

axesConfig_t axesConfig_g;


// ------------------------------------EEPROM Variables --------------------------//
extern LL_axesConfig_EE_t LL_axesConfig_EE[ NUMCONFIGLIMIT ] EEMEM;


/*---------------------------------------------------------------------
 *  blurtConfig
 */
void blurtConfig( LL_axesConfig_EE_t * config_p ){

   blurt( "\r\nprev    = ",   config_p->IndexPrevConfig );
   blurt( "\r\nconfig  = ",   config_p->configIndex     );
   blurt( "\r\nnext    = ",   config_p->IndexNextConfig );

   for( byte i= 0; i<NUMAXES; i++ ){
     blurt("\r\n  axis #  ", i );
     blurt(" = ", config_p->axisPos[i] );
   }
}
/*---------------------------------------------------------------------
 *  blurtThatConfig
 */
```

```
void blurtThatConfig( byte index ){

  LL_axesConfig_EE_t config;

  eeprom_read_block( &config,
                     &LL_axesConfig_EE[ index ],
                      sizeof( LL_axesConfig_EE_t ));
  if( config.configIndex == MOVE_NOT_YET_TAKEN ) return;
  blurtConfig( &config );
}
/*---------------------------------------------------------------------
 *  gotoThisConfig
 */
byte gotoThisConfig( byte LLindex ){

  blurt( "\r\ngotoThisConfig: ", LLindex );

  LL_axesConfig_EE_t config;
  eeprom_read_block( &config,
                     &LL_axesConfig_EE[ LLindex ],
                      sizeof( LL_axesConfig_EE_t ));
  if( config.configIndex == MOVE_NOT_YET_TAKEN ) return FALSE;

  axesConfig_g.currentAxesConfig = LLindex;

  blurtConfig( &config );
```

```
  for( byte i=0; i< NUMAXES; i++ ){
    char buf[BUFFSIZE];
    byte TWI_dest = TWI_ID_From_AxisNum( i );
            byte axisNum = i+'0';
    byte numBytes = createMoveCmd( "M", buf, axisNum,config.axisPos[i] );
    TWI_SendMessage((byte*) buf, TWI_dest, numBytes );
//  echoToUSB( "moveSent: ", (byte*)buf, numBytes, __LINE__ );
  }
  return TRUE;
}
/*---------------------------------------------------------------------
 *   botIsAtThisConfig - used cut-n-paste of code from gotoThisConfig()
 */
byte botIsAtThisConfig( byte LLindex ){

  blurt( "\r\nbotIsAtThisConfig: ", LLindex );

  LL_axesConfig_EE_t config;
  eeprom_read_block( &config,
                  &LL_axesConfig_EE[ LLindex ],
                   sizeof( LL_axesConfig_EE_t ));
  if( config.configIndex == MOVE_NOT_YET_TAKEN ) return FALSE;

  axesConfig_g.currentAxesConfig = LLindex;
```

```
  blurtConfig( &config );

  for( byte i=0; i< NUMAXES; i++ ){
    char buf[BUFFSIZE];
    byte TWI_dest = TWI_ID_From_AxisNum( i );
          byte axisNum = i+'0';
    byte numBytes = createMoveCmd( "A", buf, axisNum,config.axisPos[i] );
    TWI_SendMessage((byte*) buf, TWI_dest, numBytes );
    echoToUSB( "Axis Count Assigned: ", (byte*)buf, numBytes, __LINE__ );
  }
  return TRUE;
}
/*---------------------------------------------------------------------
 *   goToCurrentConfig
 */
static inline byte goToCurrentConfig( void ){

  if( axesConfig_g.currentAxesConfig == MOVE_NOT_YET_TAKEN )
    return FALSE;

  return gotoThisConfig( axesConfig_g.currentAxesConfig );
}
/*---------------------------------------------------------------------
 *  gotoNextConfig
 *
 *    what:  if linked-list of moves has a valid next move, execute that move
```

```
 *     how:  checks to see if a next-move exists in LL in EE, exec that move
 */
byte gotoNextConfig(  ){

  byte index = axesConfig_g.currentAxesConfig;
  byte IndexNextConfig = eeprom_read_byte(
              &LL_axesConfig_EE[ index ].IndexNextConfig );

  if( IndexNextConfig == MOVE_NOT_YET_TAKEN ) return FALSE;

  return gotoThisConfig( IndexNextConfig );          // execute move
}
/*---------------------------------------------------------------------
 * gotoPrevConfig
 */
byte gotoPrevConfig(  ){

  byte index = axesConfig_g.currentAxesConfig;
  byte IndexPrevConfig = eeprom_read_byte(
              &LL_axesConfig_EE[ index ].IndexPrevConfig );

  if( IndexPrevConfig == MOVE_NOT_YET_TAKEN ) return FALSE;

  return gotoThisConfig( IndexPrevConfig );
}
```

```
/*---------------------------------------------------------------------
 *  printConfigSeq
 */
byte printConfigSeq( byte index ){

  if( index == NUMCONFIGLIMIT ) return FALSE;  // empty list

  LL_axesConfig_EE_t axesConfig;
  byte c=0;

  while(( index != MOVE_NOT_YET_TAKEN )&&(++c<NUMCONFIGLIMIT)){

    eeprom_read_block( &axesConfig,
                       &LL_axesConfig_EE[index],
                        sizeof( LL_axesConfig_EE_t ));

    blurtConfig( &axesConfig );

    if( index == axesConfig.IndexNextConfig ) return FALSE;

    index = axesConfig.IndexNextConfig;
  }
  return TRUE;
}
/*---------------------------------------------------------------------
 *   findUnusedConfigIndex
```

```
 */
inline byte findUnusedConfigIndex( void ){

  for( byte i=0; i<NUMCONFIGLIMIT; i++) {

    byte configIndex = eeprom_read_byte(
              &LL_axesConfig_EE[ i ].configIndex );

    if( configIndex == MOVE_NOT_YET_TAKEN ) return i;
  }
  return NUMCONFIGLIMIT;
}
/*---------------------------------------------------------------------
 *   findEndOfLL
 */
inline byte findEndOfLL( byte index ){

  byte IndexNextConfig, c=0;

  if( index >= NUMCONFIGLIMIT ) return NUMCONFIGLIMIT;

  while(( index != MOVE_NOT_YET_TAKEN )&&( ++c<NUMCONFIGLIMIT )){

    IndexNextConfig = eeprom_read_byte(
              &LL_axesConfig_EE[ index ].IndexNextConfig );
```

```
    if( IndexNextConfig == MOVE_NOT_YET_TAKEN ) return index;

    if( index == IndexNextConfig ) return NUMCONFIGLIMIT;

    index = IndexNextConfig;
  }
  return NUMCONFIGLIMIT;
}
/*----------------------------------------------------------------------
 *   findRootOfLL
 */
inline byte findRootOfLL( byte currIndex ){

  byte prevIndex = currIndex;
                        // find root of list containing [index]
  while(( prevIndex != MOVE_NOT_YET_TAKEN )&&( currIndex != 0 )){

    currIndex = prevIndex;
    prevIndex = eeprom_read_byte(
            &LL_axesConfig_EE[ currIndex ].IndexPrevConfig );
  }
  return currIndex;
}
/*----------------------------------------------------------------------
 *   writeConfigEE
 */
```

```
void writeConfigEE( byte prev, byte index, byte next ){

  if( index == MOVE_NOT_YET_TAKEN ) return;

  LL_axesConfig_EE_t config;

  config.IndexPrevConfig  = prev;
  config.configIndex      = index;
  config.IndexNextConfig  = next;

  for( byte i=0; i<NUMAXES; i++ ){
    config.axisPos[i] = per_axis_2560_g[i].axisPos;
  }
  eeprom_write_block( &config,
                      &LL_axesConfig_EE[ index ],
                      sizeof( LL_axesConfig_EE_t ));

  axesConfig_g.currentAxesConfig = index;
}
/*---------------------------------------------------------------------
 *   saveConfigAtEOLL
 */
byte saveConfigAtEOLL( byte index ){

  byte newIndex = findUnusedConfigIndex( );
  if( newIndex >= NUMCONFIGLIMIT ) return FALSE;
```

```
  byte lastElementIndex = findEndOfLL( index );
  if( lastElementIndex >= NUMCONFIGLIMIT ) return FALSE;

  writeConfigEE( lastElementIndex, newIndex, MOVE_NOT_YET_TAKEN );

  eeprom_write_byte(  &LL_axesConfig_EE[ lastElementIndex ].IndexNextConfig,
                    newIndex );

  blurt( "\r\n Saved config at EOLL at index = ",
                        axesConfig_g.currentAxesConfig );

  return TRUE;
}
/*---------------------------------------------------------------------
 *   updateConfig
 */
byte updateConfig( byte index ){

  if( index >= NUMCONFIGLIMIT ) return FALSE;  // note this test includes
                                    // if( MOVE_NOT_YET_TAKEN )
  byte prev = eeprom_read_byte( &LL_axesConfig_EE[ index ].IndexPrevConfig );
  byte next = eeprom_read_byte( &LL_axesConfig_EE[ index ].IndexNextConfig );

  writeConfigEE( prev, index, next );
```

```
 return TRUE;
}
/*---------------------------------------------------------------------
 *  insertConfig
 *
 *     insert into the LL of configuration elements a new element holding
 *     currrent axes config.  New element will be at position [index+1]
 */
byte insertConfig( byte index ){

  if( index >= NUMCONFIGLIMIT ) return FALSE;

  byte newConfigIndex = findUnusedConfigIndex();
  if( newConfigIndex == NUMCONFIGLIMIT ) return FALSE;

  byte newConfigNext = eeprom_read_byte(
                    &LL_axesConfig_EE[ index ].IndexNextConfig );

  writeConfigEE( index, newConfigIndex, newConfigNext );

  eeprom_write_byte( &LL_axesConfig_EE[ index ].IndexNextConfig,
                                        newConfigIndex );
  if( newConfigNext != MOVE_NOT_YET_TAKEN )
    eeprom_write_byte( &LL_axesConfig_EE[ newConfigNext ].IndexPrevConfig,
                                          newConfigIndex );
  return TRUE;
```

```
}
/*---------------------------------------------------------------------
 *   zapConfig
 */
void zapConfig( byte index ){

  if( index >= NUMCONFIGLIMIT ) return;

  LL_axesConfig_EE_t config;

  config.IndexPrevConfig  = MOVE_NOT_YET_TAKEN;
  config.configIndex      = MOVE_NOT_YET_TAKEN;
  config.IndexNextConfig  = MOVE_NOT_YET_TAKEN;

  eeprom_write_block( &config,
                      &LL_axesConfig_EE[ index ],
                      sizeof( LL_axesConfig_EE_t ));
}
/*---------------------------------------------------------------------
 *   removeLL
 *
 *     write MOVE_NOT_YET_TAKEN to indices of Prev,Current,Next struct
 *     Note:  fcn assumes first element (root) of LL has
 *            Prev=MOVE_NOT_YET_TAKEN and that last element (tail) has
 *            Next=MOVE_NOT_YET_TAKEN.
 */
```

```
byte removeLL( byte currIndex ){

  byte nextIndex, c=0;

  nextIndex = currIndex = findRootOfLL( currIndex );

  while(( nextIndex != MOVE_NOT_YET_TAKEN )&&( ++c < NUMCONFIGLIMIT )) {

    currIndex = nextIndex;
    nextIndex = eeprom_read_byte(
              &LL_axesConfig_EE[ currIndex ].IndexNextConfig );

    if( currIndex == nextIndex ) return FALSE;

    zapConfig( currIndex );
  }
  if( c < NUMCONFIGLIMIT ) return TRUE;

  return FALSE;
}
/*---------------------------------------------------------------------
 *   beginSaveSeq
 *
 *     this fcn starts a new axes-configuration-sequence at array element
 *     'index'.
 *
```

```
 *     Note:  if 'index' points to an already-used element in a sequence,
 *            the old sequence will be deleted
 */
byte beginSaveSeq( byte index ){

  if( index >= NUMCONFIGLIMIT ) return FALSE;

  byte isUsed = eeprom_read_byte( &LL_axesConfig_EE[ index ].configIndex );

  if( isUsed != MOVE_NOT_YET_TAKEN ) removeLL( index );

  writeConfigEE( MOVE_NOT_YET_TAKEN, index, MOVE_NOT_YET_TAKEN );

  return TRUE;
}

/*---------------------------------------------------------------------
 *  removeConfig
 *
 *    remove element from LL with result:
 *      prev element will point to element succeeding config being deleted
 *      successive element will point to element preceeding config deleted
 */
byte removeConfig( byte index ){

  if( index >= NUMCONFIGLIMIT ) return FALSE;
```

```
  byte prev = eeprom_read_byte( &LL_axesConfig_EE[index].IndexPrevConfig );
  byte next = eeprom_read_byte( &LL_axesConfig_EE[index].IndexNextConfig );

  if( prev != MOVE_NOT_YET_TAKEN)
    eeprom_write_byte( &LL_axesConfig_EE[ prev ].IndexNextConfig, next );
  if( next != MOVE_NOT_YET_TAKEN)
    eeprom_write_byte( &LL_axesConfig_EE[ next ].IndexPrevConfig, prev );

  zapConfig( index );

  return TRUE;
}

/*---------------------------------------------------------------------
 *   clearEEofConfigData
 */
byte clearEEofConfigData( void ){

  for( byte i=0; i<NUMCONFIGLIMIT; i++) zapConfig( i );

  return TRUE;
}
/*---------------------------------------------------------------------
 *   truncateLL
 *
```

```
 *     deletes linked list elements from current->next to EOLL
 */
byte truncateLL( void ){

  byte c=0, currIndex = axesConfig_g.currentAxesConfig;
  byte nextIndex = eeprom_read_byte(
            &LL_axesConfig_EE[ currIndex ].IndexNextConfig );

  while(( nextIndex != MOVE_NOT_YET_TAKEN )&&( ++c < NUMCONFIGLIMIT )) {

    currIndex = nextIndex;
    nextIndex = eeprom_read_byte(
            &LL_axesConfig_EE[ currIndex ].IndexNextConfig );

    zapConfig( currIndex );

    if( currIndex == nextIndex ) return FALSE;  // check for corrupted LL
  }
  if( c < NUMCONFIGLIMIT ) return TRUE;

  return FALSE;
}
/*---------------------------------------------------------------------
 *   editAxesValues
 *   [0] = 'E' - read/write/goto axes configuration in EEPROM
 *   [1] = 'A' - write axes values to EEPROM
```

```
*   [2,3]       axis value to write to axis 0
*   [4,5]       axis value to write to axis 1
*   [6,7]       axis value to write to axis 2
*   [8,9]       axis value to write to axis 3
*   [A,B]       axis value to write to axis 4
*   [C,D]       axis value to write to axis 5
*   [E,F]       axis value to write to axis 6
*   [G,H]       axis value to write to axis 7
*   [I,J]       configuration index number
*
*               this fcn assumes the target configuration at [index]
*               is already part of a linked list (ie, has prev, next)
*/
byte editAxesValues( byte * buf, byte numBytes ){

 if( numBytes < 20 ) return FALSE;

 byte i;
 LL_axesConfig_EE_t config;

 for( i=1; i<=NUMAXES; i++ ){
  config.axisPos[i-1] =  convert2bAsciiHexToHex( &buf[2*i] );
  if(( config.axisPos[i-1] > AXISLIMIT )||
     ( config.axisPos[i-1] < -1*AXISLIMIT )) return FALSE;
 }
 config.configIndex = convert2bAsciiHexToHex( &buf[2*i] );
```

```
  if( config.configIndex >= NUMCONFIGLIMIT ) return FALSE;
                              // note this test includes
                              // if( MOVE_NOT_YET_TAKEN )
  config.IndexPrevConfig =
        eeprom_read_byte( &LL_axesConfig_EE[
              config.configIndex ].IndexPrevConfig );
  config.IndexNextConfig =
        eeprom_read_byte( &LL_axesConfig_EE[
              config.configIndex ].IndexNextConfig );

  eeprom_write_block( &config,
                &LL_axesConfig_EE[ config.configIndex ],
                sizeof( LL_axesConfig_EE_t ));
  return TRUE;
}

/*---------------------------------------------------------------------
 *   verifyIndices
 *
 *    returns TRUE if both indices belong in same linked list of
 *    configurations
 */
byte verifyIndices(  ) {

  byte loIndex = axesConfig_g.loIndex;
  byte hiIndex = axesConfig_g.hiIndex;
```

```
 if(( loIndex == MOVE_NOT_YET_TAKEN )||
    ( hiIndex == MOVE_NOT_YET_TAKEN )||
    ( loIndex == hiIndex ))
   return FALSE;

 byte curr = loIndex, next;

 if( eeprom_read_byte( &LL_axesConfig_EE[ loIndex ].configIndex )
     == MOVE_NOT_YET_TAKEN ) return FALSE;

 do{
   next = eeprom_read_byte( &LL_axesConfig_EE[ curr ].IndexNextConfig );

   if( next == hiIndex ) return TRUE;

   curr = next;

 } while( next != MOVE_NOT_YET_TAKEN );

 return FALSE;
}
/*---------------------------------------------------------------------
 *  clearAxesConfigControl
 */
byte clearAxesConfigControl( void ){
```

```
  Serial.print_P(PSTR("\r\n  clearAxesConfigControl: "));

  axesConfig_g.upOrDown = SIDEWAYS;
  axesConfig_g.loIndex  = MOVE_NOT_YET_TAKEN;
  axesConfig_g.hiIndex  = MOVE_NOT_YET_TAKEN;
  axesConfig_g.iter     = CONFIG_CYCLE_ITER_LIMIT;
  axesConfig_g.i        = SIDEWAYS;

  return TRUE;
}
/*---------------------------------------------------------------------
 *  printAxesConfig
 */
byte printAxesConfig( void ){

  blurt( "current config index", axesConfig_g.currentAxesConfig );
//  blurt( "upOrDown", axesConfig_g.upOrDown );
  blurt( "loIndex", axesConfig_g.loIndex  );
  blurt( "hiIndex", axesConfig_g.hiIndex  );
//  blurt( "iteration limit", axesConfig_g.iter  );
//  blurt( "iteration count", axesConfig_g.i );
//  blurt( "recursion limit", axesConfig_g.recurseLim );

  LL_axesConfig_EE_t currentConfig, prevConfig;
```

```
  eeprom_read_block( &currentConfig,
      &LL_axesConfig_EE[  axesConfig_g.currentAxesConfig ],
                  sizeof( LL_axesConfig_EE_t ));
  if( currentConfig.configIndex == MOVE_NOT_YET_TAKEN ) return FALSE;

  eeprom_read_block( &prevConfig,
      &LL_axesConfig_EE[  currentConfig.IndexPrevConfig ],
                  sizeof( LL_axesConfig_EE_t ));
  if( prevConfig.configIndex == MOVE_NOT_YET_TAKEN ) return FALSE;

  Serial.write( "\r\nprev axes values are:" );
  blurtConfig( &prevConfig );
  Serial.write( "\r\ncurrent axes values are:" );
  blurtConfig( &currentConfig );

  return TRUE;
}
/*---------------------------------------------------------------------
 *  setupEEstruct
 *
 *    called from LEDtest2560 during setup routines
 */
void setupEEstruct( void ){

  axesConfig_g.currentAxesConfig = CONFIG_ALL_CENTERED;
  clearAxesConfigControl();
```

```
}
/*---------------------------------------------------------------------
 *   axesMoveToConfigCompleted
 *
 *      this function checks the global vars in axesConfig_g.
 *      compares the destination of a go-to-configuration move set
 *      against current positions of the axes and returns FALSE
 *      if any axis is still moving
 */
byte axesMoveToConfigCompleted( void ){

  byte retval = TRUE;
  byte index = axesConfig_g.currentAxesConfig;
  LL_axesConfig_EE_t config;
  eeprom_read_block( &config,
                     &LL_axesConfig_EE[ index ],
                      sizeof( LL_axesConfig_EE_t ));

  for( byte i=0; i<NUMAXES; i++ ){
    if( per_axis_2560_g[i].axisPos != config.axisPos[ i ] ){

      Serial.print_P(PSTR("\r\nconfig move pending"));
      blurt( " axis=", i );
      blurt( " currentPos=",per_axis_2560_g[i].axisPos );
      blurt( " destPos=",config.axisPos[ i ] );
            retval = FALSE;
```

```
    }
  }
  if( retval ) blurt( " \r\n....  No longer pending index ", index );

  return retval;
}
/*---------------------------------------------------------------------
 *  iterateAxesConfig
 *     [0] = 'E' - EEPROM command
 *     [1] = 'Q' - this function
 *           function uses axesConfig_g to check where in cycle of
 *           configurations currently positioned, iteration count,
 *           and if conditions are met, issue next move sequences.
 *           If conditions are not met, fcn places recursive call
 *           to itself in TWI queue OR quits if cycle complete
 */
byte iterateAxesConfig( byte * buf, byte numBytes ){

//  if( axesMoveToConfigCompleted( )){
    while( axesConfig_g.i < axesConfig_g.iter ){
  _delay_ms( 3000 );            // HACK - wait for prev move to fin

Serial.print_P(PSTR("\r\nmove completed..."));

    switch( axesConfig_g.upOrDown ){
            case UP:
```

```
            if( axesConfig_g.currentAxesConfig == axesConfig_g.hiIndex ){
Serial.print_P(PSTR("UP found top, calling gotoPrev"));
              axesConfig_g.upOrDown = DOWN;
              if( ! gotoPrevConfig( )){
Serial.print_P(PSTR("FAIL 001"));
                    return FALSE;
       }
      }
      else{
Serial.print_P(PSTR("UP calling gotoNext"));
       if(! gotoNextConfig( )){
Serial.print_P(PSTR("FAIL 002"));
                    return FALSE;
       }
           }
          break;
          case DOWN:
      if( axesConfig_g.currentAxesConfig == axesConfig_g.loIndex ){
       if( ++axesConfig_g.i >= axesConfig_g.iter ){
                    blurt("-> fin config_seq iter: ", axesConfig_g.iter );
                    return TRUE;
                   }
              axesConfig_g.upOrDown = UP;
Serial.print_P(PSTR("DOWN found bottom calling gotoNext"));
       if(! gotoNextConfig( )){
Serial.print_P(PSTR("FAIL 003"));
```

```
                    return FALSE;
                  }
      }
          else{
Serial.print_P(PSTR("DOWN calling gotoPrev"));
            if( ! gotoPrevConfig( )){
Serial.print_P(PSTR("FAIL 004"));
                    return FALSE;
       }
          }
        break;
        default:
Serial.print_P(PSTR("fubar-ed UP/DOWN"));

        return FALSE;
   }
 }
#if 0
 else{                    // waiting on pending config-moves
       if( ++axesConfig_g.recurseLim >= RECURSE_CALL_LIMIT ) {
    clearAxesConfigControl( );
Serial.print_P(PSTR("FAIL 005"));
    return FALSE;
       }                    // repeat move cmd.  Why not?
       else gotoThisConfig( axesConfig_g.currentAxesConfig );
 }
```

```
#endif

Serial.print_P(PSTR("exit iterateAxesConfig"));



//  TWI_placeInBuffer((uint8_t*) " EQ ", 4 );    // recursive self-call

  return TRUE;
}
/*----------------------------------------------------------------------
 *   cycleTweenConfigs
 *
 *   [0]    = 'E' - read/write/goto axes configuration in EEPROM
 *   [1]    = 'C' - cycle between configs [2,3] and [4,5] for
 *                  number of iterations
 *   [2]    = ','
 *   [3..]       low index to LL of configurations
 *   [next] = ','
 *   [next]       hi index
 *   [next] = ','
 *   [next]       number of iterations of sequence
 *
 */
byte cycleTweenConfigs( byte * buf, byte numBytes ){

  if( numBytes < 9 ) return FALSE;
```

```
  char * ptr = (char*) &buf[3];

  axesConfig_g.loIndex   = strtol( ptr, &ptr, 10 );  ptr++;
  axesConfig_g.hiIndex   = strtol( ptr, &ptr, 10 );  ptr++;
  axesConfig_g.iter      = strtol( ptr, &ptr, 10 );  ptr++;
  axesConfig_g.upOrDown  = UP;
  axesConfig_g.i         = ( 0 );
  axesConfig_g.recurseLim= ( 0 );

  printAxesConfig();

  if( verifyIndices(  ) != TRUE ){   // verify indices contiguous & valid
    Serial.print_P(PSTR("\r\n  cycleTweenConfigs:  indices err: "));
    echoToUSB( " ", buf, numBytes, __LINE__ );
    clearAxesConfigControl( );
    return FALSE;
  }
                              // arbitrary limit to keep sane
  if( axesConfig_g.iter >= CONFIG_CYCLE_ITER_LIMIT ){
      axesConfig_g.iter  = CONFIG_CYCLE_ITER_LIMIT;
  }
  gotoThisConfig( axesConfig_g.loIndex );

  return iterateAxesConfig(buf,  numBytes );
}
```

```
/*---------------------------------------------------------------------
 *  haltAxesConfigIter
 *
 *   [0] = 'E' - read/write/goto axes configuration in EEPROM
 *   [1] = 'H' - cycle between configs [2,3] and [4,5] for
 */
inline byte haltAxesConfigIter( void ){

  return clearAxesConfigControl( );
}
/*---------------------------------------------------------------------
 *  parseEEPROMdataReq
 *
 *   [0] = 'E' - read/write/goto axes configuration in EEPROM
 *            Commands to Add new Configuration Data to EE
 *   [1] = 'B' - begin new stored sequence at index [2,3]
 *         'E' - add new config data to end of LL
 *         'I' - insert new config data into LL at index [2,3]
 *         'U' - update current axes config at index [2,3]
 *            Commands to delete config data from EE
 *         'D' - remove config from LL
 *         'X' - remove LL
 *         'Z' - clear array of all LL data
 *            Commands to use data in EE
 *         'L' - print List starting at Index
 *         'G' - goto configuration [2,3]
```

```
*        'N' - goto next config in LL
*        'P' - goto prev config in LL
*        'C' - cycle between configs
*        'Q' - iterate config cycles - NEVER from outside srouce
*             should only be generated by cycleTweenConfigs()
*        'A' - edit axes' values
*        'M' - screen (USB) print menu
*   [2,3]  xx - ascii-coded-hex index of config for G,S,R
*
*             The source code for the functions called below
*             is in snakebot_EEPROM2560.c, protos in *.h
*/
byte parseEEPROMdataReq( byte * buf, byte numBytes ){

 if( buf[0] != 'E' ) return FALSE;
 if( numBytes < 2 )  return FALSE;

 echoToUSB( "parseEEPROMdataReq: ",buf, numBytes, __LINE__ );

 byte index = (byte) atoi((char*) &buf[2] );

 switch( buf[1] ){
   case 'B': return beginSaveSeq(     index );
   case 'E': return saveConfigAtEOLL( index );
   case 'I': return insertConfig(     index );
   case 'U': return updateConfig(     index );
```

```
    case 'D': return removeConfig(     index );
    case 'X': return removeLL(         index );
    case 'Z': return clearEEofConfigData(    );
            case 'T': return truncateLL(             );

    case 'L': return printConfigSeq(   index );
    case 'N': return gotoNextConfig(         );
    case 'P': return gotoPrevConfig(         );
    case 'G': return gotoThisConfig(   index );
            case 'A': return botIsAtThisConfig(index );
            case 'Y': return printAxesConfig(        );

    case 'C': return cycleTweenConfigs( buf, numBytes  );
//    case 'A': return editAxesValues(    buf, numBytes );
    case 'Q': return iterateAxesConfig( buf, numBytes );
    case 'H': return haltAxesConfigIter( );
    case 'M': return printEEmenu( buf, numBytes );

    default:  return FALSE;
  }
}
```

Snakebot_EEPROM2560.h

```
/*
 *
 *   Snakebot_EEPROM2560.h
 */
#ifndef _SNAKEBOT_EEPROM_H_
#define _SNAKEBOT_EEPROM_H_

byte gotoThisConfig(   byte index );
byte gotoNextConfig(   void );
byte gotoPrevConfig(   void );
byte printConfigSeq(   byte index );
byte saveConfigAtEOLL( byte index );
byte updateConfig(     byte index );
byte insertConfig(     byte index );
byte beginSaveSeq(     byte index );
byte removeConfig(     byte index );
byte removeLL(         byte currIndex );
byte clearEEofConfigData( void );
//axesConfigStatus_et axesMoveToConfigCompleted( void );
byte cycleTweenConfigs( byte * buf, byte numBytes );
byte editAxesValues(    byte * buf, byte numBytes );
byte iterateAxesConfig( byte * buf, byte numBytes );
byte haltAxesConfigIter( void );
void setupEEstruct( void );
```

```
byte readSelfCmdBuf( byte * buf );
byte writeSelfCmdBuf( byte * buf, byte numBytes );
byte parseEEPROMdataReq( byte * buf, byte numBytes );

#define TIME_LIMIT_ON_MOVE_TO_CONFIG 4  /* arbitrary */
#define CONFIG_CYCLE_ITER_LIMIT 22  /* arbitrary limit */
#define UP 'U'
#define DOWN 'D'
#define ALTERNATE 'A'
#define SIDEWAYS (0xff)
#define RECURSE_CALL_LIMIT 44


typedef struct axesConfig_t{

  byte currentAxesConfig;         // most recent index of move-to-config
  byte loIndex;                   // index of lowest config in cycle
  byte hiIndex;                   // index of highest config
  byte iter;                     // limit of iterations b/t lo-hi
  byte upOrDown;                  // currently traversing up or down LL
  byte i;                        // count of iterations so far
  byte recurseLim;                // count of recursive calls at config#

}axesConfig_t;

/*
```

```
static inline void setup2560SelfSendRingBuffer( void ){
  extern RingBuff_t selfSend_ringBuf;
  RingBuffer_InitBuffer( &selfSend_ringBuf );
};
*/

#ifndef _blurt_
#define _blurt_
/*-------------------------------------------------------------------------
 *  blurt
 */
static void inline blurt( const char * descr, byte data ){

    Serial.write( descr );
    Serial.print( data, DEC );
}
#endif

#ifndef _blurt_16
#define _blurt_16

static void inline blurt16( const char * descr, int data ){

    Serial.write( descr );
    Serial.print( data, DEC );
}
```

```
#endif




#endif /* eof */
```

Version.h

```
/*
 *  version.h
 *
 *    this file contains version numbers for both snakebot_328p
 *    and snakebot_2560 projects
 *
 *    Each time the version number is updated a comment-explanation
 *    of the changes associated should be included
 */

void setupVernum( byte vernum );    // code in snakebot_common.c



/* ----------------------  328p  ----------------------------- */

#define VERNUM_328P 3

        // 6/29/12 - added command 'A' which assigns axis count

//#define VERNUM_328P 2         // 6/27/12 added bugfix hack to solve
                                //        slowdown of moves AFMotor.cpp
                                                                //        also added __LINE__ to
debug
                                // 6/26/12 begin keeping versions
```

```
/* ----------------------  2560  ------------------------------ */

#define VERNUM_2560 3          // 6/29/12 added EEPROM command 'A'
                               //      which tells bot it is at config
                                                                  //      'index'
//#define VERNUM_2560 2        // 6/26/12 added EEPROM command 'T'
                               //      which truncates a LL
                                                                  //      added EEPROM
command 'Y'
                                                                  //      which prints current
index
//#define VERNUM_2560 1        /* 6/26/12 begin keeping versions */
```

Embedded C for the 328p

Snakebot_328p.c

```
/*
   snakebot_328p.c

    06 Jun 12 - added axis '8', the sled holding the baseplate with the other 8 axes
                - nuked axes' limit-checking...trying to escape grad school, not production
                                - reversed sign of step-increment ini checkJoystickDeflection
    06 Dec 11 - changed message passing paradigm:
                    1 - make move commands to absolute axes' values
                                    2 - add move index to each axes' move commands
                                    3 - add move-command-echo to axes above axis being moved
                                       so upper axes can update limits
            4 - eliminate error checking and re-sending
    16 Aug 11 - un-abandoned.  Bug was AFMotor.cpp <325,326> copied to <<333> and
<342,342> to <351>
                these lines enabled the latch to power motor shield, and were being disabled in
                subsequent calls.  Why did it work in IDE?
                Added Joystick calibration and storage of EE data (joystick specs, TWI_id, axis#).
    13 Aug 11 - abandoned.  Compiles, loads, runs.  Only one of two steppers moves.  Can't find
bug.

*/

#include <WProgram.h>
```

```
#include <stdlib.h>
#include <avr/io.h>       //This contains definitions for all the registers locations and some other things, must always be included
#include <util/delay.h>   //Contains some delay functions that will generate accurate delays of ms and us
#include <avr/interrupt.h>
#include <HardwareSerial.h>
#include <print.h>
#include <wire.h>
#include <ctype.h>                // for recognizing hexidecimal

#include <avr/pgmspace.h>         // for reading variables/strings from FLASH mem
#include <avr/eeprom.h>           // for reading/writing variables to/from EEPROM
#include <lightweightRingBuff.h>  // for TWI_ringBuf - ring buffer routines

#include <AFMotor.h>

#include <snakebot_328p.h>
#include <snakebot_common.h>
#include <version.h>              // version numbers

#define byte uint8_t

// ----------------------------------Global Variables -------------------------//
RingBuff_t TWI_ringBuf;           // receive ring buffer
byte TWI_Tbuf[BUFFSIZE];          // transmit buffer
```

```
byte TWI_ID_g;                  // value read from EEPROM at startup
axisStatus_t UDaxis_g;
axisStatus_t LRaxis_g;

volatile byte UDJoystick_g = 0;       // value read from the up/down joystick pot (0<=val<=255)
volatile byte LRJoystick_g = 0;
volatile byte TemperatureOfChip = 0;   // not currently used...but isn't it neat to be able?

AF_Stepper    LRMotor(200, 1);        // the left/right motor connects to motor shield motor port 2
AF_Stepper    UDMotor(200, 2);        // the up/down motor connects to motor shield motor port 1
                              // both have 200 steps/revolution

joyVals_t UDJoySpec_g, LRJoySpec_g;    // data read from EEPROM at startup

byte printJoyToUSB_g = 0;             // can be enabled in joystickEE(...)



//
// ------------------------------------EEPROM Variables --------------------------//
//
                              // these values should be written to EEPROM
                              // by a different executable...and are exptected
                              // to be valid by this executable
byte TWI_ID_EE    EEMEM;
byte axisNumUD_EE EEMEM;
```

```
byte axisNumLR_EE EEMEM;



joyVals_t UDJoySpec_EE EEMEM;
joyVals_t LRJoySpec_EE EEMEM;       // data read from EEPROM at startup



/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 *  ADC_vect
 *
 *    This ISR implements a round-robin alternation of AD inputs to read, and stores the AD values
 *    to global variables which are processed externally to the ISR.  The CurrentADInput set in the
 *    round-robin for the 'next' AD input
 *
 *    The registers that control the AD (and thereby implement the round-robin) are written at the end
 *    of the ISR.  It is at the end (instead of beginning) to allow the assignment circuitry time to settle
 *    between AD readings
 *
 *    on the 2560, there are 16 AD inputs (Ports F and K...make your own acronym).  Port F will be used
 *    to read the 8 axes' force sensors (F for force...duh).
 */
ISR( ADC_vect, ISR_BLOCK ){
```

```
        static byte currentADInput = UPDOWNJOYSTICK;            // round-robin index...arbitrary init
val
                                                // AD 'looks' at inputs to the mux
        switch( currentADInput ) {                      // cycling between inputs 0,1,and 8

          case LEFTRIGHTJOYSTICK:                       // currently reading the Left/Right joystick
            LRJoystick_g = ADCH;                        // store value in AD to global variable
            currentADInput = UPDOWNJOYSTICK;                // what will be read in next ISR
            break;

          case UPDOWNJOYSTICK:
            UDJoystick_g = ADCH;
            currentADInput = LEFTRIGHTJOYSTICK;
            break;

          case CHIPTEMPERATURE:
          default:
            TemperatureOfChip = ADCH;
            currentADInput = LEFTRIGHTJOYSTICK;
            break;
        }                                               // these lines set up for next AD source
        ADMUX &= CLEARMUXBITS;                          // clears righthandmost 4 bits
ADMUX.MUXnn
        ADMUX |= currentADInput;                        // sets or leaves clear ADMUX.MUXnn
                                                // note: time b/t interrupts for AD to settle
```

```
}

/*---------------------------------------------------------------------------------------------------
 *----------------------------------------------SETUP----------------------------------------------
 *---------------------------------------------------------------------------------------------------*/

/*-------------------------------------------------------------------
 *  setupLEDs
 */
void setupLEDs( void ){

  DDRB  |=  ( 1 << PINB2 );            // PIN 10 = PortB pin 2 = output
  DDRB  |=  ( 1 << PINB5 );            // PIN 13 = PortB pin 5 = output

  PORTB |=  ( 1 << PINB2 );             // turn pin 10 LED ON
  _delay_ms( 250 );
  PORTB &= ~( 1 << PINB2 );             // turn pin 10 LED OFF

  PORTB |=  ( 1 << PINB5 );             // turn pin 13 LED ON
  _delay_ms( 250 );
  PORTB &= ~( 1 << PINB5 );             // turn pin 13 LED OFF
}
/*-------------------------------------------------------------------
 *  setupGlobalsFromEE
 */
 void setupGlobalsFromEE( void ){
```

```
  TWI_ID_g         = eeprom_read_byte( &TWI_ID_EE );

  UDaxis_g.axisNum = eeprom_read_byte( &axisNumUD_EE );
  UDaxis_g.axisPos = 0;
  UDaxis_g.prevPos = 0;
  UDaxis_g.moveCount = 0;
  UDaxis_g.topLimit = AXISLIMIT;
  UDaxis_g.lowLimit = AXISLIMIT*(-1);

  LRaxis_g.axisNum = eeprom_read_byte( &axisNumLR_EE );
  LRaxis_g.axisPos = 0;
  LRaxis_g.prevPos = 0;
  LRaxis_g.moveCount = 0;
  LRaxis_g.topLimit = AXISLIMIT;
  LRaxis_g.lowLimit = AXISLIMIT*(-1);

  eeprom_read_block( &UDJoySpec_g, &UDJoySpec_EE, sizeof( joyVals_t ));
  eeprom_read_block( &LRJoySpec_g, &LRJoySpec_EE, sizeof( joyVals_t ));
}
/*---------------------------------------------------------------------
 *  setupSteppers
 */

void setupSteppers( void ){
```

```
  LRMotor.setSpeed(( uint16_t ) 7 );
  UDMotor.setSpeed(( uint16_t ) 7 );

  LRMotor.step( 2, FORWARD,  SINGLE );       // LR must move b/f UD to fix
  LRMotor.step( 2, BACKWARD, SINGLE );       // bug w/ PWM ctrl of driver IC

  UDMotor.step( 2, FORWARD,  SINGLE );
  UDMotor.step( 2, BACKWARD, SINGLE );
}
/*----------------------------------------------------------------------
 *  setup_2560
 */
void setup_328p( void ){

  Serial.begin(9600);

  setupGlobalsFromEE( );
  setupTWI( );
  setupTimer1( );
  setupAD( );
  setupLEDs( );
  setupSteppers( );
  setupVernum( VERNUM_328P );

  sei();
```

```
  Serial.print_P(PSTR("\r\n  snakebot_328p.c setup fin\r\nTWI id:  "));
  Serial.print( TWI_ID_g, DEC );
  Serial.print_P(PSTR("\r\n  "));
}
/*---------------------------------------------------------------------
 *-----------------------FUNCTIONS----------------------------------
 *---------------------------------------------------------------------*/


/*---------------------------------------------------------------------
 *  assignAxes
 */
byte assignAxes( void ){
  switch( TWI_ID_g ){
    case I2C_SLAVE_10:  UDaxis_g.axisNum = 0;  LRaxis_g.axisNum = 1; break;
    case I2C_SLAVE_32:  UDaxis_g.axisNum = 2;  LRaxis_g.axisNum = 3; break;
    case I2C_SLAVE_54:  UDaxis_g.axisNum = 4;  LRaxis_g.axisNum = 5; break;
    case I2C_SLAVE_76:  UDaxis_g.axisNum = 6;  LRaxis_g.axisNum = 7; break;
    case I2C_SLAVE_88:  UDaxis_g.axisNum = 8;  LRaxis_g.axisNum = 8; break;
    default:  return FALSE;
  }
  return TRUE;
}
/*---------------------------------------------------------------------
 *  updateLimits
 */
byte updateLimits( axisStatus_t * thisAxisStatus_p ,
```

```
                axisStatus_t * movedAxisStatus_p ){

  if( thisAxisStatus_p->axisNum <= movedAxisStatus_p->axisNum )
    return FALSE;

  thisAxisStatus_p->topLimit -= movedAxisStatus_p->prevPos;
  thisAxisStatus_p->topLimit += movedAxisStatus_p->axisPos;

  thisAxisStatus_p->lowLimit -= movedAxisStatus_p->prevPos;
  thisAxisStatus_p->lowLimit += movedAxisStatus_p->axisPos;

  return TRUE;
}
/*---------------------------------------------------------------------
 *  parseAxesLimUpdate
 *
 *    a lower axis sent notification that it moved.
 */
byte parseAxesLimUpdate( byte * buf, byte numBytes ){

  if( numBytes < 3 ) return FALSE;
  if(( buf[0] != 'U' )) return FALSE;
  axisStatus_t movedAxisStatus;
  convertAsciiAxisStatusToDec((char*) &buf[1], &movedAxisStatus );

  switch( TWI_ID_g ){
```

```
    case I2C_SLAVE_10:  return FALSE;    // axes 0&1 never change lim
    case I2C_SLAVE_32:
    case I2C_SLAVE_54:
    case I2C_SLAVE_76:  break;          // these are good...continue
            case I2C_MASTER_00:
    default:  return FALSE;             // bad problem.  Crash & burn
   }
   switch( movedAxisStatus.axisNum ){
    case 0:
            case 2:
            case 4: return updateLimits( &UDaxis_g, &movedAxisStatus );
    case 1:
    case 3:
            case 5: return updateLimits( &LRaxis_g, &movedAxisStatus );
            case 6:
            case 7:                     // uppermost two axes need tell no-one
    default:return FALSE;
   }
}
/*---------------------------------------------------------------------
 * parseAxesStatusReq
 *
 *   [0] = 'P'
 *   [1] = 'R' or 'A'
 *         'R' = request
 *         'A' = axis answer
```

```
 *   [2] = axis num ascii
 */
byte parseAxesStatusReq( byte * buf, byte numBytes ){

  if( numBytes < 3 ) return FALSE;
  if(( buf[0] != 'P' )||( buf[1] != 'R' )) return FALSE;
  byte axisNum = buf[2] - '0';
  axisStatus_t * axisStatus_p;

  if(      axisNum == UDaxis_g.axisNum ) { axisStatus_p = &UDaxis_g;  }
  else if( axisNum == LRaxis_g.axisNum ) { axisStatus_p = &LRaxis_g;  }
  else return FALSE;

  TWI_Tbuf[0] = 'P';
  TWI_Tbuf[1] = 'A';
  byte len = convertAxisStatusToAscii( axisStatus_p, (char*)&TWI_Tbuf[2] );
  TWI_SendMessage( TWI_Tbuf, I2C_MASTER_00, len + 2 );
  printAxisStatus_t( axisStatus_p );
//echoToUSB( "AxisStatusString: ", TWI_Tbuf, len+2 );
  return TRUE;
}
/*---------------------------------------------------------------------
 * notifyAxis
 *
 *   it is assumed the buffer TWI_Tbuf has been filled before this fcn call
 */
```

```
static inline void notifyAxis( byte * buf, byte TWIaddr, byte numBytes ){

  TWI_SendMessage( buf, TWIaddr, numBytes );
}
/*---------------------------------------------------------------------
 *   tellZed( )
 *
 *     inform TWI_id 00 (master, the 2560) of a move in this axis
 */
byte tellZed( byte axisNum ){

  axisStatus_t * axisStatus_p;

  if(      axisNum == UDaxis_g.axisNum ) axisStatus_p = &UDaxis_g;
  else if( axisNum == LRaxis_g.axisNum ) axisStatus_p = &LRaxis_g;
  else return FALSE;

  TWI_Tbuf[0] = 'U';
  byte len = convertAxisStatusToAscii( axisStatus_p, (char*)&TWI_Tbuf[1] );
  notifyAxis( TWI_Tbuf, I2C_MASTER_00,len );

  return TRUE;
}
/*---------------------------------------------------------------------
 *  clearOverLim
 */
```

```
void clearOverLim( axisStatus_t * axis_p ){

  byte mask = ( 1 << PINB2 );

  if( PORTB & mask ) {                    // if pin is ON

    PORTB &= ~mask;                       // turn pin 10 LED OFF

          mask = ( 1<< axis_p->axisNum );          // re-use local var
    byte buf[6] = "LT0";
          convertHexTo2bAsciiHex( mask, &buf[3] );
          buf[5] = 0;                     // NULL terminator
    TWI_SendMessage( buf, I2C_MASTER_00, 6 );   // turn off TOP led
          buf[1] = 'B';
    TWI_SendMessage( buf, I2C_MASTER_00, 6 );   // turn off BOTTOM led
  }
}
/*---------------------------------------------------------------------
 *  signalOverLim
 */
byte signalOverLim( axisStatus_t * axis_p,
                                        moveParms_t  * moveParms_p ){

  byte mask = ( 1<< axis_p->axisNum );
  byte buf[6] = "LT1";
  convertHexTo2bAsciiHex( mask, &buf[3] );
```

```
  buf[5] = 0;


  if( moveParms_p->axisDest > axis_p->topLimit ){
    TWI_SendMessage( buf, I2C_MASTER_00, 6 );   // turn on TOP led
  }
  else if( moveParms_p->axisDest < axis_p->lowLimit ){
            buf[1] = 'B';
    TWI_SendMessage( buf, I2C_MASTER_00, 6 );   // turn on BOTTOM led
  }
  else return FALSE;;


  PORTB |=  ( 1 << PINB2 );                  // turn pin 10 LED ON


  return TRUE;
}
/*---------------------------------------------------------------------
 *  moveThisAxis
 *
 *    axis_p points to the structure relating to one of the axes controlled
 *    by this 328p (either LR or UD).
 *    motor_p points to struct controlling the motor described by axis_p
 *    destPos is the axis count destination config int16_t signed int
 */
byte moveThisAxis( axisStatus_t * axis_p,
                AF_Stepper   * motor_p,
                                        moveParms_t  * moveParms_p ){
```

```
  uint8_t posOrNeg, stepSize;
  char ibuf[4];
  int16_t stepSize16;

//  printMoveParms( moveParms_p, __LINE__ );
//  printAxisStatus_t( axis_p );

  clearOverLim( axis_p );           // if over-limit LED is set, clear it

  switch( moveParms_p->axisDest ){
    case RELEASE_AXES:              // remove power to axes to allow
      motor_p->release();
      setupGlobalsFromEE( );        // zero-s position, offset, etc
            return TRUE;
    case INIT_AXES:
      setupGlobalsFromEE( );        // zero-s position, offset, etc
      setupSteppers( );             // power to steppers, shuffle
            setupLEDs();             // turn off status LEDs
            clearOverLim( axis_p );
            return TRUE;
    case ZERO_AXES:                 // move axes to position zero
            moveParms_p->axisDest = 0;
           default: break;
  }
//  printMoveParms( moveParms_p, __LINE__ );
```

```
/* no longer concerned with limit-checking...escape from school, not production code
  if(( moveParms_p->axisDest > axis_p->topLimit )||
     ( moveParms_p->axisDest < axis_p->lowLimit )){
    signalOverLim( axis_p, moveParms_p );
//  return signalOverLim( axis_p, moveParms_p );
  }
*/

  if( moveParms_p->axisDest < axis_p->axisPos ) {
    posOrNeg = BACKWARD;
    stepSize16 = axis_p->axisPos - moveParms_p->axisDest;
  }
  else if ( moveParms_p->axisDest > axis_p->axisPos ) {
    posOrNeg = FORWARD;
    stepSize16 = moveParms_p->axisDest - axis_p->axisPos;
  }
  else return tellZed( moveParms_p->axisNum );


  axis_p->prevPos = axis_p->axisPos;
  axis_p->axisPos = moveParms_p->axisDest;
  axis_p->moveCount++;
//  printAxisStatus_t( axis_p );

  do{
```

```
    stepSize = min( stepSize16, MAX_STEP_SIZE );
    stepSize16 -= stepSize;
    motor_p->step( stepSize, posOrNeg, SINGLE );
//    echoToUSB( "stepSize: ", (byte*) itoa( stepSize, ibuf, 10 ), 3, __LINE__ );
  }while( stepSize16  );

  return tellZed( moveParms_p->axisNum );
}
/*---------------------------------------------------------------------------
 *   parseMoveCommand( )
 *     args:  buf[] an array of byte ascii args from the USB or TWI
 *       buf[0]     = 'M' or 'J'
 *       buf[1]     = '1' == axis #1 == [0->7] ascii
 *       buf[2]     = ','
 *          [3,4,5] = ascii decimal axis destination value
 */
byte parseMoveCommand( byte * buf, byte numBytes ){

  moveParms_t moveParms;

  if( numBytes < 4 ) return FALSE;

  if( !parseMoveParms( &moveParms, buf )){
    Serial.print_P(PSTR("\r\n  FAILED parse <xYZaB>"));
    printMoveParms( &moveParms, __LINE__ );
    return FALSE;
```

```
  }
  printMoveParms( &moveParms, __LINE__ );

  if( moveParms.TWI_dest != TWI_ID_g ) return FALSE;

  if( moveParms.axisNum == UDaxis_g.axisNum ) {

    return moveThisAxis( &UDaxis_g, &UDMotor, &moveParms );
  }
  if( moveParms.axisNum == LRaxis_g.axisNum ) {

    return moveThisAxis( &LRaxis_g, &LRMotor, &moveParms );
  }
  return FALSE;
}

/*---------------------------------------------------------------------
 *  parseLEDcommand
 *    args:  pointer to ascii buffer
 *           number of bytes in buffer usable.  Must be == 5
 *    returns:  TRUE if successful, FALSE if not
 *           ex: L21
 *           [0] L - indicates an LED on/off message
 *           [1]   - PortB bit number (2 and 5 are legal values)
 *           [2]   - on or off:  ascii-boolean 1 or 0
 *                   NOTE:  LEDs are HI-actuated:  writing that bit HI
```

```
 *                  turns the LED on.
 */
byte parseLEDcommand( byte * buf, byte numBytes ){
                              // first verify args are valid
  if( numBytes < 3 ) return FALSE;
  if( buf[0] != 'L') return FALSE;

  byte PortBbitNum = buf[1] - '0';
  byte onOrOff = buf[2];

  if(( PortBbitNum != 5 )&&( PortBbitNum != 2)) return FALSE;

  if(      onOrOff == '1' )  PORTB |=  ( 1 << PortBbitNum );   // ON
  else if( onOrOff == '0' )  PORTB &= ~( 1 << PortBbitNum );   // OFF
  else return FALSE;

  return TRUE;
}

/*---------------------------------------------------------------------
 *  parsePingTWI
 *
 *    eg T3P  ping controller of axis #3
 *       T7R  response from controller of axis #7
 */
byte parsePingTWI( byte * buf, byte numBytes ){
```

```
  if( numBytes < 3 )  return FALSE;
  if( buf[0] != 'T' ) return FALSE;

  byte axisNum = buf[1] - '0';       // convert ascii->decimal
  byte TWI_dest = TWI_ID_From_AxisNum( axisNum );

  if( TWI_dest != TWI_ID_g ) return FALSE;
  if( buf[2] != 'P' )        return FALSE;

  buf[2] = 'R';                 // this TWI_slave was pinged.  Send reply.
  buf[3] = SPACE;
  echoToUSB( "Ping resp", buf, 4, __LINE__ );
  TWI_SendMessage( buf, I2C_MASTER_00, 4 );

  return TRUE;
}
/*---------------------------------------------------------------------
 *  createJoystickMoveReq
 *
 *     args:  buf[] an array of byte ascii args from the USB or TWI
 *       ex:  J1,-22  or J7,126
 *       buf[0]   = 'J'
 *       buf[1]   = '1' == axis #1
 *       buf[2,3] = destination axis count.  decimal-ascii
 */
```

```
inline static byte createJoystickMoveReq(
                        char     * buf,             // dest
                        byte       axisNum,         // ascii
                        int16_t    destAxisPos  ){  // hex


  return createMoveCmd( "J", buf, axisNum, destAxisPos  );
}
/*---------------------------------------------------------------------------
 *  calcStepSize
 */
byte calcStepSize( byte current, byte bottom, byte top ){


  byte range = top     - bottom;
  byte defl  = current - bottom;
  float percent = ((float)defl/(float)range);
  float step = percent * MAX_STEP_SIZE;
  return (byte) step;
}
/*---------------------------------------------------------------------------
 *  checkJoystickDeflection
 */
void checkJoystickDeflection( void ){


  byte axisNum, stepSize;
  int16_t destAxisPos;
```

```
if( UDJoystick_g > ( UDJoySpec_g.center + UDJoySpec_g.null_range )){

  stepSize = calcStepSize( UDJoystick_g,
                           UDJoySpec_g.center + UDJoySpec_g.null_range,
                           UDJoySpec_g.top );
  destAxisPos = UDaxis_g.axisPos - stepSize;
  axisNum     = UDaxis_g.axisNum + '0';
}
else if( UDJoystick_g < ( UDJoySpec_g.center - UDJoySpec_g.null_range )){

  stepSize = calcStepSize( 0xff - UDJoystick_g,
                           0xff - UDJoySpec_g.center + UDJoySpec_g.null_range,
                           0xff - UDJoySpec_g.bottom );
  destAxisPos = UDaxis_g.axisPos + stepSize;
  axisNum     = UDaxis_g.axisNum + '0';
}
else if( LRJoystick_g > ( LRJoySpec_g.center + LRJoySpec_g.null_range )){

  stepSize = calcStepSize( LRJoystick_g,
                           LRJoySpec_g.center + LRJoySpec_g.null_range,
                           LRJoySpec_g.top );
  destAxisPos = LRaxis_g.axisPos - stepSize;
  axisNum     = LRaxis_g.axisNum + '0';
}
else if( LRJoystick_g < ( LRJoySpec_g.center - LRJoySpec_g.null_range )){
```

```
    stepSize = calcStepSize( 0xff - LRJoystick_g,
                             0xff - LRJoySpec_g.center + LRJoySpec_g.null_range,
                             0xff - LRJoySpec_g.bottom );
    destAxisPos = LRaxis_g.axisPos + stepSize;
    axisNum     = LRaxis_g.axisNum + '0';
  }
  else return;           // joystick not deflected

  if( stepSize == 0 ) return;

  byte buf[BUFFSIZE],
       numBytes = createJoystickMoveReq( (char*)buf,
                                         axisNum,
                                         destAxisPos);
  parseMoveCommand( buf, numBytes );
  echoToUSB( "JoyGenMoveReq", TWI_Tbuf, numBytes, __LINE__ );
}
/*---------------------------------------------------------------------
 *  printJoyToUSB
 */
void printJoyToUSB( void ){

  Serial.print_P(PSTR("\r\n LR: ")); Serial.print( LRJoystick_g, DEC );
  Serial.print_P(PSTR(" UD: "));     Serial.print( UDJoystick_g, DEC );
}
/*---------------------------------------------------------------------
```

```
 *  printJoySpecs
 */
void printJoySpecs( void ){

  Serial.print_P(PSTR("\r\n LR: ")); Serial.print( LRJoySpec_g.top,        DEC );
  Serial.print_P(PSTR(", "));        Serial.print( LRJoySpec_g.center,     DEC );
  Serial.print_P(PSTR(", "));        Serial.print( LRJoySpec_g.bottom,     DEC );
  Serial.print_P(PSTR(", "));        Serial.print( LRJoySpec_g.null_range, DEC );

  Serial.print_P(PSTR("\r\n UD: ")); Serial.print( UDJoySpec_g.top,        DEC );
  Serial.print_P(PSTR(", "));        Serial.print( UDJoySpec_g.center,     DEC );
  Serial.print_P(PSTR(", "));        Serial.print( UDJoySpec_g.bottom,     DEC );
  Serial.print_P(PSTR(", "));        Serial.print( UDJoySpec_g.null_range, DEC );

}
/*---------------------------------------------------------------------
 *  joystickEE
 *
 *        buf[0] - 'U' - 'U' == up/down or
 *                 'L' == left/right
 *                 'C' == center
 *                 'G' == read from EE to globals
 *                 'P' == print current joystick AD to USB
 *                 'V' == print current joystick min/max/cntr vals
 *        buf[1] - 'T' - 'T' == store current AD as 'top'
 *                 'B' == store current AD as 'bottom'
```

```
 *                    '1' == set global printJoyToUSB_g to ON


 */
byte joystickEE( byte * buf ){

   Serial.print_P(PSTR("\r\n    Joystick_EE: "));
   Serial.write( buf, 2 );
   Serial.print_P(PSTR(" = "));

   switch( buf[0] ){

    case 'P':  printJoyToUSB_g = buf[1] - '0';   break;
    case 'V':  printJoySpecs();                  break;

    case 'G':                  // read from EE to globals
     eeprom_read_block( &UDJoySpec_g, &UDJoySpec_EE, sizeof( joyVals_t ));
     eeprom_read_block( &LRJoySpec_g, &LRJoySpec_EE, sizeof( joyVals_t ));
     Serial.print_P(PSTR(".  Read EE->RAM."));
     break;

    case 'U':
    case 'D':
     Serial.print( UDJoystick_g, DEC );
     switch( buf[1] ){
       case 'T':  eeprom_write_byte( &UDJoySpec_EE.top,    UDJoystick_g ); break;
       case 'B':  eeprom_write_byte( &UDJoySpec_EE.bottom, UDJoystick_g ); break;
```

```
   default:  Serial.print_P(PSTR(" FAILED"));                return FALSE;
  }
  break;

case 'L':
case 'R':
  Serial.print( LRJoystick_g, DEC );
  switch( buf[1] ){
    case 'T':  eeprom_write_byte( &LRJoySpec_EE.top,    LRJoystick_g ); break;
    case 'B':  eeprom_write_byte( &LRJoySpec_EE.bottom, LRJoystick_g ); break;
    default:   Serial.print_P(PSTR(" FAILED"));                return FALSE;
  }
  break;

case 'C':                  // write joystick center and null-range data
{                          // center is calc by averaging mult. samples
  static uint16_t sumCntrDataUD=0,
                  sumCntrDataLR=0;
  static byte     numCntrData=0;

  sumCntrDataLR += LRJoystick_g;
  sumCntrDataUD += UDJoystick_g;
  numCntrData++;

  byte avgCntrLR = (byte)( sumCntrDataLR / numCntrData );
  byte avgCntrUD = (byte)( sumCntrDataUD / numCntrData );
```

```
        eeprom_write_byte( &LRJoySpec_EE.center, avgCntrLR );
        eeprom_write_byte( &UDJoySpec_EE.center, avgCntrUD );

        Serial.print_P(PSTR(", LR.center: ")); Serial.print( avgCntrLR, DEC );
        Serial.print_P(PSTR(", UD.center: ")); Serial.print( avgCntrUD, DEC );

#ifdef NUKED

        int8_t UDdeviation = ( avgCntrUD - UDJoystick_g );
        int8_t LRdeviation = ( avgCntrLR - LRJoystick_g );

        static uint16_t       sumNulldeviationUD  = 0,
                              sumNulldeviationLR  = 0;
        if( UDdeviation < 0 ) sumNulldeviationUD -= UDdeviation;
        else                  sumNulldeviationUD += UDdeviation;
        if( LRdeviation < 0 ) sumNulldeviationLR -= LRdeviation;
        else                  sumNulldeviationLR += LRdeviation;

        byte avgDevUD = (byte)( sumNulldeviationUD / numCntrData );
        byte avgDevLR = (byte)( sumNulldeviationLR / numCntrData );

        eeprom_write_byte( &LRJoySpec_EE.null_range, 2*avgDevLR );
        eeprom_write_byte( &UDJoySpec_EE.null_range, 2*avgDevUD );

        Serial.print_P(PSTR(", LR.null: ")); Serial.print( avgDevLR, DEC );
```

```
      Serial.print_P(PSTR(", UD.null: ")); Serial.print( avgDevUD, DEC );
#else
      byte defaultDev = 20;
      eeprom_write_byte( &LRJoySpec_EE.null_range, defaultDev );
      eeprom_write_byte( &UDJoySpec_EE.null_range, defaultDev );
#endif

      break;
    }
    default:  return FALSE;
   }
  return TRUE;
}
/*---------------------------------------------------------------------
 *  TWI_idEE
 *
 *     calculates TWI_ID, assigns to global var, then calls assignAxes()
 *     which calculates axes numbers, writes results to EEPROM
 *       eg:  ET32
 *         buf[0,1] - TWI_id in ascii-hex
 */
byte TWI_idEE( byte * buf_p, byte numBytes ){

    if( numBytes < 4 ) return false;
    byte TWIid = convert2bAsciiHexToHex( buf_p );
```

```
    if( !TWIidIsValid( TWIid )){
      Serial.print_P(PSTR("\r\nINVALID TWI_id = "));
                    Serial.print( TWIid,  DEC );
      return FALSE;
    }

    TWI_ID_g = TWIid;
    assignAxes();                  // all vars are globals so no args

    Serial.print_P(PSTR("\r\n    TWI_id = "));
          Serial.print( TWI_ID_g,    DEC );
    Serial.print_P(PSTR("\r\n    axisNumUD_g = "));
          Serial.print( UDaxis_g.axisNum, DEC );
    Serial.print_P(PSTR("\r\n    axisNumLR_g = "));
          Serial.print( LRaxis_g.axisNum, DEC );

    eeprom_write_byte( &TWI_ID_EE,    TWI_ID_g );
    eeprom_write_byte( &axisNumUD_EE, UDaxis_g.axisNum );
    eeprom_write_byte( &axisNumLR_EE, LRaxis_g.axisNum );

    return TRUE;
}
/*---------------------------------------------------------------------
 *  parseEEdataAssign
 *
 *    this function presents options to the user via USB s.t. user selects:
```

```
 *
 *     TWI_ID, from which axes numbers are calculated
 *       eg:  ET32
 *         buf[1]   - 'T' - TWI_id set in EEPROM
 *         buf[2,3] - TWI_id in ascii-hex
 *
 *     joystick Up/Down min/max/center
 *       eg:  EJUT
 *         buf[1] - 'J' - set joystick values
 *         buf[2] - 'T' - TWI_idEE
 *
 */
byte parseEEdataAssign( byte * buf, byte numBytes ){

  if( buf[0] != 'E' ) return FALSE;
  if( numBytes < 3 ) return FALSE;

  Serial.print_P(PSTR("\r\n  Begin Assign EE data vals"));

  switch( buf[1] ){
    case 'J':  return joystickEE( &buf[2] );
    case 'T':  return TWI_idEE(   &buf[2], numBytes );
    default:   return FALSE;
  }
}
/*---------------------------------------------------------------------
```

```
 *  parsePigtailOnOff
 */
byte parsePigtailOnOff( byte * buf, byte numBytes ){

  if( buf[0] != 'G' ) return FALSE;
  if( numBytes < 2 ) return FALSE;
  if( isnan( buf[1] )) return FALSE;
  buf[2] = SPACE;
  TWI_SendMessage( buf, I2C_MASTER_00, 3 );
  return TRUE;
}
/*---------------------------------------------------------------------
 *  parseAxisPosAsgn
 *
 *              A6,-133
 *    args[0] = 'A'
 *        [1] = ascii axis number [0-8]
 *        [2] = ','
 *        [3] = ascii axis-count
 *
 */
byte parseAxisPosAsgn( byte * buf, byte numBytes ){

  if( buf[0] != 'A' ) return FALSE;
  if( numBytes < 2 ) return FALSE;
  if( isnan( buf[1] )) return FALSE;
```

```
  uint8_t axisNum = buf[1] - '0';
  axisStatus_t * axisStatus_p;
  if     ( axisNum == UDaxis_g.axisNum ) axisStatus_p = &UDaxis_g;
  else if( axisNum == LRaxis_g.axisNum ) axisStatus_p = &LRaxis_g;
  else return FALSE;

  int16_t axisPos = atoi((const char*) &buf[3] );
  if(( axisPos > axisStatus_p->topLimit ) ||
     ( axisPos < axisStatus_p->lowLimit ))   return FALSE;

  axisStatus_p->axisPos = axisPos;

  printAxisStatus_t( axisStatus_p );

  return TRUE;
}
/*---------------------------------------------------------------------
 *  parseCommand
 *
 *     if the first letter of the array 'buf' contains a valid command-type
 *     and the command parses and executes in subsequent subroutines
 *     correctly, this function returns TRUE.  If not, this function
 *     echoes the argument-string 'buf' to the USB port and returns FALSE
 */
byte parseCommand( byte * buf, byte numBytes ){
```

```
  switch( buf[0] ){

    case 'M':
    case 'J':
      if( parseMoveCommand(   buf, numBytes  ))    return TRUE;
      break;
    case 'T':
      if( parsePingTWI(       buf, numBytes ))     return TRUE;
      break;
    case 'L':
      if( parseLEDcommand(    buf, numBytes ))     return TRUE;
      break;
    case 'E':
      if( parseEEdataAssign(  buf, numBytes ))     return TRUE;
      break;
//  case 'U':
//    if( parseAxesLimUpdate( buf, numBytes ))     return TRUE;
//    break;
    case 'P':
      if( parseAxesStatusReq( buf, numBytes ))     return TRUE;
      break;
    case 'G':
      if( parsePigtailOnOff(  buf, numBytes ))     return TRUE;
      break;
    case 'A':
```

```
        if( parseAxisPosAsgn(   buf, numBytes ))     return TRUE;
        break;

      default:
        break;
          }
   echoToUSB( "Parse Failed", buf, numBytes, __LINE__ ); // default case and error-handler
          return FALSE;
}

/*---------------------------------------------------------------------
 *
 */

int main( void )
{
  byte numBytes, temp[ BUFFSIZE ];

  setup_328p( );

  while(1){

    if( Serial.available()){                    // check for USB comms

      numBytes = readUSBmsg( temp );
      parseCommand( temp, numBytes );
```

```
    }
    else if( RingBuffer_GetCount( &TWI_ringBuf )){ // check for TWI message


      numBytes = readTWImsg( temp );
      parseCommand( temp, numBytes );
    }
    else{                                  // USB-out heartbeat dots
      heartbeatDots(  );
      _delay_ms( 500 );
    }
    if( printJoyToUSB_g ) printJoyToUSB();         // calibrate joystick
    else     checkJoystickDeflection( );
  }
}
```

Snakebot_328p.h

```
/*  snakebot_328p.h
 */
#include <WProgram.h>          // necessary for definition of 'uint8_t' type
#include <inttypes.h>

/* there are 8 axes, numbered 0->7  */
#define NUMAXES 8
                        /* move state variable in per_axis_2560_t */
#define AXIS_STATE_SILL 1
#define AXIS_STATE_MOVE_SENT 2

#define MOVE_SUCCESSFUL 1
#define MOVE_CAUSED_OVERFORCE 2
#define MOVE_FAILED_OTHER 3
#define MOVE_BACKED_OUT_OF 4
#define MOVE_REFUSED_AXIS_AT_LIMIT
#define MOVE_NOT_YET_TAKEN 255

#define TIMER1TRIGGERTIME 0X4ff              // increase this value increases interval b/t A/D
conversions

#define CLEARMUXBITS      0xf0               // mux assignment bits occur in MS nibble of LS
byte
#define LEFTRIGHTJOYSTICK  0                 // ADMUX.MUXnnnn: selects multiplexed input
to A/D
```

```
#define UPDOWNJOYSTICK     1                  // AD1
#define CHIPTEMPERATURE    8                  // AD8
                                              // AD 4 and 5 implement TWI signals SDA and SLA (resp)
#define UPDOWNJOYNULL      10                 // center value +- this value is values where no
action taken
#define LRJOYNULL          12                 // center value +- this value is values where no action
taken


typedef struct joyVals_t_ {
  unsigned char top;
  unsigned char center;
  unsigned char bottom;
  unsigned char null_range;
} joyVals_t;
```

AFMotor.cpp

Originally, this was an Arduino library file, but in the process of writing the snakebot software, this file got edited extensively and corrections to the code were offered to the Arduino codebase.

```
// Adafruit Motor shield library
// copyright Adafruit Industries LLC, 2009
// this code is public domain, enjoy!
//
//  14 Aug 11 - re-arranged stepper init function to enable Stepper2 latch-power (JKE)



#include <avr/io.h>
#include <util/delay.h>
#include "WProgram.h"
#include "AFMotor.h"

#ifndef delay
#define delay( msec ) _delay_ms( msec )
#endif

static uint8_t latch_state;
//static uint32_t staticSaveduSperStep;

#if (MICROSTEPS == 8)
uint8_t microstepcurve[] = {0, 50, 98, 142, 180, 212, 236, 250, 255};
#elif (MICROSTEPS == 16)
```

```
uint8_t microstepcurve[] = {0, 25, 50, 74, 98, 120, 141, 162, 180, 197, 212, 225, 236, 244, 250,
253, 255};
#endif

AFMotorController::AFMotorController(void) { }

void AFMotorController::enable(void) {

  pinMode(MOTORLATCH,  OUTPUT);    // setup the latch
  pinMode(MOTORENABLE, OUTPUT);
  pinMode(MOTORDATA,   OUTPUT);
  pinMode(MOTORCLK,    OUTPUT);

  latch_state = 0;
  latch_tx();                  // "reset"

  digitalWrite(MOTORCLK, LOW);     // data is clocked through the 74CTH595N on the rising
edge of the
                               // clock.  We need to make sure the clock input is LOW before
                               // enabling the chip
  digitalWrite(MOTORENABLE, HIGH); // writing HI should clear the latch of any state caused
by
  digitalWrite(MOTORENABLE, LOW);  // transient voltages during power-on
}
```

```
void AFMotorController::latch_tx(void) {

  digitalWrite(MOTORLATCH, LOW);
  digitalWrite(MOTORDATA,  LOW);

  for (uint8_t i=0; i<8; i++) {

    digitalWrite(MOTORCLK, LOW);

    if (latch_state & _BV(7-i)) {  digitalWrite( MOTORDATA, HIGH); }
    else {                         digitalWrite( MOTORDATA, LOW);  }

    digitalWrite(MOTORCLK, HIGH);
  }
  digitalWrite(MOTORLATCH, HIGH);
}

static AFMotorController MC;



/****************************************
            MOTORS
****************************************/
inline void initPWM1(uint8_t freq) {    // use PWM from timer2A on PB3 (Arduino pin #11)
    TCCR2A |= _BV(COM2A1) | _BV(WGM20) | _BV(WGM21); // fast PWM, turn on oc2a
    TCCR2B = freq & 0x7;
```

```
    OCR2A = 0;
    pinMode(11, OUTPUT);
}
inline void setPWM1(uint8_t s) {       // use PWM from timer2A on PB3 (Arduino pin #11)
    OCR2A = s;
}

inline void initPWM2(uint8_t freq) {    // use PWM from timer2B (pin 3)
    TCCR2A |= _BV(COM2B1) | _BV(WGM20) | _BV(WGM21); // fast PWM, turn on oc2b
    TCCR2B = freq & 0x7;
    OCR2B = 0;
    pinMode(3, OUTPUT);
}
inline void setPWM2(uint8_t s) {       // use PWM from timer2A on PB3 (Arduino pin #3)
    OCR2B = s;
}

inline void initPWM3(uint8_t freq) {    // use PWM from timer0A / PD6 (pin 6)
    TCCR0A |= _BV(COM0A1) | _BV(WGM00) | _BV(WGM01); // fast PWM, turn on OC0A
    //TCCR0B = freq & 0x7;
    OCR0A = 0;
    pinMode(6, OUTPUT);
}
inline void setPWM3(uint8_t s) {       // use PWM from timer0A on PB3 (Arduino pin #6)
    OCR0A = s;
}
```

```
inline void initPWM4(uint8_t freq) {    // use PWM from timer0B / PD5 (pin 5)
   TCCR0A |= _BV(COM0B1) | _BV(WGM00) | _BV(WGM01); // fast PWM, turn on oc0a
   //TCCR0B = freq & 0x7;
   OCR0B = 0;
   pinMode(5, OUTPUT);
}
inline void setPWM4(uint8_t s) {        // use PWM from timer0A on PB3 (Arduino pin #6)
   OCR0B = s;
}

/*****************************************
            STEPPERS
*****************************************/

AF_Stepper::AF_Stepper(uint16_t steps, uint8_t num) {
  MC.enable();

  revsteps = steps;
  steppernum = num;
  currentstep = 0;

  if (steppernum == 1) {

    latch_state &= ~_BV(MOTOR1_A) &
                   ~_BV(MOTOR1_B) &
```

```
            ~_BV(MOTOR2_A) &
            ~_BV(MOTOR2_B); // all motor pins to 0
  MC.latch_tx();

  initPWM1(MOTOR12_64KHZ);      // use PWM for microstepping support
  initPWM2(MOTOR12_64KHZ);
  setPWM1(255);
  setPWM2(255);

  digitalWrite(11, HIGH);       // enable both H bridges
  digitalWrite( 3, HIGH);
}
else if (steppernum == 2) {

  latch_state &= ~_BV(MOTOR3_A) &
            ~_BV(MOTOR3_B) &
            ~_BV(MOTOR4_A) &
            ~_BV(MOTOR4_B); // all motor pins to 0
  MC.latch_tx();

  initPWM3(1);                  // use PWM for microstepping support
  initPWM4(1);
  setPWM3(255);
  setPWM4(255);

  digitalWrite(5, HIGH);        // enable both H bridges
```

```
    digitalWrite(6, HIGH);
  }
}

void AF_Stepper::setSpeed(uint16_t rpm) {
  usperstep = 60000000 / ((uint32_t)revsteps * (uint32_t)rpm);
  steppingcounter = 0;
  backupusperstep = usperstep;
//Serial.print(">>setSpeed.usperstep = "); Serial.print(usperstep, DEC);
}

void AF_Stepper::release(void) {

  if (steppernum == 1) {
    latch_state &= ~_BV(MOTOR1_A) &
                  ~_BV(MOTOR1_B) &
                  ~_BV(MOTOR2_A) &
                  ~_BV(MOTOR2_B); // all motor pins to 0
    MC.latch_tx();
  }
  else if (steppernum == 2) {
    latch_state &= ~_BV(MOTOR3_A) &
                  ~_BV(MOTOR3_B) &
                  ~_BV(MOTOR4_A) &
                  ~_BV(MOTOR4_B); // all motor pins to 0
    MC.latch_tx();
```

```
  }
}

void AF_Stepper::step(uint16_t steps, uint8_t dir,  uint8_t style) {

  // hack to work around bug where usperstep gets written to 50x normal
  if( usperstep != backupusperstep ) usperstep = backupusperstep;

  uint32_t uspers = usperstep;
  uint8_t ret = 0;

  if( style == INTERLEAVE ) {
    uspers /= 2;
  }
  else if (style == MICROSTEP)  {
    uspers /= MICROSTEPS;
    steps *= MICROSTEPS;
  }



  while (steps--) {
    ret = onestep(dir, style);
    delay(uspers/1000); // in ms
    steppingcounter += (uspers % 1000);
//  Serial.print(" uspers = ");       Serial.print(uspers, DEC);
```

```
    if (steppingcounter >= 1000) {
      delay(1);
      steppingcounter -= 1000;

    }
  }
  if (style == MICROSTEP) {
    while ((ret != 0) && (ret != MICROSTEPS)) {
      ret = onestep(dir, style);
      delay(uspers/1000); // in ms
      steppingcounter += (uspers % 1000);
      if (steppingcounter >= 1000) {
        delay(1);
        steppingcounter -= 1000;
      }
    }
  }
}

uint8_t AF_Stepper::onestep(uint8_t dir, uint8_t style) {
  uint8_t a, b, c, d;
  uint8_t ocrb, ocra;

  ocra = ocrb = 255;

  if (steppernum == 1) {
```

```
  a = _BV(MOTOR1_A);
  b = _BV(MOTOR2_A);
  c = _BV(MOTOR1_B);
  d = _BV(MOTOR2_B);
} else if (steppernum == 2) {
  a = _BV(MOTOR3_A);
  b = _BV(MOTOR4_A);
  c = _BV(MOTOR3_B);
  d = _BV(MOTOR4_B);
} else {
  return 0;
}

// next determine what sort of stepping procedure we're up to
switch( style ){
  case SINGLE:
    if ((currentstep/(MICROSTEPS/2)) % 2) {
      if (dir == FORWARD) {  currentstep += MICROSTEPS/2;   }
      else {                 currentstep -= MICROSTEPS/2;   }
    }
    else {          // go to the next even step
      if (dir == FORWARD) {  currentstep += MICROSTEPS;     }
      else {                 currentstep -= MICROSTEPS;     }
    }
    break;
  case DOUBLE:
```

```
  if (! (currentstep/(MICROSTEPS/2) % 2)) { // we're at an even step, weird
    if (dir == FORWARD) {  currentstep += MICROSTEPS/2;   }
    else {                 currentstep -= MICROSTEPS/2;   }
  }
  else {          // go to the next odd step
    if (dir == FORWARD) {  currentstep += MICROSTEPS;     }
    else {                 currentstep -= MICROSTEPS;     }
  }
  break;
case INTERLEAVE:
  if (dir == FORWARD) {    currentstep += MICROSTEPS/2;   }
  else {                   currentstep -= MICROSTEPS/2;   }
break;
case  MICROSTEP:
  if (dir == FORWARD) {    currentstep++;                 }
  else {                   currentstep--;                 }

  currentstep += MICROSTEPS*4;
  currentstep %= MICROSTEPS*4;

  ocra = ocrb = 0;
  if ( (currentstep >= 0) && (currentstep < MICROSTEPS)) {
    ocra = microstepcurve[MICROSTEPS - currentstep];
    ocrb = microstepcurve[currentstep];
  }
  else if  ( (currentstep >= MICROSTEPS) && (currentstep < MICROSTEPS*2)) {
```

```
      ocra = microstepcurve[currentstep - MICROSTEPS];
      ocrb = microstepcurve[MICROSTEPS*2 - currentstep];
    }
    else if  ( (currentstep >= MICROSTEPS*2) && (currentstep < MICROSTEPS*3)) {
      ocra = microstepcurve[MICROSTEPS*3 - currentstep];
      ocrb = microstepcurve[currentstep - MICROSTEPS*2];
    }
    else if  ( (currentstep >= MICROSTEPS*3) && (currentstep < MICROSTEPS*4)) {
      ocra = microstepcurve[currentstep - MICROSTEPS*3];
      ocrb = microstepcurve[MICROSTEPS*4 - currentstep];
    }
   break;
  }                  // switch( style )

  currentstep += MICROSTEPS*4;
  currentstep %= MICROSTEPS*4;

#ifdef MOTORDEBUG
  Serial.print("  current step: "); Serial.println(currentstep, DEC);
  Serial.print(", pwmA = ");       Serial.print(ocra, DEC);
  Serial.print(", pwmB = ");       Serial.println(ocrb, DEC);
#endif

  if (steppernum == 1) {
    setPWM1(ocra);
    setPWM2(ocrb);
```

```
  } else if (steppernum == 2) {
    setPWM3(ocra);
    setPWM4(ocrb);
  }


  // release all
  latch_state &= ~a & ~b & ~c & ~d; // all motor pins to 0
  if (style == MICROSTEP) {
    if ((currentstep >= 0)            && (currentstep < MICROSTEPS))   latch_state |= a | b;
    if ((currentstep >= MICROSTEPS)   && (currentstep < MICROSTEPS*2)) latch_state |= b | c;
    if ((currentstep >= MICROSTEPS*2) && (currentstep < MICROSTEPS*3)) latch_state |= c |
d;
    if ((currentstep >= MICROSTEPS*3) && (currentstep < MICROSTEPS*4)) latch_state |= d |
a;
  }
  else {
    switch (currentstep/(MICROSTEPS/2)) {
    case 0:  latch_state |= a;     /* energize coil 1 only */    break;
    case 1:  latch_state |= a | b; /* energize coil 1+2 */       break;
    case 2:  latch_state |= b;     /* energize coil 2 only */    break;
    case 3:  latch_state |= b | c; /* energize coil 2+3 */       break;
    case 4:  latch_state |= c;     /* energize coil 3 only */    break;
    case 5:  latch_state |= c | d; /* energize coil 3+4 */       break;
    case 6:  latch_state |= d;     /* energize coil 4 only */    break;
    case 7:  latch_state |= d | a; /* energize coil 1+4 */       break;
```

```
    }
  }

  MC.latch_tx();
  return currentstep;
}
```

Embedded C common to both 2560 and 328p

Snakebot_common.c

```
/*
 *
 *   Snakebot_common.c
 */
#include <WProgram.h>
#include <avr/io.h>       //This contains definitions for all the registers locations and some other things, must always be included
#include <util/delay.h>   //Contains some delay functions that will generate accurate delays of ms and us
#include <avr/interrupt.h>
#include <HardwareSerial.h>
#include <print.h>
#include <wire.h>
#include <ctype.h>                // for recognizing hexidecimal
#include <math.h>


#include <avr/pgmspace.h>         // for reading variables/strings from FLASH mem
#include <avr/eeprom.h>           // for reading/writing variables to/from EEPROM
#include <lightweightRingBuff.h>   // for TWI_ringBuf - ring buffer routines
```

```
#include <snakebot_common.h>

// ------------------------------------Global Variables -------------------------//
extern RingBuff_t TWI_ringBuf;
extern byte TWI_ID_g;

/*-------------------------------------------------------------------------
 *  extern "C" void __cxa_pure_virtual()
 *
 *  pure f-ing magic
 */
extern "C" void __cxa_pure_virtual()
{
 cli();
 for (;;);
}

/*-------------------------------------------------------------------------------------------------
 * ----------------------------------INTERRUPT SERVICE ROUTINES---------------------------------
---------
 *-------------------------------------------------------------------------------------------------*/

/*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 *  TIMER1_COMPB_vect
 *
```

```
 *    this ISR is called when Timer1 count equals TIMER1TRIGGERTIME.  Since Timer1
triggers the A/D ISR,
 *    this is the interval between AD sampling.  All the Timer1 ISR does is set Timer1 to zero
 */
ISR( TIMER1_COMPB_vect, ISR_BLOCK ){
  TCNT1 = 0;

#ifdef __AVR_ATmega2560__
//  extern uint32_t timer_g;
//  timer_g++;
#endif
};                                      // reset Timer1 and trigger the A/D
/*---------------------------------------------------------------------
 *  BADISR_vect
 *
 *   catchall ISR for ISR sources not addressed elsewhere.
 *
 *   The Serial.print(...) is hopeful thinking:  likely if interrupts are
 *   askew the USB interrupt won't work.
 *   Ergo strobing LEDs, requiring no interrupt
 *
 */
ISR(BADISR_vect)
{
#ifdef __AVR_ATmega2560__
```

```
void setupOverforceLEDs(  void );      // prototypes for fcns LEDtest2560.c
void setupUpperLimitLEDs( void );
void setupLowerLimitLEDs( void );

  setupOverforceLEDs();
  setupUpperLimitLEDs();
  setupLowerLimitLEDs();
  setupUpperLimitLEDs();
  setupOverforceLEDs();

#elif defined __AVR_ATmega328P__

  PORTB |=  ( 1 << PINB2 );            // turn pin 10 LED ON
  _delay_ms( 500 );
  PORTB &= ~( 1 << PINB2 );            // turn pin 10 LED OFF

  PORTB |=  ( 1 << PINB5 );            // turn pin 13 LED ON
  _delay_ms( 500 );
  PORTB &= ~( 1 << PINB5 );            // turn pin 13 LED OFF

#endif

  Serial.print_P(PSTR("\r\n BAD Interrupt occurred...."));
}
/*---------------------------------------------------------------------
 *  TWI_RcvEvent
```

```
 *
 *    TWIrcvBufcount must be decremented when TWI_rcvBuf is read from
 */
void TWI_RcvEvent( int howMany ){

  while(( Wire.available() > 0 )&&
      (!( RingBuffer_IsFull( &TWI_ringBuf )))){

    byte temp = Wire.receive();
    RingBuffer_Insert( &TWI_ringBuf, temp );
  }
}
/*---------------------------------------------------------------------
 *  TWI_placeInBuffer
 */
void TWI_placeInBuffer( uint8_t * ptr, uint8_t howMany ){

  cli();
                        // pre-pend a msg-separator
  if(!( RingBuffer_IsFull( &TWI_ringBuf ))){
    RingBuffer_Insert( &TWI_ringBuf, __SEPARATOR__ );
  }

  while(( howMany-- )&&
      (!( RingBuffer_IsFull( &TWI_ringBuf )))){
```

```
    RingBuffer_Insert( &TWI_ringBuf, *ptr );
            ptr++;
  }
  sei();
}
/*------------------------------------------------------------------------
 *  TWI_SendMessage
 *
 *    if the 2560's TWI pins are connected to a TWI client that is powered
 *    off, the 2560 will lock up on Wire.endTransmission()
 */
byte  TWI_SendMessage( byte * ptr, byte i2c_dest, byte msgLen ){
  Wire.beginTransmission( i2c_dest );
  byte separator = __SEPARATOR__;
  Wire.send( &separator, 1 );              // pre-pend a msg-separator
  Wire.send( ptr, msgLen );
  return( Wire.endTransmission());
}
/*------------------------------------------------------------------------
 *  isSeparator
 */
inline boolean isSeparator( byte c ){

  return ( c == __SEPARATOR__ );
}
/*------------------------------------------------------------------------
```

```
 *  readTWImsg
 */
byte readTWImsg( byte * buf ){

 byte i=0;

 while(( isWhitespace( RingBuffer_peek( &TWI_ringBuf )))||
      ( isSeparator(  RingBuffer_peek( &TWI_ringBuf )))){

  RingBuffer_Remove( &TWI_ringBuf ); // trash white and separator
 }
 do{
  buf[i] = RingBuffer_Remove( &TWI_ringBuf );

 }while((!( isWhitespace( buf[i] )))&&
       (( ++i ) < BUFFSIZE )&&
       ( RingBuffer_GetCount( &TWI_ringBuf ))&&
                  (!( isSeparator( RingBuffer_peek( &TWI_ringBuf )))));

 buf[i++]=0;
 return i;
}

/*---------------------------------------------------------------------
 *  setupTWI
 */
```

```
void setupTWI(  ){

  RingBuffer_InitBuffer( &TWI_ringBuf );
  Wire.begin( TWI_ID_g );
  Wire.onReceive( TWI_RcvEvent );
}
/*-----------------------------------------------------------------------
 *  readUSBmsg
 */
byte readUSBmsg( byte * buf ){

  byte i=0;

  while( isWhitespace( Serial.peek( ))){ Serial.read( ); }
  do{
    buf[i] = Serial.read();

  } while(( !( isWhitespace( buf[i] )))&&
            (( ++i ) < BUFFSIZE ) &&
            ( Serial.available( )));
  buf[i++]=0;
  return i;
}
/*-----------------------------------------------------------------------------------------------------
 *  setupTimer1
 *
```

```
        *    Write to the registers that control Timer1 such that Timer1 generates interrupts at intervals
        *    equal to count TIMER1TRIGGERTIME.  Timer1's interrupts are used to cause the AD to
read.
        */
        int setupTimer1( void ) {
         TCNT1 = 0;                          // Timer1 reset to 0
         TCCR1A = 0;                          // COM1Ax = 0 -> normal mode
         TCCR1B = (1<<CS10) | (1<<CS12);              // CSxx=0x05 -> clock = 16MHz/1024
                                        // ~4.1sec b/t overflow interrupts
                                        // WGMxx = 0b000 -> mode 0 -> upcounter
                                        // WGMxx = 0b0100 -> mode 4 -> Clear Timer on Compare
(CTC)
        //TCCR1B |= (1<<WGM12 );                    // to OCR1A.  Note in this mode, overflow int
never trigger
         TCCR1C = 0;                         // not using 'force output compare' to generate waveforms
         OCR1B = 0X4ff;                          // this value is compared to current Timer1 value
                                        // increasing it will increase time between compareB interrupts
         TIMSK1 = (1<<OCIE1B);                    // timer 1 interrupt enable on compare B
                                        // must use T1CompareB...only T1CompareB, T1Overflow
triggers AD
         return TRUE;
        }
        /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
         *  setupAD
         *
         *    Write to the registers that control the Analog-to-Digital circuitry
```

```
        *    There is one AD with a 6-input multiplexer (mux).  The AD is interrupt driven by
        *    Timer1 interrupt
        *
        *    This was written for the m328p cpu.  Will it work on the 2560?  yes, it works for both.
        */
       int setupAD( void ){
        DIDR0   =    0;                          // enable all ADC0..ADC5 dis-able-able inputs
        ADMUX   =   (1 << REFS0 );                // Voltage Reference == AVcc
        ADMUX  |=   (1 << ADLAR );                 // left adjust.  Will read only ADCH, ignore
2LSBits
        ADMUX  |=   (1 << MUX0 );                 // begin AD conversions reading ADC1 (force
sensor 1)
                                                  // Will cycle through others Selected in ISR

// note MUX5 in ADCSRB = 0 -> Port F

//           ADCSRB = 1 -> Port K not used
        ADCSRB  = (( 1 << ADTS2 ) | (1 << ADTS0 ));
        ADCSRB &= ~ (1 << ADTS1 );                 // ADCSRB = b101 -> mode 5 -> Timer1
Compare B triggers A/D
        ADCSRB &= ~ (1 << ACME );                  // disable the A/D comparator MUX bit (p.247)

        ADCSRA  =   (1 << ADATE );                // ADC Auto Trigger Enable, per ADCSRB.ADTS
bits (above)
        ADCSRA |=   (1 << ADPS2 );                // ADC Clock Prescaler mode = b100 -> mode4 ->
div by 32
```

```
  ADCSRA |=   (1 << ADEN );                  // enable A/D, clearing all other control bits -
EXEC 2nd LAST
  ADCSRA |=   (1 << ADIE );                  // AD Interrupt Enable - MUST EXECUTE LAST

  return TRUE;
 }

 /*-------------------------------------------------------------------------------------------------
  *----------------------------------------------FUNCTIONS----------------------------------------
  *-------------------------------------------------------------------------------------------------*/

 /*-----------------------------------------------------------------
  *  heartbeatDots
  *
  *  input  pointer to counter to limit number dots/line to 10
  *  output none
  */
 void heartbeatDots( void ){
  static byte beatCount = 0, delay=0;
  if( ++delay < 10 ) return; delay=0;
  if( beatCount & 1 )  Serial.print_P(PSTR("."));          // heartbeat indication
  else                 Serial.print_P(PSTR("*"));          // heartbeat indication
  if( ++ beatCount >= 100 ) {
    beatCount = 0;
    Serial.print_P(PSTR("\r\n"));
  }
```

```
}

/*---------------------------------------------------------------------
 *  TWI_ID_From_AxisNum
 */
byte TWI_ID_From_AxisNum( byte axisNum ){
  switch( axisNum ){
      case  0:
      case  1:  return( I2C_SLAVE_10 );
      case  2:
      case  3:  return( I2C_SLAVE_32 );
      case  4:
      case  5:  return( I2C_SLAVE_54 );
      case  6:
      case  7:  return( I2C_SLAVE_76 );
      case  8:  return( I2C_SLAVE_88 );
      default:  return( I2C_FAILURE );
  }
}

/*---------------------------------------------------------------------
 *  convertHexTo2bAsciiHex
 */
byte convertHexTo2bAsciiHex( byte HexNum, byte * asciiHexArray ){
  if( asciiHexArray == NULL ) return FALSE;
```

```
  byte MSN = ( HexNum & 0xF0 ) >> 4;
  byte LSN = ( HexNum & 0x0F );

  if(( MSN > 0x0F )||
     ( LSN > 0x0F )) return FALSE;

  if( MSN <= 9 ) asciiHexArray[0]  =  ( MSN + '0');     // decimal 0->9
  else           asciiHexArray[0]  =  ( MSN + 55 ) ;    // A->F hex

  if( LSN <= 9)  asciiHexArray[1]  =  ( LSN + '0');
  else           asciiHexArray[1]  =  ( LSN + 55 );

  return TRUE;
}

/*---------------------------------------------------------------------
 * convert2bAsciiHexToHex
 */
byte convert2bAsciiHexToHex( byte * buf ){

  byte retval;

  retval  = convert1bAsciiHexToHex( buf[0] ) << 4;  // Most Significant Nibble
  retval |= convert1bAsciiHexToHex( buf[1] );       // LSN

  return retval;
```

```
}
/*---------------------------------------------------------------------
 * convert3bAsciiHexToHex
 */
uint16_t convert3bAsciiHexToHex( byte * buf ){

  uint16_t retval;

  retval  = convert1bAsciiHexToHex( buf[0] );       // Most Significant Nibble
  retval <<= 4;                                     // left shift 4 bits
  retval |= convert1bAsciiHexToHex( buf[1] );
  retval <<= 4;
  retval |= convert1bAsciiHexToHex( buf[2] );

  return retval;
}
/*---------------------------------------------------------------------
 * convert4bAsciiHexToHex
 */
uint16_t convert4bAsciiHexToHex( byte * buf ){

  uint16_t retval = convert3bAsciiHexToHex( buf );

  retval <<= 4;
  retval |= convert1bAsciiHexToHex( buf[3] );       // LSN
```

```
  return retval;
}

/*---------------------------------------------------------------------
 *  TWIidIsValid
 */
byte TWIidIsValid( byte TWIid ){

  switch( TWIid ){
    case I2C_SLAVE_10:
    case I2C_SLAVE_32:
    case I2C_SLAVE_54:
    case I2C_SLAVE_76:
          case I2C_SLAVE_88: return TRUE;
    default:          return FALSE;
  }
}

/*---------------------------------------------------------------------
 *  echoToUSB
 */
void echoToUSB( const char * descr, byte * buf, byte numBytes, uint16_t lineNum ){

  Serial.print_P(PSTR("\r\n<"));       Serial.print( lineNum,  DEC );
  Serial.print_P(PSTR(">"));           Serial.write( descr );
  Serial.print_P(PSTR(", TWI_id: "));  Serial.print( TWI_ID_g, DEC );
```

```
  Serial.print_P(PSTR(",  echo [ "));   Serial.print( numBytes, DEC );
  Serial.print_P(PSTR(" ]= " ));

  if( buf == NULL ) return;

  for( byte i=0; i<numBytes; i++){
    if( isascii( buf[i] )) {
            Serial.write( &buf[i], 1 );
    }
           else{
            Serial.print( buf[i], HEX );
/*          byte temp[2];
            convertHexTo2bAsciiHex( buf[i], temp );
            Serial.write( temp, 1 );
*/         }
  }
}
/*---------------------------------------------------------------------
 *  print2Bhex
 */
void print2Bhex( const char * descr, byte hex2b ){
  byte ascii2b[2];
  convertHexTo2bAsciiHex( hex2b, ascii2b );
  echoToUSB( descr, ascii2b, 2, __LINE__ );
}
/*---------------------------------------------------------------------
```

```
 *  printMoveParms
 */
byte printMoveParms( moveParms_t * moveParms_p, uint16_t lineNum ){

  if( moveParms_p == NULL ) return FALSE;
  Serial.print_P(PSTR("\r\n moveParms->axisNum: "));
  Serial.print( moveParms_p->axisNum, DEC );
  Serial.print_P(PSTR("\r\n moveParms->TWI_dest: "));
  Serial.print( moveParms_p->TWI_dest, DEC );
  Serial.print_P(PSTR("\r\n moveParms->axisDest: "));
  Serial.print( moveParms_p->axisDest, DEC );
  return TRUE;
}
/*---------------------------------------------------------------------
 *  parseMoveParms
 *       buf[0]     = 'M' or 'J'
 *       buf[1]     = '1' == axis #1 == [0->7] in ascii
 *       buf[2]     = ','
 *          [3,4,5] = ascii decimal axis destination value
 */
byte parseMoveParms( moveParms_t * moveParms_p, byte * buf ){

  if(( buf[0] != 'M' )&&( buf[0] != 'J' )) return FALSE;

  moveParms_p->axisNum  = convert1bAsciiHexToHex( buf[1] );
  if( moveParms_p->axisNum >= NUMAXES )     return FALSE;
```

```
  moveParms_p->TWI_dest = TWI_ID_From_AxisNum( moveParms_p->axisNum );
  if( moveParms_p->TWI_dest == I2C_FAILURE) return FALSE;

  moveParms_p->axisDest = atoi((char*) &buf[3] );
  if(( moveParms_p->axisDest == RELEASE_AXES )||
     ( moveParms_p->axisDest == INIT_AXES )||
     ( moveParms_p->axisDest == ZERO_AXES )||
    (( moveParms_p->axisDest <= MAX_AXIS_COUNT_LIM )&&
     ( moveParms_p->axisDest >= MAX_AXIS_COUNT_LIM*(-1))))return TRUE;

  return FALSE;
}

/*---------------------------------------------------------------------
 *  createMoveCmd
 *
 *     args:  buf[] an array of byte ascii args from the USB or TWI
 *       ex:  J1P1S1
 *       buf[0]   = 'J' or 'M'
 *       buf[1]   = '1' == axis #1
 *       buf[2]   = ','
 *       buf[3,...] = destination axis count.  decimal-ascii
 */
byte createMoveCmd( char     * JorM,           // 'J' or 'M'
                    char     * buf,            // dest
```

```
                byte       axisNum,         // ascii
                int16_t    destAxisPos  ){  // hex
  memset( buf, 0, BUFFSIZE );
  buf[0] = *JorM;
  buf[1] = axisNum;
  buf[2]=',';
  itoa( destAxisPos, (char*)&buf[3], 10 );
  buf[ strlen( buf )] = ',';
  buf[ strlen( buf )] = SPACE;

  return( strlen( buf ));
}
/*---------------------------------------------------------------------
 * mangle
 */
static inline byte mangle( int16_t num, char * buf, byte  i ){

  itoa( num, &buf[i], 10 );
  buf[ strlen( buf )] = ',';
  return  strlen( buf );
}
/*---------------------------------------------------------------------
 * convertAxisStatusToAscii
 */
byte convertAxisStatusToAscii( axisStatus_t * axisStatus_p, char * buf ){
```

```
  byte i = 0;
  memset( buf, 0, BUFFSIZE-2 );
  i = mangle(( int16_t ) axisStatus_p->axisNum,   buf, i );
  i = mangle(( int16_t ) axisStatus_p->axisPos,   buf, i );
  i = mangle(( int16_t ) axisStatus_p->prevPos,   buf, i );
  i = mangle(( int16_t ) axisStatus_p->topLimit,  buf, i );
  i = mangle(( int16_t ) axisStatus_p->lowLimit,  buf, i );
  i = mangle(( int16_t ) axisStatus_p->moveCount, buf, i );
  buf[i] = SPACE;                 // append ' '
  return i;
}
/*---------------------------------------------------------------------
 *  convertAsciiAxisStatusToDec
 */
byte convertAsciiAxisStatusToDec( char * buf, axisStatus_t * axisStatus_p ){

  axisStatus_p->axisNum   = strtol( buf, &buf, 10 );  buf++;
  axisStatus_p->axisPos   = strtol( buf, &buf, 10 );  buf++;
  axisStatus_p->prevPos   = strtol( buf, &buf, 10 );  buf++;
  axisStatus_p->topLimit  = strtol( buf, &buf, 10 );  buf++;
  axisStatus_p->lowLimit  = strtol( buf, &buf, 10 );  buf++;
  axisStatus_p->moveCount = strtol( buf, &buf, 10 );
  return TRUE;
}
/*---------------------------------------------------------------------
 *  printAxisStatus_t
```

```
 */
byte printAxisStatus_t( axisStatus_t * axisStatus_p ){

  if( axisStatus_p == NULL ) return FALSE;
  Serial.print_P(PSTR("\r\n axisNum=" ));  Serial.print((int) axisStatus_p->axisNum,  DEC );
  Serial.print_P(PSTR("\r\n axisPos=" ));  Serial.print((int) axisStatus_p->axisPos,  DEC );
  Serial.print_P(PSTR("\r\n prevPos=" ));  Serial.print((int) axisStatus_p->prevPos,  DEC );
  Serial.print_P(PSTR("\r\n topLimit=" )); Serial.print((int) axisStatus_p->topLimit, DEC );
  Serial.print_P(PSTR("\r\n lowLimit=" )); Serial.print((int) axisStatus_p->lowLimit, DEC );
  Serial.print_P(PSTR("\r\n moveCount=" ));Serial.print((int) axisStatus_p->moveCount,DEC );
  return TRUE;
}
/*---------------------------------------------------------------------
 *
 *       Routines for Maintenance of EEPROM Linked List of Axes Config
 */
 uint8_t *heapptr, *stackptr;
uint16_t diff1, diff2=0;
int get_free_memory()
{
  int free_memory;
extern unsigned int __data_start;
extern unsigned int __data_end;
extern unsigned int __bss_start;
extern unsigned int __bss_end;
extern unsigned int __heap_start;
```

```
//extern void *__malloc_heap_start; --> already declared as char*
//extern void *__malloc_margin;     --> already declared as a size_t
extern void *__brkval;

  if((int)__brkval == 0)
    free_memory = ((int)&free_memory) - ((int)&__bss_end);
  else
    free_memory = ((int)&free_memory) - ((int)__brkval);

  stackptr = (uint8_t *)malloc(4);          // use stackptr temporarily
  heapptr = stackptr;                     // save value of heap pointer
//free(stackptr);      // free up the memory again (sets stackptr to 0)
  stackptr =  (uint8_t *)(SP);           // save value of stack pointer


  Serial.print_P(PSTR("\r\n-->  get_free_memory: "));
  Serial.print( free_memory, DEC );

  Serial.print_P(PSTR(", heapptr= "));     Serial.print((int) heapptr, HEX );
  Serial.print_P(PSTR(", stackptr= "));    Serial.print((int) stackptr, HEX );


  Serial.print_P(PSTR(", RAMEND= "));      Serial.print( RAMEND, HEX );
  Serial.print_P(PSTR(", SP= "));          Serial.print( SP, HEX );
  Serial.print_P(PSTR(", __bss_start= ")); Serial.print( (int)__bss_start, HEX );
  Serial.print_P(PSTR(", __bss_end= "));   Serial.print( (int)__bss_end, HEX );
```

```
  Serial.print_P(PSTR(", __data_start= "));Serial.print( (int)__data_start, HEX );
  Serial.print_P(PSTR(", __data_end= "));  Serial.print( (int)__data_end, HEX );
  Serial.print_P(PSTR(", __heap_start= "));Serial.print( (int)__heap_start, HEX );
  Serial.print_P(PSTR(", __brkval= "));    Serial.print( (int)__brkval, HEX );
  Serial.print_P(PSTR(", __malloc_heap_start= "));Serial.print( (int)__malloc_heap_start, HEX );
  Serial.print_P(PSTR(", __malloc_margin= "));Serial.print( (int)__malloc_margin, HEX );

  return free_memory;
}

/*---------------------------------------------------------------------
 *  setupVernum
 */
void setupVernum( byte vernum ){

  Serial.print_P(PSTR("\r\n  Software Version:  "));
  Serial.print( vernum, DEC );
  Serial.print_P(PSTR("\r\n"));
}
```

Snakebot_common.h

```
/*
 *
 *   Snakebot_common.h
 */

#ifndef __SNAKEBOT_COMMON_H__
#define __SNAKEBOT_COMMON_H__

#include <WProgram.h>          // necessary for definition of 'uint8_t' type
#include <inttypes.h>

#ifndef delay
#define delay( msec ) _delay_ms( msec )
#endif

#ifndef FALSE
#define FALSE false
#endif

#ifndef TRUE
#define TRUE true
#endif

#define NULL_TERMINATOR 32
#define SPACE ' '
```

```
#define __SEPARATOR__ '_'
                        /* there are 9 axes, numbered 0->8  */
#ifndef NUMAXES
#define NUMAXES 9
#endif

#define BUFFSIZE 32             /* buffer size in bytes for both USB and TWI comms */
#define MAX_STEP_SIZE  (30)

#define  I2C_MASTER_00  0       /* TWI ID number for Master UNO  */
#define  I2C_SLAVE_10  10       /* TWI ID number Slave UNO w/ steppers 1&2  */
#define  I2C_SLAVE_32  32
#define  I2C_SLAVE_54  54
#define  I2C_SLAVE_76  76
#define  I2C_SLAVE_88  88       /* only one axis (8): there is no (9) */
#define  I2C_FAILURE  255

#define MAX_AXIS_COUNT_LIM 11111 /* no more limits...never reach this */
#define AXISLIMIT MAX_AXIS_COUNT_LIM
#define RELEASE_AXES 32767       /* max 16-bit signed int */
#define INIT_AXES (RELEASE_AXES-1)
#define ZERO_AXES (INIT_AXES-1)

typedef struct axisStatus_t{

  uint8_t axisNum;
```

```
  int16_t axisPos;
  int16_t prevPos;
  int16_t topLimit;
  int16_t lowLimit;
  uint8_t moveCount;

} axisStatus_t;

typedef struct moveParms_t{
  byte axisNum;
  byte TWI_dest;
  int16_t axisDest;
} moveParms_t;

// possible results returned from Wire.endTransmission()
#define I2C_XMT_SUCCESS 0
#define I2C_XMT_DATA_TOO_LONG 1
#define I2C_XMT_NACK_ADDRESS 2
#define I2C_XMT_NACK_DATA 3
#define I2C_XMT_OTHER_ERR 4

//ISR( TIMER1_COMPB_vect, ISR_BLOCK ){ TCNT1 = 0; };            // reset Timer1 and
trigger the A/D
void TWI_RcvEvent( int howMany );
byte  TWI_SendMessage( byte * ptr, byte i2c_dest, byte msgLen );
void TWI_placeInBuffer( uint8_t * ptr, uint8_t howMany );
```

```
//byte isWhitespace( byte c );
byte readTWImsg( byte * buf );
void setupTWI( void );
byte readUSBmsg( byte * buf );
int setupTimer1( void ) ;
int setupAD( void );
void heartbeatDots( void );
byte TWI_ID_From_AxisNum( byte axisNum );
//byte verifyTWI_destAndAxisMatch( byte TWI_id, byte axisNum );
byte convertHexTo2bAsciiHex( byte HexNum, byte * asciiHexArray );
//byte convert1bAsciiHexToHex( byte retval );
byte convert2bAsciiHexToHex( byte * buf );
uint16_t convert3bAsciiHexToHex( byte * buf );
uint16_t convert4bAsciiHexToHex( byte * buf );
byte TWIidIsValid( byte TWIid );
byte assignAxes( void );
void echoToUSB( const char * descr, byte * buf, byte numBytes, uint16_t lineNum );
void print2Bhex( const char * descr, byte hex2b );
byte parseMoveParms( moveParms_t * moveParms, byte * buf );
byte printMoveParms( moveParms_t * moveParms_p, uint16_t lineNum );
byte createMoveCmd( char * JorM,char * buf, byte axisNum, int16_t destAxisPos  );
byte printAxisStatus_t( axisStatus_t * axisStatus_p );
byte convertAxisStatusToAscii( axisStatus_t * axisStatus_p, char * buf );
byte convertAsciiAxisStatusToDec( char * buf, axisStatus_t * axisStatus_p );

/*  Inline functions take no compile-space, but are inserted into
```

```
   the code where 'called'
 */
/*---------------------------------------------------------------------
 * convert1bAsciiHexToHex
 */
inline byte convert1bAsciiHexToHex( byte retval ){

  if( !isxdigit( retval )) return 0;

  if( isdigit( retval )) retval -= '0';    // decimal 0->9
  else                   retval -=  55;    // A->F hex

  return retval;
}

/*---------------------------------------------------------------------
 *  isWhitespace
 *
 *     iscntrl( c ) return TRUE if any of 1st 32 nonprinting ascii char
 *     isspace( c ) return TRUE if space, \f ,\n, \r, \t, \v
 *
 */
inline byte isWhitespace( byte c ){

  return(( iscntrl( c ))||
         ( isspace( c )));
```

```
}
/*--------------------------------------------------------------------
 *   verifyTWI_destAndAxisMatch( )
 */
inline byte verifyTWI_destAndAxisMatch( byte TWI_id, byte axisNum ){
  return( TWI_id == TWI_ID_From_AxisNum( axisNum ));
}
#ifndef _blurt_
#define _blurt_
/*--------------------------------------------------------------------
 *  blurt
 */
static void inline blurt( const char * descr, byte data ){

    Serial.write( descr );
    Serial.print( data, DEC );
}
#endif
#endif
```

Software from Arduino Library

HardwareSerial.cpp

```
/*
  HardwareSerial.cpp - Hardware serial library for Wiring
  Copyright (c) 2006 Nicholas Zambetti.  All right reserved.

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA

  Modified 23 November 2006 by David A. Mellis
  Modified 28 September 2010 by Mark Sproul
*/
```

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <inttypes.h>
#include "wiring.h"
#include "wiring_private.h"

// this next line disables the entire HardwareSerial.cpp,
// this is so I can support Attiny series and any other chip without a uart
#if defined(UBRRH) || defined(UBRR0H) || defined(UBRR1H) || defined(UBRR2H) ||
defined(UBRR3H)

#include "HardwareSerial.h"

// Define constants and variables for buffering incoming serial data.  We're
// using a ring buffer (I think), in which rx_buffer_head is the index of the
// location to which to write the next incoming character and rx_buffer_tail
// is the index of the location from which to read.
#if (RAMEND < 1000)
  #define RX_BUFFER_SIZE 32
#else
  #define RX_BUFFER_SIZE 128
#endif

struct ring_buffer
{
```

```
  unsigned char buffer[RX_BUFFER_SIZE];
  int head;
  int tail;
};

#if defined(UBRRH) || defined(UBRR0H)
  ring_buffer rx_buffer  =  { { 0 }, 0, 0 };
#endif
#if defined(UBRR1H)
  ring_buffer rx_buffer1  =  { { 0 }, 0, 0 };
#endif
#if defined(UBRR2H)
  ring_buffer rx_buffer2  =  { { 0 }, 0, 0 };
#endif
#if defined(UBRR3H)
  ring_buffer rx_buffer3  =  { { 0 }, 0, 0 };
#endif

inline void store_char(unsigned char c, ring_buffer *rx_buffer)
{
  int i = (unsigned int)(rx_buffer->head + 1) % RX_BUFFER_SIZE;

  // if we should be storing the received character into the location
  // just before the tail (meaning that the head would advance to the
  // current location of the tail), we're about to overflow the buffer
  // and so we don't write the character or advance the head.
```

```
  if (i != rx_buffer->tail) {
    rx_buffer->buffer[rx_buffer->head] = c;
    rx_buffer->head = i;
  }
}

#if defined(USART_RX_vect)
  SIGNAL(USART_RX_vect)
  {
  #if defined(UDR0)
    unsigned char c  =  UDR0;
  #elif defined(UDR)
    unsigned char c  =  UDR;  //  atmega8535
  #else
    #error UDR not defined
  #endif
    store_char(c, &rx_buffer);
  }
#elif defined(SIG_USART0_RECV) && defined(UDR0)
  SIGNAL(SIG_USART0_RECV)
  {
    unsigned char c  =  UDR0;
    store_char(c, &rx_buffer);
  }
#elif defined(SIG_UART0_RECV) && defined(UDR0)
  SIGNAL(SIG_UART0_RECV)
```

```
  {
    unsigned char c  =  UDR0;
    store_char(c, &rx_buffer);
  }
//#elif defined(SIG_USART_RECV)
#elif defined(USART0_RX_vect)
  // fixed by Mark Sproul this is on the 644/644p
  //SIGNAL(SIG_USART_RECV)
  SIGNAL(USART0_RX_vect)
  {
  #if defined(UDR0)
    unsigned char c  =  UDR0;
  #elif defined(UDR)
    unsigned char c  =  UDR;  //  atmega8, atmega32
  #else
    #error UDR not defined
  #endif
    store_char(c, &rx_buffer);
  }
#elif defined(SIG_UART_RECV)
  // this is for atmega8
  SIGNAL(SIG_UART_RECV)
  {
  #if defined(UDR0)
    unsigned char c  =  UDR0;  //  atmega645
  #elif defined(UDR)
```

```
    unsigned char c  =  UDR;  //  atmega8
  #endif
    store_char(c, &rx_buffer);
  }
#elif defined(USBCON)
  #warning No interrupt handler for usart 0
  #warning Serial(0) is on USB interface
#else
  #error No interrupt handler for usart 0
#endif

//#if defined(SIG_USART1_RECV)
#if defined(USART1_RX_vect)
  //SIGNAL(SIG_USART1_RECV)
  SIGNAL(USART1_RX_vect)
  {
    unsigned char c = UDR1;
    store_char(c, &rx_buffer1);
  }
#elif defined(SIG_USART1_RECV)
  #error SIG_USART1_RECV
#endif

#if defined(USART2_RX_vect) && defined(UDR2)
  SIGNAL(USART2_RX_vect)
  {
```

```
    unsigned char c = UDR2;
    store_char(c, &rx_buffer2);
  }
#elif defined(SIG_USART2_RECV)
  #error SIG_USART2_RECV
#endif

#if defined(USART3_RX_vect) && defined(UDR3)
  SIGNAL(USART3_RX_vect)
  {
    unsigned char c = UDR3;
    store_char(c, &rx_buffer3);
  }
#elif defined(SIG_USART3_RECV)
  #error SIG_USART3_RECV
#endif



// Constructors ////////////////////////////////////////////////////////////////

HardwareSerial::HardwareSerial(ring_buffer *rx_buffer,
  volatile uint8_t *ubrrh, volatile uint8_t *ubrrl,
  volatile uint8_t *ucsra, volatile uint8_t *ucsrb,
  volatile uint8_t *udr,
  uint8_t rxen, uint8_t txen, uint8_t rxcie, uint8_t udre, uint8_t u2x)
```

```
{
  _rx_buffer = rx_buffer;
  _ubrrh = ubrrh;
  _ubrrl = ubrrl;
  _ucsra = ucsra;
  _ucsrb = ucsrb;
  _udr = udr;
  _rxen = rxen;
  _txen = txen;
  _rxcie = rxcie;
  _udre = udre;
  _u2x = u2x;
}

// Public Methods //////////////////////////////////////////////////////////////

void HardwareSerial::begin(long baud)
{
  uint16_t baud_setting;
  bool use_u2x = true;

#if F_CPU == 16000000UL
  // hardcoded exception for compatibility with the bootloader shipped
  // with the Duemilanove and previous boards and the firmware on the 8U2
  // on the Uno and Mega 2560.
  if (baud == 57600) {
```

```
    use_u2x = false;
  }
#endif

  if (use_u2x) {
    *_ucsra = 1 << _u2x;
    baud_setting = (F_CPU / 4 / baud - 1) / 2;
  } else {
    *_ucsra = 0;
    baud_setting = (F_CPU / 8 / baud - 1) / 2;
  }

  // assign the baud_setting, a.k.a. ubbr (USART Baud Rate Register)
  *_ubrrh = baud_setting >> 8;
  *_ubrrl = baud_setting;

  sbi(*_ucsrb, _rxen);
  sbi(*_ucsrb, _txen);
  sbi(*_ucsrb, _rxcie);
}

void HardwareSerial::end()
{
  cbi(*_ucsrb, _rxen);
  cbi(*_ucsrb, _txen);
  cbi(*_ucsrb, _rxcie);
```

```
}

int HardwareSerial::available(void)
{
  return (unsigned int)(RX_BUFFER_SIZE + _rx_buffer->head - _rx_buffer->tail) %
RX_BUFFER_SIZE;
}

int HardwareSerial::peek(void)
{
  if (_rx_buffer->head == _rx_buffer->tail) {
    return -1;
  } else {
    return _rx_buffer->buffer[_rx_buffer->tail];
  }
}

int HardwareSerial::read(void)
{
  // if the head isn't ahead of the tail, we don't have any characters
  if (_rx_buffer->head == _rx_buffer->tail) {
    return -1;
  } else {
    unsigned char c = _rx_buffer->buffer[_rx_buffer->tail];
    _rx_buffer->tail = (unsigned int)(_rx_buffer->tail + 1) % RX_BUFFER_SIZE;
    return c;
```

```
  }
}

void HardwareSerial::flush()
{
  // don't reverse this or there may be problems if the RX interrupt
  // occurs after reading the value of rx_buffer_head but before writing
  // the value to rx_buffer_tail; the previous value of rx_buffer_head
  // may be written to rx_buffer_tail, making it appear as if the buffer
  // don't reverse this or there may be problems if the RX interrupt
  // occurs after reading the value of rx_buffer_head but before writing
  // the value to rx_buffer_tail; the previous value of rx_buffer_head
  // may be written to rx_buffer_tail, making it appear as if the buffer
  // were full, not empty.
  _rx_buffer->head = _rx_buffer->tail;
}

void HardwareSerial::write(uint8_t c)
{
  while (!((*_ucsra) & (1 << _udre)))
    ;

  *_udr = c;
}

void HardwareSerial::print_P( const char * ptr )
```

```
{
  while(pgm_read_byte(ptr) != 0x00) print(pgm_read_byte(ptr++));
}



// Preinstantiate Objects //////////////////////////////////////////////////

#if defined(UBRRH) && defined(UBRRL)
  HardwareSerial Serial(&rx_buffer, &UBRRH, &UBRRL, &UCSRA, &UCSRB, &UDR,
RXEN, TXEN, RXCIE, UDRE, U2X);
#elif defined(UBRR0H) && defined(UBRR0L)
  HardwareSerial Serial(&rx_buffer, &UBRR0H, &UBRR0L, &UCSR0A, &UCSR0B, &UDR0,
RXEN0, TXEN0, RXCIE0, UDRE0, U2X0);
#elif defined(USBCON)
  #warning no serial port defined  (port 0)
#else
  #error no serial port defined  (port 0)
#endif

#if defined(UBRR1H)
  HardwareSerial Serial1(&rx_buffer1, &UBRR1H, &UBRR1L, &UCSR1A, &UCSR1B,
&UDR1, RXEN1, TXEN1, RXCIE1, UDRE1, U2X1);
#endif
#if defined(UBRR2H)
```

```
  HardwareSerial Serial2(&rx_buffer2, &UBRR2H, &UBRR2L, &UCSR2A, &UCSR2B,
&UDR2, RXEN2, TXEN2, RXCIE2, UDRE2, U2X2);
#endif
#if defined(UBRR3H)
  HardwareSerial Serial3(&rx_buffer3, &UBRR3H, &UBRR3L, &UCSR3A, &UCSR3B,
&UDR3, RXEN3, TXEN3, RXCIE3, UDRE3, U2X3);
#endif

#endif // whole file
```

Twi.c

```
/*
  twi.c - TWI/I2C library for Wiring & Arduino
  Copyright (c) 2006 Nicholas Zambetti.  All right reserved.

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
*/

#include <math.h>
#include <stdlib.h>
#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <compat/twi.h>
```

```
#ifndef cbi
#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#endif

#ifndef sbi
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
#endif

#include "twi.h"

static volatile uint8_t twi_state;
static uint8_t twi_slarw;

static void (*twi_onSlaveTransmit)(void);
static void (*twi_onSlaveReceive)(uint8_t*, int);

static uint8_t twi_masterBuffer[TWI_BUFFER_LENGTH];
static volatile uint8_t twi_masterBufferIndex;
static uint8_t twi_masterBufferLength;

static uint8_t twi_txBuffer[TWI_BUFFER_LENGTH];
static volatile uint8_t twi_txBufferIndex;
static volatile uint8_t twi_txBufferLength;

static uint8_t twi_rxBuffer[TWI_BUFFER_LENGTH];
```

```
static volatile uint8_t twi_rxBufferIndex;

static volatile uint8_t twi_error;

/*
 * Function twi_init
 * Desc     readys twi pins and sets twi bitrate
 * Input    none
 * Output   none
 */
void twi_init(void)
{
  // initialize state
  twi_state = TWI_READY;

  #if defined(__AVR_ATmega168__) || defined(__AVR_ATmega8__) ||
defined(__AVR_ATmega328P__)
    // activate internal pull-ups for twi
    // as per note from atmega8 manual pg167
    sbi(PORTC, 4);
    sbi(PORTC, 5);
  #else
    // activate internal pull-ups for twi
    // as per note from atmega128 manual pg204
    sbi(PORTD, 0);
    sbi(PORTD, 1);
```

```
  #endif

  // initialize twi prescaler and bit rate
  cbi(TWSR, TWPS0);
  cbi(TWSR, TWPS1);
  TWBR = ((CPU_FREQ / TWI_FREQ) - 16) / 2;

  /* twi bit rate formula from atmega128 manual pg 204
  SCL Frequency = CPU Clock Frequency / (16 + (2 * TWBR))
  note: TWBR should be 10 or higher for master mode
  It is 72 for a 16mhz Wiring board with 100kHz TWI */

  // enable twi module, acks, and twi interrupt
  TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWEA);
}

/*
 * Function twi_slaveInit
 * Desc     sets slave address and enables interrupt
 * Input    none
 * Output   none
 */
void twi_setAddress(uint8_t address)
{
  // set twi slave address (skip over TWGCE bit)
  TWAR = address << 1;
```

```
}

/*
 * Function twi_readFrom
 * Desc     attempts to become twi bus master and read a
 *          series of bytes from a device on the bus
 * Input    address: 7bit i2c device address
 *          data: pointer to byte array
 *          length: number of bytes to read into array
 * Output   number of bytes read
 */
uint8_t twi_readFrom(uint8_t address, uint8_t* data, uint8_t length)
{
  uint8_t i;

  // ensure data will fit into buffer
  if(TWI_BUFFER_LENGTH < length){
    return 0;
  }

  // wait until twi is ready, become master receiver
  while(TWI_READY != twi_state){
    continue;
  }
  twi_state = TWI_MRX;
  // reset error state (0xFF.. no error occured)
```

```
twi_error = 0xFF;

// initialize buffer iteration vars
twi_masterBufferIndex = 0;
twi_masterBufferLength = length-1;  // This is not intuitive, read on...
// On receive, the previously configured ACK/NACK setting is transmitted in
// response to the received byte before the interrupt is signalled.
// Therefor we must actually set NACK when the _next_ to last byte is
// received, causing that NACK to be sent in response to receiving the last
// expected byte of data.

// build sla+w, slave device address + w bit
twi_slarw = TW_READ;
twi_slarw |= address << 1;

// send start condition
TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWEA) | _BV(TWINT) | _BV(TWSTA);

// wait for read operation to complete
while(TWI_MRX == twi_state){
  continue;
}

if (twi_masterBufferIndex < length)
  length = twi_masterBufferIndex;
```

```
  // copy twi buffer to data
  for(i = 0; i < length; ++i){
    data[i] = twi_masterBuffer[i];
  }

  return length;
}

/*
 * Function twi_writeTo
 * Desc     attempts to become twi bus master and write a
 *          series of bytes to a device on the bus
 * Input    address: 7bit i2c device address
 *          data: pointer to byte array
 *          length: number of bytes in array
 *          wait: boolean indicating to wait for write or not
 * Output   0 .. success
 *          1 .. length to long for buffer
 *          2 .. address send, NACK received
 *          3 .. data send, NACK received
 *          4 .. other twi error (lost bus arbitration, bus error, ..)
 */
uint8_t twi_writeTo(uint8_t address, uint8_t* data, uint8_t length, uint8_t wait)
{
  uint8_t i;
```

```
// ensure data will fit into buffer
if(TWI_BUFFER_LENGTH < length){
  return 1;
}

// wait until twi is ready, become master transmitter
while(TWI_READY != twi_state){
  continue;
}
twi_state = TWI_MTX;
// reset error state (0xFF.. no error occured)
twi_error = 0xFF;

// initialize buffer iteration vars
twi_masterBufferIndex = 0;
twi_masterBufferLength = length;

// copy data to twi buffer
for(i = 0; i < length; ++i){
  twi_masterBuffer[i] = data[i];
}

// build sla+w, slave device address + w bit
twi_slarw = TW_WRITE;
twi_slarw |= address << 1;
```

```
  // send start condition
  TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWEA) | _BV(TWINT) | _BV(TWSTA);

  // wait for write operation to complete
  while(wait && (TWI_MTX == twi_state)){
    continue;
  }

  if (twi_error == 0xFF)
    return 0;        // success
  else if (twi_error == TW_MT_SLA_NACK)
    return 2;        // error: address send, nack received
  else if (twi_error == TW_MT_DATA_NACK)
    return 3;        // error: data send, nack received
  else
    return 4;        // other twi error
}

/*
 * Function twi_transmit
 * Desc     fills slave tx buffer with data
 *          must be called in slave tx event callback
 * Input    data: pointer to byte array
 *          length: number of bytes in array
 * Output   1 length too long for buffer
 *          2 not slave transmitter
```

```
 *          0 ok
 */
uint8_t twi_transmit(uint8_t* data, uint8_t length)
{
  uint8_t i;

  // ensure data will fit into buffer
  if(TWI_BUFFER_LENGTH < length){
    return 1;
  }

  // ensure we are currently a slave transmitter
  if(TWI_STX != twi_state){
    return 2;
  }

  // set length and copy data into tx buffer
  twi_txBufferLength = length;
  for(i = 0; i < length; ++i){
    twi_txBuffer[i] = data[i];
  }

  return 0;
}

/*
```

```
 * Function twi_attachSlaveRxEvent
 * Desc     sets function called before a slave read operation
 * Input    function: callback function to use
 * Output   none
 */
void twi_attachSlaveRxEvent( void (*function)(uint8_t*, int) )
{
  twi_onSlaveReceive = function;
}

/*
 * Function twi_attachSlaveTxEvent
 * Desc     sets function called before a slave write operation
 * Input    function: callback function to use
 * Output   none
 */
void twi_attachSlaveTxEvent( void (*function)(void) )
{
  twi_onSlaveTransmit = function;
}

/*
 * Function twi_reply
 * Desc     sends byte or readys receive line
 * Input    ack: byte indicating to ack or to nack
 * Output   none
```

```
 */
void twi_reply(uint8_t ack)
{
  // transmit master read ready signal, with or without ack
  if(ack){
    TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWINT) | _BV(TWEA);
  }else{
            TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWINT);
  }
}

/*
 * Function twi_stop
 * Desc     relinquishes bus master status
 * Input    none
 * Output   none
 */
void twi_stop(void)
{
  // send stop condition
  TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWEA) | _BV(TWINT) | _BV(TWSTO);

  // wait for stop condition to be exectued on bus
  // TWINT is not set after a stop condition!
  while(TWCR & _BV(TWSTO)){
    continue;
```

```
  }

  // update twi state
  twi_state = TWI_READY;
}

/*
 * Function twi_releaseBus
 * Desc     releases bus control
 * Input    none
 * Output   none
 */
void twi_releaseBus(void)
{
  // release bus
  TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWEA) | _BV(TWINT);

  // update twi state
  twi_state = TWI_READY;
}

SIGNAL(TWI_vect)
{
  switch(TW_STATUS){
    // All Master
    case TW_START:     // sent start condition
```

```
case TW_REP_START: // sent repeated start condition
  // copy device address and r/w bit to output register and ack
  TWDR = twi_slarw;
  twi_reply(1);
  break;

// Master Transmitter
case TW_MT_SLA_ACK:  // slave receiver acked address
case TW_MT_DATA_ACK: // slave receiver acked data
  // if there is data to send, send it, otherwise stop
  if(twi_masterBufferIndex < twi_masterBufferLength){
    // copy data to output register and ack
    TWDR = twi_masterBuffer[twi_masterBufferIndex++];
    twi_reply(1);
  }else{
    twi_stop();
  }
  break;
case TW_MT_SLA_NACK:  // address sent, nack received
  twi_error = TW_MT_SLA_NACK;
  twi_stop();
  break;
case TW_MT_DATA_NACK: // data sent, nack received
  twi_error = TW_MT_DATA_NACK;
  twi_stop();
  break;
```

```
case TW_MT_ARB_LOST: // lost bus arbitration
  twi_error = TW_MT_ARB_LOST;
  twi_releaseBus();
  break;

// Master Receiver
case TW_MR_DATA_ACK: // data received, ack sent
  // put byte into buffer
  twi_masterBuffer[twi_masterBufferIndex++] = TWDR;
case TW_MR_SLA_ACK:  // address sent, ack received
  // ack if more bytes are expected, otherwise nack
  if(twi_masterBufferIndex < twi_masterBufferLength){
    twi_reply(1);
  }else{
    twi_reply(0);
  }
  break;
case TW_MR_DATA_NACK: // data received, nack sent
  // put final byte into buffer
  twi_masterBuffer[twi_masterBufferIndex++] = TWDR;
case TW_MR_SLA_NACK: // address sent, nack received
  twi_stop();
  break;
// TW_MR_ARB_LOST handled by TW_MT_ARB_LOST case

// Slave Receiver
```

```
case TW_SR_SLA_ACK:   // addressed, returned ack
case TW_SR_GCALL_ACK: // addressed generally, returned ack
case TW_SR_ARB_LOST_SLA_ACK:   // lost arbitration, returned ack
case TW_SR_ARB_LOST_GCALL_ACK: // lost arbitration, returned ack
  // enter slave receiver mode
  twi_state = TWI_SRX;
  // indicate that rx buffer can be overwritten and ack
  twi_rxBufferIndex = 0;
  twi_reply(1);
  break;
case TW_SR_DATA_ACK:       // data received, returned ack
case TW_SR_GCALL_DATA_ACK: // data received generally, returned ack
  // if there is still room in the rx buffer
  if(twi_rxBufferIndex < TWI_BUFFER_LENGTH){
    // put byte in buffer and ack
    twi_rxBuffer[twi_rxBufferIndex++] = TWDR;
    twi_reply(1);
  }else{
    // otherwise nack
    twi_reply(0);
  }
  break;
case TW_SR_STOP: // stop or repeated start condition received
  // put a null char after data if there's room
  if(twi_rxBufferIndex < TWI_BUFFER_LENGTH){
    twi_rxBuffer[twi_rxBufferIndex] = '\0';
```

```
  }
  // sends ack and stops interface for clock stretching
  twi_stop();
  // callback to user defined callback
  twi_onSlaveReceive(twi_rxBuffer, twi_rxBufferIndex);
  // since we submit rx buffer to "wire" library, we can reset it
  twi_rxBufferIndex = 0;
  // ack future responses and leave slave receiver state
  twi_releaseBus();
  break;
case TW_SR_DATA_NACK:       // data received, returned nack
case TW_SR_GCALL_DATA_NACK: // data received generally, returned nack
  // nack back at master
  twi_reply(0);
  break;

// Slave Transmitter
case TW_ST_SLA_ACK:          // addressed, returned ack
case TW_ST_ARB_LOST_SLA_ACK: // arbitration lost, returned ack
  // enter slave transmitter mode
  twi_state = TWI_STX;
  // ready the tx buffer index for iteration
  twi_txBufferIndex = 0;
  // set tx buffer length to be zero, to verify if user changes it
  twi_txBufferLength = 0;
  // request for txBuffer to be filled and length to be set
```

```
  // note: user must call twi_transmit(bytes, length) to do this
  twi_onSlaveTransmit();
  // if they didn't change buffer & length, initialize it
  if(0 == twi_txBufferLength){
    twi_txBufferLength = 1;
    twi_txBuffer[0] = 0x00;
  }
  // transmit first byte from buffer, fall
case TW_ST_DATA_ACK: // byte sent, ack returned
  // copy data to output register
  TWDR = twi_txBuffer[twi_txBufferIndex++];
  // if there is more to send, ack, otherwise nack
  if(twi_txBufferIndex < twi_txBufferLength){
    twi_reply(1);
  }else{
    twi_reply(0);
  }
  break;
case TW_ST_DATA_NACK: // received nack, we are done
case TW_ST_LAST_DATA: // received ack, but we are done already!
  // ack future responses
  twi_reply(1);
  // leave slave receiver state
  twi_state = TWI_READY;
  break;
```

```
    // All
    case TW_NO_INFO:   // no state information
      break;
    case TW_BUS_ERROR: // bus error, illegal stop/start
      twi_error = TW_BUS_ERROR;
      twi_stop();
      break;
  }
}
```

Wire.cpp

```
/*
  TwoWire.cpp - TWI/I2C library for Wiring & Arduino
  Copyright (c) 2006 Nicholas Zambetti.  All right reserved.

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
*/

extern "C" {
  #include <stdlib.h>
  #include <string.h>
  #include <inttypes.h>
}
```

```
#include "..\wire\twi.h"
#include <Wire.h>

// Initialize Class Variables /////////////////////////////////////////////////

uint8_t TwoWire::rxBuffer[BUFFER_LENGTH];
uint8_t TwoWire::rxBufferIndex = 0;
uint8_t TwoWire::rxBufferLength = 0;

uint8_t TwoWire::txAddress = 0;
uint8_t TwoWire::txBuffer[BUFFER_LENGTH];
uint8_t TwoWire::txBufferIndex = 0;
uint8_t TwoWire::txBufferLength = 0;

uint8_t TwoWire::transmitting = 0;
void (*TwoWire::user_onRequest)(void);
void (*TwoWire::user_onReceive)(int);

// Constructors ////////////////////////////////////////////////////////////////

TwoWire::TwoWire()
{
}
```

```
// Public Methods //////////////////////////////////////////////////////////////


void TwoWire::begin(void)
{
  rxBufferIndex = 0;
  rxBufferLength = 0;

  txBufferIndex = 0;
  txBufferLength = 0;

  twi_init();
}

void TwoWire::begin(uint8_t address)
{
  twi_setAddress(address);
  twi_attachSlaveTxEvent(onRequestService);
  twi_attachSlaveRxEvent(onReceiveService);
  begin();
}

void TwoWire::begin(int address)
{
  begin((uint8_t)address);
}
```

```
uint8_t TwoWire::requestFrom(uint8_t address, uint8_t quantity)
{
  // clamp to buffer length
  if(quantity > BUFFER_LENGTH){
    quantity = BUFFER_LENGTH;
  }
  // perform blocking read into buffer
  uint8_t read = twi_readFrom(address, rxBuffer, quantity);
  // set rx buffer iterator vars
  rxBufferIndex = 0;
  rxBufferLength = read;

  return read;
}

uint8_t TwoWire::requestFrom(int address, int quantity)
{
  return requestFrom((uint8_t)address, (uint8_t)quantity);
}

void TwoWire::beginTransmission(uint8_t address)
{
  // indicate that we are transmitting
  transmitting = 1;
  // set address of targeted slave
```

```
  txAddress = address;
  // reset tx buffer iterator vars
  txBufferIndex = 0;
  txBufferLength = 0;
}

void TwoWire::beginTransmission(int address)
{
  beginTransmission((uint8_t)address);
}

uint8_t TwoWire::endTransmission(void)
{
  // transmit buffer (blocking)
  int8_t ret = twi_writeTo(txAddress, txBuffer, txBufferLength, 1);
  // reset tx buffer iterator vars
  txBufferIndex = 0;
  txBufferLength = 0;
  // indicate that we are done transmitting
  transmitting = 0;
  return ret;
}

// must be called in:
// slave tx event callback
// or after beginTransmission(address)
```

```
void TwoWire::send(uint8_t data)
{
  if(transmitting){
  // in master transmitter mode
    // don't bother if buffer is full
    if(txBufferLength >= BUFFER_LENGTH){
      return;
    }
    // put byte in tx buffer
    txBuffer[txBufferIndex] = data;
    ++txBufferIndex;
    // update amount in buffer
    txBufferLength = txBufferIndex;
  }else{
  // in slave send mode
    // reply to master
    twi_transmit(&data, 1);
  }
}

// must be called in:
// slave tx event callback
// or after beginTransmission(address)
void TwoWire::send(uint8_t* data, uint8_t quantity)
{
  if(transmitting){
```

```
  // in master transmitter mode
    for(uint8_t i = 0; i < quantity; ++i){
      send(data[i]);
    }
  }else{
  // in slave send mode
    // reply to master
    twi_transmit(data, quantity);
  }
}

// must be called in:
// slave tx event callback
// or after beginTransmission(address)
void TwoWire::send(char* data)
{
  send((uint8_t*)data, strlen(data));
}

// must be called in:
// slave tx event callback
// or after beginTransmission(address)
void TwoWire::send(int data)
{
  send((uint8_t)data);
}
```

```
// must be called in:
// slave rx event callback
// or after requestFrom(address, numBytes)
uint8_t TwoWire::available(void)
{
  return rxBufferLength - rxBufferIndex;
}

// must be called in:
// slave rx event callback
// or after requestFrom(address, numBytes)
uint8_t TwoWire::receive(void)
{
  // default to returning null char
  // for people using with char strings
  uint8_t value = '\0';

  // get each successive byte on each call
  if(rxBufferIndex < rxBufferLength){
    value = rxBuffer[rxBufferIndex];
    ++rxBufferIndex;
  }

  return value;
}
```

```
// behind the scenes function that is called when data is received
void TwoWire::onReceiveService(uint8_t* inBytes, int numBytes)
{
  // don't bother if user hasn't registered a callback
  if(!user_onReceive){
    return;
  }
  // don't bother if rx buffer is in use by a master requestFrom() op
  // i know this drops data, but it allows for slight stupidity
  // meaning, they may not have read all the master requestFrom() data yet
  if(rxBufferIndex < rxBufferLength){
    return;
  }
  // copy twi rx buffer into local read buffer
  // this enables new reads to happen in parallel
  for(uint8_t i = 0; i < numBytes; ++i){
    rxBuffer[i] = inBytes[i];
  }
  // set rx iterator vars
  rxBufferIndex = 0;
  rxBufferLength = numBytes;
  // alert user program
  user_onReceive(numBytes);
}
```

```
// behind the scenes function that is called when data is requested
void TwoWire::onRequestService(void)
{
  // don't bother if user hasn't registered a callback
  if(!user_onRequest){
    return;
  }
  // reset tx buffer iterator vars
  // !!! this will kill any pending pre-master sendTo() activity
  txBufferIndex = 0;
  txBufferLength = 0;
  // alert user program
  user_onRequest();
}

// sets function called on slave write
void TwoWire::onReceive( void (*function)(int) )
{
  user_onReceive = function;
}

// sets function called on slave read
void TwoWire::onRequest( void (*function)(void) )
{
  user_onRequest = function;
}
```

```
// Preinstantiate Objects //////////////////////////////////////////////////

TwoWire Wire = TwoWire();
```

Pins_arduino.c

```
/*
  pins_arduino.c - pin definitions for the Arduino board
  Part of Arduino / Wiring Lite

  Copyright (c) 2005 David A. Mellis

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General
  Public License along with this library; if not, write to the
  Free Software Foundation, Inc., 59 Temple Place, Suite 330,
  Boston, MA  02111-1307  USA

  $Id$
*/

#include <avr/io.h>
```

```
#include <util/delay.h>
#include "wiring_private.h"
#include "pins_arduino.h"

// On the Arduino board, digital pins are also used
// for the analog output (software PWM).  Analog input
// pins are a separate set.

// ATMEL ATMEGA8 & 168 / ARDUINO
//
//               +-\/-+
//           PC6  1|    |28  PC5 (AI 5)
//     (D 0) PD0  2|    |27  PC4 (AI 4)
//     (D 1) PD1  3|    |26  PC3 (AI 3)
//     (D 2) PD2  4|    |25  PC2 (AI 2)
// PWM+ (D 3) PD3  5|    |24  PC1 (AI 1)
//     (D 4) PD4  6|    |23  PC0 (AI 0)
//           VCC  7|    |22  GND
//           GND  8|    |21  AREF
//           PB6  9|    |20  AVCC
//           PB7 10|    |19  PB5 (D 13)
// PWM+ (D 5) PD5 11|    |18  PB4 (D 12)
// PWM+ (D 6) PD6 12|    |17  PB3 (D 11) PWM
//     (D 7) PD7 13|    |16  PB2 (D 10) PWM
//     (D 8) PB0 14|    |15  PB1 (D 9) PWM
//               +----+
```

```
//
// (PWM+ indicates the additional PWM pins on the ATmega168.)

// ATMEL ATMEGA1280 / ARDUINO
//
// 0-7 PE0-PE7   works
// 8-13 PB0-PB5  works
// 14-21 PA0-PA7 works
// 22-29 PH0-PH7 works
// 30-35 PG5-PG0 works
// 36-43 PC7-PC0 works
// 44-51 PJ7-PJ0 works
// 52-59 PL7-PL0 works
// 60-67 PD7-PD0 works
// A0-A7 PF0-PF7
// A8-A15 PK0-PK7

#define PA 1
#define PB 2
#define PC 3
#define PD 4
#define PE 5
#define PF 6
#define PG 7
#define PH 8
#define PJ 10
```

```
#define PK 11
#define PL 12


#if defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__)
const uint16_t PROGMEM port_to_mode_PGM[] = {
	(uint16_t)NOT_A_PORT,
	(uint16_t)&DDRA,
	(uint16_t)&DDRB,
	(uint16_t)&DDRC,
	(uint16_t)&DDRD,
	(uint16_t)&DDRE,
	(uint16_t)&DDRF,
	(uint16_t)&DDRG,
	(uint16_t)&DDRH,
	(uint16_t)NOT_A_PORT,
	(uint16_t)&DDRJ,
	(uint16_t)&DDRK,
	(uint16_t)&DDRL,
};

const uint16_t PROGMEM port_to_output_PGM[] = {
	(uint16_t)NOT_A_PORT,
	(uint16_t)&PORTA,
	(uint16_t)&PORTB,
	(uint16_t)&PORTC,
```

```
        (uint16_t)&PORTD,
        (uint16_t)&PORTE,
        (uint16_t)&PORTF,
        (uint16_t)&PORTG,
        (uint16_t)&PORTH,
        (uint16_t)NOT_A_PORT,
        (uint16_t)&PORTJ,
        (uint16_t)&PORTK,
        (uint16_t)&PORTL,
};

const uint16_t PROGMEM port_to_input_PGM[] = {
        (uint16_t)NOT_A_PIN,
        (uint16_t)&PINA,
        (uint16_t)&PINB,
        (uint16_t)&PINC,
        (uint16_t)&PIND,
        (uint16_t)&PINE,
        (uint16_t)&PINF,
        (uint16_t)&PING,
        (uint16_t)&PINH,
        (uint16_t)NOT_A_PIN,
        (uint16_t)&PINJ,
        (uint16_t)&PINK,
        (uint16_t)&PINL,
};
```

```
const uint8_t PROGMEM digital_pin_to_port_PGM[] = {
	// PORTLIST
	// -------------------------------------------
	PE	, // PE 0 ** 0 ** USART0_RX
	PE	, // PE 1 ** 1 ** USART0_TX
	PE	, // PE 4 ** 2 ** PWM2
	PE	, // PE 5 ** 3 ** PWM3
	PG	, // PG 5 ** 4 ** PWM4
	PE	, // PE 3 ** 5 ** PWM5
	PH	, // PH 3 ** 6 ** PWM6
	PH	, // PH 4 ** 7 ** PWM7
	PH	, // PH 5 ** 8 ** PWM8
	PH	, // PH 6 ** 9 ** PWM9
	PB	, // PB 4 ** 10 ** PWM10
	PB	, // PB 5 ** 11 ** PWM11
	PB	, // PB 6 ** 12 ** PWM12
	PB	, // PB 7 ** 13 ** PWM13
	PJ	, // PJ 1 ** 14 ** USART3_TX
	PJ	, // PJ 0 ** 15 ** USART3_RX
	PH	, // PH 1 ** 16 ** USART2_TX
	PH	, // PH 0 ** 17 ** USART2_RX
	PD	, // PD 3 ** 18 ** USART1_TX
	PD	, // PD 2 ** 19 ** USART1_RX
	PD	, // PD 1 ** 20 ** I2C_SDA
	PD	, // PD 0 ** 21 ** I2C_SCL
```

```
PA      , // PA 0 ** 22 ** D22
PA      , // PA 1 ** 23 ** D23
PA      , // PA 2 ** 24 ** D24
PA      , // PA 3 ** 25 ** D25
PA      , // PA 4 ** 26 ** D26
PA      , // PA 5 ** 27 ** D27
PA      , // PA 6 ** 28 ** D28
PA      , // PA 7 ** 29 ** D29
PC      , // PC 7 ** 30 ** D30
PC      , // PC 6 ** 31 ** D31
PC      , // PC 5 ** 32 ** D32
PC      , // PC 4 ** 33 ** D33
PC      , // PC 3 ** 34 ** D34
PC      , // PC 2 ** 35 ** D35
PC      , // PC 1 ** 36 ** D36
PC      , // PC 0 ** 37 ** D37
PD      , // PD 7 ** 38 ** D38
PG      , // PG 2 ** 39 ** D39
PG      , // PG 1 ** 40 ** D40
PG      , // PG 0 ** 41 ** D41
PL      , // PL 7 ** 42 ** D42
PL      , // PL 6 ** 43 ** D43
PL      , // PL 5 ** 44 ** D44
PL      , // PL 4 ** 45 ** D45
PL      , // PL 3 ** 46 ** D46
PL      , // PL 2 ** 47 ** D47
```

```
        PL      , // PL 1 ** 48 ** D48
        PL      , // PL 0 ** 49 ** D49
        PB      , // PB 3 ** 50 ** SPI_MISO
        PB      , // PB 2 ** 51 ** SPI_MOSI
        PB      , // PB 1 ** 52 ** SPI_SCK
        PB      , // PB 0 ** 53 ** SPI_SS
        PF      , // PF 0 ** 54 ** A0
        PF      , // PF 1 ** 55 ** A1
        PF      , // PF 2 ** 56 ** A2
        PF      , // PF 3 ** 57 ** A3
        PF      , // PF 4 ** 58 ** A4
        PF      , // PF 5 ** 59 ** A5
        PF      , // PF 6 ** 60 ** A6
        PF      , // PF 7 ** 61 ** A7
        PK      , // PK 0 ** 62 ** A8
        PK      , // PK 1 ** 63 ** A9
        PK      , // PK 2 ** 64 ** A10
        PK      , // PK 3 ** 65 ** A11
        PK      , // PK 4 ** 66 ** A12
        PK      , // PK 5 ** 67 ** A13
        PK      , // PK 6 ** 68 ** A14
        PK      , // PK 7 ** 69 ** A15
};

const uint8_t PROGMEM digital_pin_to_bit_mask_PGM[] = {
        // PIN IN PORT
```

```
// ----------------------------------------
_BV( 0 )        , // PE 0 ** 0 ** USART0_RX
_BV( 1 )        , // PE 1 ** 1 ** USART0_TX
_BV( 4 )        , // PE 4 ** 2 ** PWM2
_BV( 5 )        , // PE 5 ** 3 ** PWM3
_BV( 5 )        , // PG 5 ** 4 ** PWM4
_BV( 3 )        , // PE 3 ** 5 ** PWM5
_BV( 3 )        , // PH 3 ** 6 ** PWM6
_BV( 4 )        , // PH 4 ** 7 ** PWM7
_BV( 5 )        , // PH 5 ** 8 ** PWM8
_BV( 6 )        , // PH 6 ** 9 ** PWM9
_BV( 4 )        , // PB 4 ** 10 ** PWM10
_BV( 5 )        , // PB 5 ** 11 ** PWM11
_BV( 6 )        , // PB 6 ** 12 ** PWM12
_BV( 7 )        , // PB 7 ** 13 ** PWM13
_BV( 1 )        , // PJ 1 ** 14 ** USART3_TX
_BV( 0 )        , // PJ 0 ** 15 ** USART3_RX
_BV( 1 )        , // PH 1 ** 16 ** USART2_TX
_BV( 0 )        , // PH 0 ** 17 ** USART2_RX
_BV( 3 )        , // PD 3 ** 18 ** USART1_TX
_BV( 2 )        , // PD 2 ** 19 ** USART1_RX
_BV( 1 )        , // PD 1 ** 20 ** I2C_SDA
_BV( 0 )        , // PD 0 ** 21 ** I2C_SCL
_BV( 0 )        , // PA 0 ** 22 ** D22
_BV( 1 )        , // PA 1 ** 23 ** D23
_BV( 2 )        , // PA 2 ** 24 ** D24
```

```
_BV( 3 )        , // PA 3 ** 25 ** D25
_BV( 4 )        , // PA 4 ** 26 ** D26
_BV( 5 )        , // PA 5 ** 27 ** D27
_BV( 6 )        , // PA 6 ** 28 ** D28
_BV( 7 )        , // PA 7 ** 29 ** D29
_BV( 7 )        , // PC 7 ** 30 ** D30
_BV( 6 )        , // PC 6 ** 31 ** D31
_BV( 5 )        , // PC 5 ** 32 ** D32
_BV( 4 )        , // PC 4 ** 33 ** D33
_BV( 3 )        , // PC 3 ** 34 ** D34
_BV( 2 )        , // PC 2 ** 35 ** D35
_BV( 1 )        , // PC 1 ** 36 ** D36
_BV( 0 )        , // PC 0 ** 37 ** D37
_BV( 7 )        , // PD 7 ** 38 ** D38
_BV( 2 )        , // PG 2 ** 39 ** D39
_BV( 1 )        , // PG 1 ** 40 ** D40
_BV( 0 )        , // PG 0 ** 41 ** D41
_BV( 7 )        , // PL 7 ** 42 ** D42
_BV( 6 )        , // PL 6 ** 43 ** D43
_BV( 5 )        , // PL 5 ** 44 ** D44
_BV( 4 )        , // PL 4 ** 45 ** D45
_BV( 3 )        , // PL 3 ** 46 ** D46
_BV( 2 )        , // PL 2 ** 47 ** D47
_BV( 1 )        , // PL 1 ** 48 ** D48
_BV( 0 )        , // PL 0 ** 49 ** D49
_BV( 3 )        , // PB 3 ** 50 ** SPI_MISO
```

```
        _BV( 2 )        , // PB 2 ** 51 ** SPI_MOSI
        _BV( 1 )        , // PB 1 ** 52 ** SPI_SCK
        _BV( 0 )        , // PB 0 ** 53 ** SPI_SS
        _BV( 0 )        , // PF 0 ** 54 ** A0
        _BV( 1 )        , // PF 1 ** 55 ** A1
        _BV( 2 )        , // PF 2 ** 56 ** A2
        _BV( 3 )        , // PF 3 ** 57 ** A3
        _BV( 4 )        , // PF 4 ** 58 ** A4
        _BV( 5 )        , // PF 5 ** 59 ** A5
        _BV( 6 )        , // PF 6 ** 60 ** A6
        _BV( 7 )        , // PF 7 ** 61 ** A7
        _BV( 0 )        , // PK 0 ** 62 ** A8
        _BV( 1 )        , // PK 1 ** 63 ** A9
        _BV( 2 )        , // PK 2 ** 64 ** A10
        _BV( 3 )        , // PK 3 ** 65 ** A11
        _BV( 4 )        , // PK 4 ** 66 ** A12
        _BV( 5 )        , // PK 5 ** 67 ** A13
        _BV( 6 )        , // PK 6 ** 68 ** A14
        _BV( 7 )        , // PK 7 ** 69 ** A15
};

const uint8_t PROGMEM digital_pin_to_timer_PGM[] = {
        // TIMERS
        // ---------------------------------------
        NOT_ON_TIMER    , // PE 0 ** 0 ** USART0_RX
        NOT_ON_TIMER    , // PE 1 ** 1 ** USART0_TX
```

```
TIMER3B       , // PE 4 ** 2 ** PWM2
TIMER3C       , // PE 5 ** 3 ** PWM3
TIMER0B       , // PG 5 ** 4 ** PWM4
TIMER3A       , // PE 3 ** 5 ** PWM5
TIMER4A       , // PH 3 ** 6 ** PWM6
TIMER4B       , // PH 4 ** 7 ** PWM7
TIMER4C       , // PH 5 ** 8 ** PWM8
TIMER2B       , // PH 6 ** 9 ** PWM9
TIMER2A       , // PB 4 ** 10 ** PWM10
TIMER1A       , // PB 5 ** 11 ** PWM11
TIMER1B       , // PB 6 ** 12 ** PWM12
TIMER0A       , // PB 7 ** 13 ** PWM13
NOT_ON_TIMER      , // PJ 1 ** 14 ** USART3_TX
NOT_ON_TIMER      , // PJ 0 ** 15 ** USART3_RX
NOT_ON_TIMER      , // PH 1 ** 16 ** USART2_TX
NOT_ON_TIMER      , // PH 0 ** 17 ** USART2_RX
NOT_ON_TIMER      , // PD 3 ** 18 ** USART1_TX
NOT_ON_TIMER      , // PD 2 ** 19 ** USART1_RX
NOT_ON_TIMER      , // PD 1 ** 20 ** I2C_SDA
NOT_ON_TIMER      , // PD 0 ** 21 ** I2C_SCL
NOT_ON_TIMER      , // PA 0 ** 22 ** D22
NOT_ON_TIMER      , // PA 1 ** 23 ** D23
NOT_ON_TIMER      , // PA 2 ** 24 ** D24
NOT_ON_TIMER      , // PA 3 ** 25 ** D25
NOT_ON_TIMER      , // PA 4 ** 26 ** D26
NOT_ON_TIMER      , // PA 5 ** 27 ** D27
```

```
NOT_ON_TIMER    , // PA 6 ** 28 ** D28
NOT_ON_TIMER    , // PA 7 ** 29 ** D29
NOT_ON_TIMER    , // PC 7 ** 30 ** D30
NOT_ON_TIMER    , // PC 6 ** 31 ** D31
NOT_ON_TIMER    , // PC 5 ** 32 ** D32
NOT_ON_TIMER    , // PC 4 ** 33 ** D33
NOT_ON_TIMER    , // PC 3 ** 34 ** D34
NOT_ON_TIMER    , // PC 2 ** 35 ** D35
NOT_ON_TIMER    , // PC 1 ** 36 ** D36
NOT_ON_TIMER    , // PC 0 ** 37 ** D37
NOT_ON_TIMER    , // PD 7 ** 38 ** D38
NOT_ON_TIMER    , // PG 2 ** 39 ** D39
NOT_ON_TIMER    , // PG 1 ** 40 ** D40
NOT_ON_TIMER    , // PG 0 ** 41 ** D41
NOT_ON_TIMER    , // PL 7 ** 42 ** D42
NOT_ON_TIMER    , // PL 6 ** 43 ** D43
TIMER5C     , // PL 5 ** 44 ** D44
TIMER5B     , // PL 4 ** 45 ** D45
TIMER5A     , // PL 3 ** 46 ** D46
NOT_ON_TIMER    , // PL 2 ** 47 ** D47
NOT_ON_TIMER    , // PL 1 ** 48 ** D48
NOT_ON_TIMER    , // PL 0 ** 49 ** D49
NOT_ON_TIMER    , // PB 3 ** 50 ** SPI_MISO
NOT_ON_TIMER    , // PB 2 ** 51 ** SPI_MOSI
NOT_ON_TIMER    , // PB 1 ** 52 ** SPI_SCK
NOT_ON_TIMER    , // PB 0 ** 53 ** SPI_SS
```

```
        NOT_ON_TIMER    , // PF 0 ** 54 ** A0
        NOT_ON_TIMER    , // PF 1 ** 55 ** A1
        NOT_ON_TIMER    , // PF 2 ** 56 ** A2
        NOT_ON_TIMER    , // PF 3 ** 57 ** A3
        NOT_ON_TIMER    , // PF 4 ** 58 ** A4
        NOT_ON_TIMER    , // PF 5 ** 59 ** A5
        NOT_ON_TIMER    , // PF 6 ** 60 ** A6
        NOT_ON_TIMER    , // PF 7 ** 61 ** A7
        NOT_ON_TIMER    , // PK 0 ** 62 ** A8
        NOT_ON_TIMER    , // PK 1 ** 63 ** A9
        NOT_ON_TIMER    , // PK 2 ** 64 ** A10
        NOT_ON_TIMER    , // PK 3 ** 65 ** A11
        NOT_ON_TIMER    , // PK 4 ** 66 ** A12
        NOT_ON_TIMER    , // PK 5 ** 67 ** A13
        NOT_ON_TIMER    , // PK 6 ** 68 ** A14
        NOT_ON_TIMER    , // PK 7 ** 69 ** A15
};
#else
// these arrays map port names (e.g. port B) to the
// appropriate addresses for various functions (e.g. reading
// and writing)
const uint16_t PROGMEM port_to_mode_PGM[] = {
        (uint16_t)NOT_A_PORT,
        (uint16_t)NOT_A_PORT,
        (uint16_t)&DDRB,
        (uint16_t)&DDRC,
```

```
	(uint16_t)&DDRD,
};

const uint16_t PROGMEM port_to_output_PGM[] = {
	(uint16_t)NOT_A_PORT,
	(uint16_t)NOT_A_PORT,
	(uint16_t)&PORTB,
	(uint16_t)&PORTC,
	(uint16_t)&PORTD,
};

const uint16_t PROGMEM port_to_input_PGM[] = {
	(uint16_t)NOT_A_PORT,
	(uint16_t)NOT_A_PORT,
	(uint16_t)&PINB,
	(uint16_t)&PINC,
	(uint16_t)&PIND,
};

const uint8_t PROGMEM digital_pin_to_port_PGM[] = {
	PD, /* 0 */
	PD,
	PD,
	PD,
	PD,
	PD,
```

```
	PD,
	PD,
	PB, /* 8 */
	PB,
	PB,
	PB,
	PB,
	PB,
	PC, /* 14 */
	PC,
	PC,
	PC,
	PC,
	PC,
};

const uint8_t PROGMEM digital_pin_to_bit_mask_PGM[] = {
	_BV(0), /* 0, port D */
	_BV(1),
	_BV(2),
	_BV(3),
	_BV(4),
	_BV(5),
	_BV(6),
	_BV(7),
	_BV(0), /* 8, port B */
```

```
	_BV(1),
	_BV(2),
	_BV(3),
	_BV(4),
	_BV(5),
	_BV(0), /* 14, port C */
	_BV(1),
	_BV(2),
	_BV(3),
	_BV(4),
	_BV(5),
};

const uint8_t PROGMEM digital_pin_to_timer_PGM[] = {
	NOT_ON_TIMER, /* 0 - port D */
	NOT_ON_TIMER,
	NOT_ON_TIMER,
	// on the ATmega168, digital pin 3 has hardware pwm
#if defined(__AVR_ATmega8__)
	NOT_ON_TIMER,
#else
	TIMER2B,
#endif
	NOT_ON_TIMER,
	// on the ATmega168, digital pins 5 and 6 have hardware pwm
#if defined(__AVR_ATmega8__)
```

```
        NOT_ON_TIMER,
        NOT_ON_TIMER,
#else
        TIMER0B,
        TIMER0A,
#endif
        NOT_ON_TIMER,
        NOT_ON_TIMER, /* 8 - port B */
        TIMER1A,
        TIMER1B,
#if defined(__AVR_ATmega8__)
        TIMER2,
#else
        TIMER2A,
#endif
        NOT_ON_TIMER,
        NOT_ON_TIMER,
        NOT_ON_TIMER,
        NOT_ON_TIMER, /* 14 - port C */
        NOT_ON_TIMER,
        NOT_ON_TIMER,
        NOT_ON_TIMER,
        NOT_ON_TIMER,
};
#endif
```

Print.cpp

```
/*
 Print.cpp - Base class that provides print() and println()
 Copyright (c) 2008 David A. Mellis.  All right reserved.

 This library is free software; you can redistribute it and/or
 modify it under the terms of the GNU Lesser General Public
 License as published by the Free Software Foundation; either
 version 2.1 of the License, or (at your option) any later version.

 This library is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 Lesser General Public License for more details.

 You should have received a copy of the GNU Lesser General Public
 License along with this library; if not, write to the Free Software
 Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA

 Modified 23 November 2006 by David A. Mellis
 */

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

```
#include "wiring.h"

#include "Print.h"

// Public Methods //////////////////////////////////////////////////////////////

/* default implementation: may be overridden */
void Print::write(const char *str)
{
  while (*str)
    write(*str++);
}

/* default implementation: may be overridden */
void Print::write(const uint8_t *buffer, size_t size)
{
  while (size--)
    write(*buffer++);
}
/*
void Print::print(const String &s)
{
  for (int i = 0; i < s.length(); i++) {
    write(s[i]);
  }
}
```

```
*/
void Print::print(const char str[])
{
  write(str);
}

void Print::print(char c, int base)
{
  print((long) c, base);
}

void Print::print(unsigned char b, int base)
{
  print((unsigned long) b, base);
}

void Print::print(int n, int base)
{
  print((long) n, base);
}

void Print::print(unsigned int n, int base)
{
  print((unsigned long) n, base);
}
```

```
void Print::print(long n, int base)
{
  if (base == 0) {
    write(n);
  } else if (base == 10) {
    if (n < 0) {
      print('-');
      n = -n;
    }
    printNumber(n, 10);
  } else {
    printNumber(n, base);
  }
}

void Print::print(unsigned long n, int base)
{
  if (base == 0) write(n);
  else printNumber(n, base);
}

void Print::print(double n, int digits)
{
  printFloat(n, digits);
}
```

```
void Print::println(void)
{
  print('\r');
  print('\n');
}
/*
void Print::println(const String &s)
{
  print(s);
  println();
}
*/
void Print::println(const char c[])
{
  print(c);
  println();
}

void Print::println(char c, int base)
{
  print(c, base);
  println();
}

void Print::println(unsigned char b, int base)
{
```

```
  print(b, base);
  println();
}

void Print::println(int n, int base)
{
  print(n, base);
  println();
}

void Print::println(unsigned int n, int base)
{
  print(n, base);
  println();
}

void Print::println(long n, int base)
{
  print(n, base);
  println();
}

void Print::println(unsigned long n, int base)
{
  print(n, base);
  println();
```

```
}

void Print::println(double n, int digits)
{
  print(n, digits);
  println();
}

// Private Methods /////////////////////////////////////////////////////////////

void Print::printNumber(unsigned long n, uint8_t base)
{
  unsigned char buf[8 * sizeof(long)]; // Assumes 8-bit chars.
  unsigned long i = 0;

  if (n == 0) {
    print('0');
    return;
  }

  while (n > 0) {
    buf[i++] = n % base;
    n /= base;
  }

  for (; i > 0; i--)
```

```
    print((char) (buf[i - 1] < 10 ?
      '0' + buf[i - 1] :
      'A' + buf[i - 1] - 10));
}

void Print::printFloat(double number, uint8_t digits)
{
  // Handle negative numbers
  if (number < 0.0)
  {
     print('-');
     number = -number;
  }

  // Round correctly so that print(1.999, 2) prints as "2.00"
  double rounding = 0.5;
  for (uint8_t i=0; i<digits; ++i)
    rounding /= 10.0;

  number += rounding;

  // Extract the integer part of the number and print it
  unsigned long int_part = (unsigned long)number;
  double remainder = number - (double)int_part;
  print(int_part);
```

```
  // Print the decimal point, but only if there are digits beyond
  if (digits > 0)
    print(".");

  // Extract digits from the remainder one at a time
  while (digits-- > 0)
  {
    remainder *= 10.0;
    int toPrint = int(remainder);
    print(toPrint);
    remainder -= toPrint;
  }
}
```

Wiring_digital.c

```
/*
  wiring_digital.c - digital input and output functions
  Part of Arduino - http://www.arduino.cc/

  Copyright (c) 2005-2006 David A. Mellis

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General
  Public License along with this library; if not, write to the
  Free Software Foundation, Inc., 59 Temple Place, Suite 330,
  Boston, MA  02111-1307  USA

  Modified 28 September 2010 by Mark Sproul

  $Id: wiring.c 248 2007-02-03 15:36:30Z mellis $
*/
```

```
#include "pins_arduino.h"
#include "wiring_private.h"



void pinMode(uint8_t pin, uint8_t mode)
{
	uint8_t bit = digitalPinToBitMask(pin);
	uint8_t port = digitalPinToPort(pin);
	volatile uint8_t *reg;

	if (port == NOT_A_PIN) return;

	// JWS: can I let the optimizer do this?
	reg = portModeRegister(port);

	if (mode == INPUT) {
		uint8_t oldSREG = SREG;
  cli();
		*reg &= ~bit;
		SREG = oldSREG;
	} else {
		uint8_t oldSREG = SREG;
  cli();
		*reg |= bit;
```

```
                SREG = oldSREG;
        }
}

// Forcing this inline keeps the callers from having to push their own stuff
// on the stack. It is a good performance win and only takes 1 more byte per
// user than calling. (It will take more bytes on the 168.)
//
// But shouldn't this be moved into pinMode? Seems silly to check and do on
// each digitalread or write.
//
// Mark Sproul:
// - Removed inline. Save 170 bytes on atmega1280
// - changed to a switch statment; added 32 bytes but much easier to read and maintain.
// - Added more #ifdefs, now compiles for atmega645
//
//static inline void turnOffPWM(uint8_t timer) __attribute__ ((always_inline));
//static inline void turnOffPWM(uint8_t timer)
static void turnOffPWM(uint8_t timer)
{
        switch (timer)
        {
                #if defined(TCCR1A) && defined(COM1A1)
                case TIMER1A:   cbi(TCCR1A, COM1A1);    break;
                #endif
                #if defined(TCCR1A) && defined(COM1B1)
```

```
case TIMER1B:   cbi(TCCR1A, COM1B1);    break;
#endif

#if defined(TCCR2) && defined(COM21)
case  TIMER2:   cbi(TCCR2, COM21);      break;
#endif

#if defined(TCCR0A) && defined(COM0A1)
case  TIMER0A:  cbi(TCCR0A, COM0A1);    break;
#endif

#if defined(TIMER0B) && defined(COM0B1)
case  TIMER0B:  cbi(TCCR0A, COM0B1);    break;
#endif
#if defined(TCCR2A) && defined(COM2A1)
case  TIMER2A:  cbi(TCCR2A, COM2A1);    break;
#endif
#if defined(TCCR2A) && defined(COM2B1)
case  TIMER2B:  cbi(TCCR2A, COM2B1);    break;
#endif

#if defined(TCCR3A) && defined(COM3A1)
case  TIMER3A:  cbi(TCCR3A, COM3A1);    break;
#endif
#if defined(TCCR3A) && defined(COM3B1)
case  TIMER3B:  cbi(TCCR3A, COM3B1);    break;
```

```
                #endif
                #if defined(TCCR3A) && defined(COM3C1)
                case  TIMER3C:  cbi(TCCR3A, COM3C1);    break;
                #endif

                #if defined(TCCR4A) && defined(COM4A1)
                case  TIMER4A:  cbi(TCCR4A, COM4A1);    break;
                #endif
                #if defined(TCCR4A) && defined(COM4B1)
                case  TIMER4B:  cbi(TCCR4A, COM4B1);    break;
                #endif
                #if defined(TCCR4A) && defined(COM4C1)
                case  TIMER4C:  cbi(TCCR4A, COM4C1);    break;
                #endif
                #if defined(TCCR5A)
                case  TIMER5A:  cbi(TCCR5A, COM5A1);    break;
                case  TIMER5B:  cbi(TCCR5A, COM5B1);    break;
                case  TIMER5C:  cbi(TCCR5A, COM5C1);    break;
                #endif
        }
}

void digitalWrite(uint8_t pin, uint8_t val)
{
        uint8_t timer = digitalPinToTimer(pin);
        uint8_t bit = digitalPinToBitMask(pin);
```

```
	uint8_t port = digitalPinToPort(pin);
	volatile uint8_t *out;

	if (port == NOT_A_PIN) return;

	// If the pin that support PWM output, we need to turn it off
	// before doing a digital write.
	if (timer != NOT_ON_TIMER) turnOffPWM(timer);

	out = portOutputRegister(port);

	if (val == LOW) {
		uint8_t oldSREG = SREG;
	  cli();
		*out &= ~bit;
		SREG = oldSREG;
	} else {
		uint8_t oldSREG = SREG;
	  cli();
		*out |= bit;
		SREG = oldSREG;
	}
}

int digitalRead(uint8_t pin)
{
```

```
        uint8_t timer = digitalPinToTimer(pin);
        uint8_t bit = digitalPinToBitMask(pin);
        uint8_t port = digitalPinToPort(pin);

        if (port == NOT_A_PIN) return LOW;

        // If the pin that support PWM output, we need to turn it off
        // before getting a digital reading.
        if (timer != NOT_ON_TIMER) turnOffPWM(timer);

        if (*portInputRegister(port) & bit) return HIGH;
        return LOW;
}
```

DOS Batch file to program 328p array

```
avrdude -pm328p -carduino -Uflash:w:snakebot_328p.hex -P com3

avrdude -pm328p -carduino -Uflash:w:snakebot_328p.hex -P com4

avrdude -pm328p -carduino -Uflash:w:snakebot_328p.hex -P com7

avrdude -pm328p -carduino -Uflash:w:snakebot_328p.hex -P com8

avrdude -pm328p -carduino -Uflash:w:snakebot_328p.hex -P com29
```

Bibliography

1. *FLEXIBLE ROBOTICS.* **Vyas, Lona and Aquino, D.** 2, pp 187-189, s.l. : BJU INTERNATIONAL , Jan 2011, Vol. 107.

2. *A comprehensive review of single-incision laparoscopic surgery (SILS) and natural orifice transluminal endoscopic surgery (NOTES) techniques for cholecystectomy.* **Chamberlain, RS.** 2009, Journal of gastrointestinal surgery, pp. 13(9):1733-40.

3. *The Development of a Robotic Endoscope.* **Slatkin, Brett, Burdick, Joel and Grundfest, , Warren.** Aug 1995. Intelligent Robots and Systems 95. 'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on. pp. vol.2, no., pp.162-171 vol.2, 5-9 .

4. *Curved rigid laryngoscope: Missing link between direct suspension laryngoscopy and indirect techniques.* **G., Friedrich.** 2009 , European archives of oto-rhino-laryngology., pp. 10;266(10):1583-8. .

5. *Design and kinematic modeling of constant curvature continuum robots: A review.* . **RJ, Webster.** 2010, The International journal of robotics research., pp. ;29(13):1661-83. .

6. **Paul, C., Valero-Cuevas, F.J. and Lipson, H.** Design and control of tensegrity robots for locomotion. *Robotics, IEEE Transactions on.* 206, Vols. vol.22, , no.5, pp.944-957.

7. **Rieffel, John, Valero-Cuevas, Francisco and Lipson, Hod.** Morphological communication: exploiting coupled dynamics in a complex mechanical structure to achieve locomotion. *J. R. Soc. Interface.* April 6, 2010, Vol. 45, 613-621.

8. *Compact and flexible raster scanning multiphoton endoscope capable of imaging unstained tissue. PNAS* . **DR., Rivera.** 2011 , Proceedings of the National Academy of Sciences, pp. 10-25;108(43):17598-603.

9. **Wada, Mitsuo.** *Flexibly foldable arm. , patent 4685349* Aug 11, 1987.

10. **Seufert, Wolf D.** *Device for carrying observation and/or manipulation instruments. , patent 4054128* Oct 18, 1977.

11. **Belson, Amir.** *Steerable endoscope and improved method of insertion. , patent 8062212* Nov 22, 2011.

12. **Stokes, Robert W.** *FLEXIBLE VIEWING PROBE FOR ENDOSCOPIC USE. , patent 3190286* Jun 22, 1965.

13. *A survey of snake-inspired robot designs.* **James K Hopkins and Brent W Spranklin and Satyandra, K.Gupta.** 2009, Bioinspiration & Biomimetics., p. 4(2):021001. .

14. *Development of a helical climbing modular snake robot.* **Polchankajorn, P. and Maneewarn, T.** pp.197-202, 9-13 May 2011. Robotics and Automation (ICRA), 2011 IEEE International Conference on , . doi: 10.1109/ICRA.2011.5979894.

15. *An experimental hyper redundant serpentine robot.* **Maity, A., Majumder, S. and Ghosh, S.** s.l. : Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on, pp.3180-3185, 10-13 Oct. 2010. doi: 10.1109/ICSMC.2010.5642271.

16. *Multijoint Inspection Robot.* **Asano, K.** 1982 IE-30(3), IEEE Transactions on Industrial Electronics, pp. 277-81.

17. *The `elephant trunk' manipulator, design and implementation.* **Hannan, M.W. and Walker, I.D.** s.l. : Advanced Intelligent Mechatronics, 2001. Proceedings. 2001 IEEE/ASME International Conference on , vol.1, no., pp.14-19 vol.1, 2001. doi: 10.1109/AIM.2001.936423.

18. **Festo.** Bionic Handling Assistant – flexible and compliant movement . [Online] http://www.festo.com/cms/en_corp/9655.htm.

19. *Continuum robots and underactuated grasping.* **Giri, N and Walker, I.** Montreal, Canada. : ASME International Workshop on Underactuated Grasping, 19 August 2010, .

20. *Preliminary Results on the Design of a Robotic Tentacle End Effector.* **Nemir, David C.** s.l. : American Control Conference, 1989 ,, pp.2374-2376, 21-23 June 1989.

21. **Camran Nezhat, M.D.** *Nezhat's History of Endoscopy: A Historical Analysis of Endoscopy's* .. s.l. : Endo Press , 2011. 3897569167.

22. Society of Laparoendoscopic Surgeons. *Nezhat's History of Endoscopy.* [Online] http://laparoscopy.blogs.com/endoscopyhistory/chapter_06/.

23. **LONG Jean-Alexandre, CINQUIN Philippe (2), TROCCAZ Jocelyne (2),.** Development of the Miniaturised Endoscope Holder LER (Light Endoscope Robot) for Laparoscopic Surgery. *Journal of Endourology.* September 15, , 2007, Vol. Volume: 21 , Issue 8.

24. Da Vinci Robotics. *Da Vinci Robotics.* [Online] http://www.davincisurgery.com/.

25. **Anderson, Victor C.** *tensor arm manipulator. , patent 3497083* 1970.

26. *TENSOR ARM MANIPULATOR DESIGN.* **VC, ANDERSON and RC, HORN.** Issue: 8 Pages: 54-& , s.l. : MECHANICAL ENGINEERING , Published: 1967, Vol. Volume: 89.

27. *Statics and dynamics of continuum robots with general tendon routing and external loading.* . **DC., Rucker.** 2011, IEEE transactions on robotics., pp. ;27(6):1033-44. .

28. *Miniature in vivo robot for laparoendoscopic single-site surgery.* **O, Dolghi.** 2011, Surg Endosc, pp. 10;25(10):3453-8.

29. i-Snake Surgical Robot for Minimally Invasive Surgery. [Online] http://www1.imperial.ac.uk/medicine/research/researchthemes/healthtechnologies/surgicaltechnologies/isnake/.

30. i-Snake®: Surgery Evolved | A film by the Wellcome Trust. *How the Isanke evolved.* [Online] most relevant at time_code 1:12 -> 1:14 and 2:34->2:39. . Https://www.youtube.com/watch?v=5UxuNHb9ehg.

31. *Eichhorn KWG. evaluation of force data with a force/torque sensor during FESS A step towards robot-assisted surgery.* . **Wagner, I, Kunkel, M.E, Eichhorn, K.W.G, Westphal, R, Wahl, F.M, Bootz, F, Tingelhoff, K, Rilk, M.** 2008, HNO, pp. 56(8):789-94.

32. *A flexible, tendon-controlled device for endoscopy.* . . **RH, Sturges.** 1993, The International journal of robotics research, pp. ;12(2):121-31. .

33. *Design of a snake-like manipulator.* **Clement, WT and liaigo, R.M.** s.l. : Robotics and Amonomous Sys, 265-282, Vol. 6.

34. *A path planning algorithm using generalized potential model for hyper-redundant robots with 2-DOF joints.* **Chien-Chou Lin, Jen-Hui Chuang, Cheng-Tieng Hsieh.** 2011, International Journal of Advanced Robotic Systems, pp. 06;8(2):49-58.

35. **Aldrich, J.B., Skelton, R.E. and Kreutz-Delgado, K.** Control synthesis for a class of light and agile robotic tensegrity structures. *American Control Conference, 2003. Proceedings of the 2003.* 2003, Vol. vol.6, pp. 5245- 5251.

36. **Research, BCC.** Worldwide Endoscopy Market Expected to Grow to USD 33.7 Billion by 2016. *Worldwide Endoscopy Market Expected to Grow to USD 33.7 Billion by 2016.* [Online] 2011. http://www.wallstreetnewshour.com/NewsStory.aspx?ID=978633.

37. *Design of an integrated master/slave robotic system for minimally invasive surgery.* **Li J, Zhou N, Wang S, Gao Y, Liu D.** 2011, The International Journal of Medical Robotics and Computer Assisted Surgery, pp. pre-hard-copy website.

38. *ENDOSCOPIC INSTRUMENTATION: EVOLUTION, PHYSICAL PRINCIPLES AND CLINICAL ASPECTS.* **RA., Miller.** 1986 , British Medical Bulletin., pp. January 01;42(3):223-5.

39. NeoGuide systems ready with a new strategy, $43M . . *Neoguide i.* [Online] Http://www.bizjournals.com.libproxy.lib.unc.edu/sanjose/stories/2008/01/14/story7.html.

40. noscar o. *noscar o.* [Online] Http://www.noscar.org/.

41. **Low, George M.** *FLEXIBLE/RIGIDIFIABLE CABLE ASSEMBLY. patent 3625084* Dec 7, 1971.

42. **Incorporated, L Lomb.** *BIDIRECTIONALLY FLEXIBLE SEGMENTED TUBE. , patent 3071161* 1963.

43. **Lee, Woojin.** *Robotically controlled surgical instruments. , patent 7699835* Apr 20, 2010.

44. **Mori, Toshiyuki.** *BEND ABLE TUBE OF AN ENDOSCOPE. , patent 3739770* Jun 19, 1973.

45. **Ohline, Robert M.** *Tendon-driven endoscope and methods of insertion. , patent 6858005* Feb 22, 2005.

46. **SHELDON, G. J.** *FLEXIBLE TUBE STRUCTURES. , patent 3060972* Oct 30, 1962.

47. **Swinehart, Charles.** *Non-metallic, multi-strand control cable for steerable instruments. , patent 8083879* Dec 27, 2011.

48. **Wentz, John D.** *Flexible positioning appendage. , patent 5297443* Mar 29, 1994.

49. **Rogers, Theodore W.** *Fixture for shape-sensing optical fiber in a kinematic chain. , patent 7815376* Oct 19, 2010. fiber optic encoder