

Tunneling Robot

A Major Qualifying Project
Submitted to the faculty of
WORCESTER POLYTECHNIC INSTITUTE
In partial fulfillment of the requirements for the
Degree of Bachelor of Science

May 13, 2020

Submitted By:

Thomas Gibbia
tjgibbia@wpi.edu
Spencer Howes
sghowes@wpi.edu
Theodorus Lundblade
trlundblade@wpi.edu

Tomás Ringer-Silva
tfringersilva@wpi.edu
Benjamin Stevens
bgstevens@wpi.edu
Christian Tweed
cdtweed@wpi.edu

Project Advisors:

Professor Michael A. Gennert
Professor Pradeep Radhakrishnan
Professor Jerome Schaufeld



WPI

This report represents the work of WPI undergraduate students submitted to the faculty as evidence of completion of a degree requirement. WPI routinely publishes these reports on its website without editorial or peer review. For more information about the projects program at WPI, please see <http://www.wpi.edu/academics/undergraduates/project-learning.html>

Table of Contents

Table of Figures	iv
List of Tables	vii
Authorship Page.....	viii
Acknowledgements.....	xi
Abstract	xii
Executive Summary.....	1
Introduction.....	2
Stakeholders and Scope.....	2
Design Goals:	2
Stakeholder Introduction and Background:	2
Background.....	3
Applications.....	4
Oil Industry.....	4
Giant Tunnel Boring Machines	4
Underground Utilities	5
Technology:	7
Drill Rod and Bent Sub	7
Mud Motor	7
Jetting:.....	8
Rotary Steerable System (RSS).....	9
Drawbacks and Potential Issues	9
Principles of Boring Geology	10
Types of Soil.....	11
Obstacles within Soil.....	11
Boring Methods	14
Overarching Methods of Boring.....	14
Boring Heads.....	15
Forward Drive Methods	18
Steering Methods	18
Methodology.....	19
Systems	24
Boring.....	24

Initial Design	24
Fabrication	24
Electrical.....	25
Sensors.....	27
Assessment.....	27
Next Steps	28
Extension.....	29
Initial Design	29
Fabrication	29
Electrical.....	30
Assessment.....	34
Next Steps	34
Steering.....	39
Initial Design	39
Fabrication	41
Electrical.....	42
Assessment.....	44
Next Steps	44
Anchoring.....	49
Initial Design	49
Fabrication	49
Electrical.....	49
Next Steps	50
Chassis.....	51
Initial Design	51
Fabrication	51
Electrical Mounting.....	52
Sensors.....	53
Next Steps	55
Power Delivery	55
Software.....	56
Architecture	56
Frontend	57
Controls	57
Status Visualization.....	58

Path Visualization	60
Next Steps	61
Server.....	62
Next Steps	64
Pathing	65
Implementation.....	65
Next Steps	73
Data Layer	74
Downstream Communications.....	74
Upstream Communications	75
Error Handling.....	77
Next Steps	77
Embedded System.....	78
Communication.....	78
Conclusions.....	78
Bibliography.....	80
Appendix A – Videos	84
Appendix B – Software Architecture.....	85
Appendix C – Software Usage.....	88
Source Repository	88
Dependencies.....	88
Frontend/JavaScript.....	88
Starting the Software	88
Appendix D – Methodology Chart.....	89
Appendix C – How to Use Software.....	104
Appendix D – Videos of Operations	105

Table of Figures

Figure 1 - A diagram of Eastman's tilting drill from the May 1934 issue of Science Direct magazine. [1]	3
Figure 2 - A curved section of the underground tunnel dug under Seattle. [37]	4
Figure 3 - A schematic showing the primary features of Bertha [38]	5
Figure 4 - A directional drilling machine in action RMPS Solutions [39]	6
Figure 5 - An outline of the directional drilling process for underground utilities [40]	6
Figure 6 – A fixed angle bent sub [59]	7
Figure 7 - An illustration of a mud motor in a directional boring machine. [41]	8
Figure 8 - An outline of the jetting process [42]	8
Figure 9 - A horizontal boring machine with real-time speed adjustment [43]	9
Figure 10 - A directional drilling blowout [61]	10
Figure 11 - 'The soil classification triangle [4, pp. 213–286].	11
Figure 12 - A visualization of extreme karst void development in limestone [8, pp. iii–137]	12
Figure 13 - An example of a borehole log showing the depth of bedrock [8, pp. iii–137].	13
Figure 14 - 'The Life Cycle of Soil [11]	13
Figure 15 - A horizontal percussion boring machine going through a rock [14]	14
Figure 16 - A model of a drilling and reaming process [15, No. 8]	15
Figure 17 - Roller Cone Bits: Tricone and Single Cone [16, pp. 365–395].	15
Figure 18 - Cutterhead on a Robbins TBM for the Mumbai Water Tunnel [18, pp. 125–126].	16
Figure 19 - Five different drag-bit designs. [19]	17
Figure 20 - Three ubiquitous designs of fixed-cutter bits. [16]	17
Figure 21 - An illustration of the shared components of a roller cone bit and a PDC bit together on a hybrid bit [19].	18
Figure 22 - A chart for the methodology of initial decisions. The expanded chart can be found in Appendix D.	20
Figure 23 - The PJRC Teensy 3.6 [46]	21
Figure 24 - Software Architecture: Frontend and Server	22
Figure 25 - Software Architecture: Data Layer and Arduino Script	23
Figure 26 - A render of the boring module without exterior components.	24
Figure 27 - Chiaphua Components Limited PM25R Motor Curves [47]	25
Figure 28 – The boring motor, the CCL PM25R [48]	26
Figure 29 – Shaft speed calculation code	26
Figure 30 – CTR Electronics' Victor SPX [49]	26
Figure 31 - SparkFun Thermocouple Breakout - MAX31855K [50]	27
Figure 32 - Left: the boring speed selection interface. Right: The boring output shaft with magnet and Hall Effect sensor.	28
Figure 33 - A render of the initial extension design with cutaway shell.	29
Figure 34 - A photo of the two extension gears printed from the Form 2 SLA printer.	29

Figure 35 - A render of the final extension module.	30
Figure 36 - A section view of the final extension module.	30
Figure 37 - uxcell DC 12V 335RPM Gear Box Motor with centric output shaft model no. a18030100ux0053 [51].....	31
Figure 38 - Specifications for similar 300RPM motor [51].....	31
Figure 39 - Extension Tachometry Setup	31
Figure 40 - DROK L298 Dual H Bridge Motor Speed Controller [52]	32
Figure 41 – H Bridge control code.....	32
Figure 42 - Extension Limit Switches.....	33
Figure 43 - Extension Motor Safety Logic.....	33
Figure 46 - Limit switch wires under compression.....	38
Figure 48 - Limit Switch Mounting.....	39
Figure 49b - A section view of the steering module.	40
Figure 49c - A section view of the steering module with the dynamic components emphasized.	40
Figure 50 - A series of images showing the manufacture of the aluminum sliders. Clockwise from upper left: CAM of the part, cut stock, a minimill control screen, a finished part, and a milling operation.	41
Figure 51 - uxcell 12V DC 250RPM Gear Motor with eccentric output shaft model no. A16071400ux0622 [53]	42
Figure 52 - Steering Motor Specifications [53]	42
Figure 53 – Vishay 10 turn Potentiometer 534B1103JL [54]	43
Figure 54 - Steering with installed bands.....	43
Figure 55 - Steering angle as a function of slider block progression.....	44
Figure 56 - Two different failure states of current spur gears. Red indicates fracture lines, while blue indicates mode of force.....	45
Figure 57 - An example of the FEA load setup and results.	45
Figure 58 - Stress concentrations on the gear tooth. Note the three primary locations of stress. Force is applied from the left in both images.	46
Figure 59 - A view of stresses in the gear created by isolating stresses above a certain point.	46
Figure 60 - Deflection of the gear tooth by material and tooth count.	47
Figure 61 - Gear safety factors by tooth count and material.....	47
Figure 62 - Left: the current steering gearing on the prototype. Right: the proposed gearing for future iterations.....	48
Figure 63 – An illustration of force components on the steering slider, left, and steering roller, right.	49
Figure 64 - DIMINUS DC 6V Mini Air Pump [55].....	50
Figure 65 – Honeywell Pressure Sensor HSCDANN030PA2A3 [56].....	50
Figure 66 - A render of the chassis for the boring module.	51
Figure 67 - A photo of the original chassis with many of the components in place.	52
Figure 68 – Top view of wiring for Teensy, motor controllers, and power supply	52
Figure 69 – Side view of motor controller mounting assembly with top controller removed	53

Figure 70 – SparkFun’s LSM9DS1 breakout board, aka 9DoF Sensor Stick [57]	53
Figure 71 - The IMU affixed next to the Teensy	54
Figure 72 – Attitude calculation code [58]	54
Figure 73 - Power supply with voltage regulators.....	56
Figure 74 - Early Controls UI Iteration.....	57
Figure 75a - Final Controls UI Iteration.....	58
Figure 1b - Final Command Log Iteration.....	58
Figure 76 - D3 Status Visualization	59
Figure 77 – Example Rendering.....	63
Figure 78 - Database Schema.....	63
Figure 79 - Protocol Diagram for the Server.....	64
Figure 80 - Angle Subdivision where $\theta = 3.5$	66
Figure 81 - Rendering of all paths with 10° turns	67
Figure 82 - Rendering of all paths with 3.5° turns	68
Figure 83 - Yaw and Pitch Testing.....	70
Figure 84- Drawing Sub-curves	71
Figure 85 - Knot formulation [32].....	72
Figure 86 - Catmull-Rom Spline [33]	72
Figure 87- Result of Path Correction and Centripetal Catmull-Rom Spline.....	73
Figure 88 - Controls JSON Message Format.....	74
Figure 89 - Path JSON Message Format.....	75
Figure 90 - Main Module Communication Loop.....	75
Figure 91 - Status JSON Message Format	76
Figure 92 - pathStatus JSON message format.....	77
Figure 93 - Error JSON message format	77

List of Tables

Table 1 - Initial design goals deciding during the planning phase.	2
Table 2 - Summary of trenchless boring methods	10
Table 3 – Pressure loss in a hydraulic line [45].....	35
Table 4 - Table of values for gear calculations and FEA.....	45
Table 5 -_Power Consumption Calculations.....	55

Authorship Page

Section	Author(s)
Abstract	Tomás Ringer-Silva
Executive Summary	Tomás Ringer-Silva/Benjamin Stevens
Introduction	
Stakeholders and Scope	Theo Lundblade
Design Goals:	Theo Lundblade
Stakeholder Introduction	Theo Lundblade
Background	
Applications	Benjamin Stevens
Oil Industry	Benjamin Stevens
Giant Tunnel Boring Machines	Benjamin Stevens
Underground Utilities	Benjamin Stevens
Technology:	Benjamin Stevens
Drill Rod and Bent Sub	Benjamin Stevens
Mud Motor	Benjamin Stevens
Jetting:	Benjamin Stevens
Rotary Steerable System (RSS)	Benjamin Stevens
Drawbacks and Potential Issues	Benjamin Stevens
Principles of Boring Geology	Tomás Ringer-Silva
Types of Soil	Tomás Ringer-Silva
Obstacles within Soil	Tomás Ringer-Silva
Boring Methods	Tomás Ringer-Silva
Overarching Methods of Boring	Tomás Ringer-Silva
Boring Heads	Tomás Ringer-Silva
Forward Drive Methods	Tomás Ringer-Silva
Steering Methods	Tomás Ringer-Silva
Methodology	Christian Tweed/Benjamin Stevens
Systems	
Boring	
Initial Design	Tomás Ringer-Silva
Fabrication	Tomás Ringer-Silva
Electrical	Spencer Howes
Sensors	Spencer Howes
Assessment	Tomás Ringer-Silva
Next Steps	Tomás Ringer-Silva
Extension	
Initial Design	Tomás Ringer-Silva
Fabrication	Tomás Ringer-Silva
Electrical	Spencer Howes
Assessment	Tomás Ringer-Silva

Next Steps	Benjamin Stevens/Tomás Ringer-Silva
Steering	
Initial Design	Tomás Ringer-Silva
Fabrication	Benjamin Stevens
Electrical	Spencer Howes
Assessment	Tomás Ringer-Silva
Next Steps	Tomás Ringer-Silva
Anchoring	
Initial Design	Benjamin Stevens
Fabrication	Benjamin Stevens
Electrical	Spencer Howes
Next Steps	Benjamin Stevens
Chassis	
Initial Design	Tomás Ringer-Silva
Fabrication	Tomás Ringer-Silva
Electrical Mounting	Spencer Howes
Sensors	Spencer Howes
Next Steps	Benjamin Stevens/ Tomás Ringer-Silva
Power Delivery	Spencer Howes
Software	
Architecture	Thomas Gibbia
Frontend	Christian Tweed
Controls	Thomas Gibbia
Status Visualization	Christian Tweed/Thomas Gibbia
Path Visualization	Christian Tweed
Next Steps	Christian Tweed/Thomas Gibbia
Server	Christian Tweed
Next Steps	Christian Tweed
Pathing	Christian Tweed
Implementation	Christian Tweed
Next Steps	Christian Tweed
Data Layer	
Downstream Communications	Thomas Gibbia
Upstream Communications	Thomas Gibbia
Error Handling	Thomas Gibbia
Next Steps	Thomas Gibbia
Embedded System	
Communication	Thomas Gibbia
Conclusions	
Bibliography	Christian Tweed
Appendix A – Videos of Operations	Tomás Ringer-Silva
Appendix B – Software Architecture	Christian Tweed/Thomas Gibbia
Appendix C – Software Usage	
Source Repository	
Dependencies	Christian Tweed/Thomas Gibbia

Frontend/JavaScript	Christian Tweed/Thomas Gibbia
Python	Thomas Gibbia
Starting the Software	Thomas Gibbia
Appendix D – Methodology Chart	Theo Lundblade/ Tomás Ringer-Silva

Acknowledgements

After completing this project, it became clear that certain professors were instrumental to the process. We would like to thank our advisors, Professor Gennert, Professor Radhakrishnan, and Professor Schaufeld. As well as professors Daniello, Stults, and Hera, and James Loiselle, who were not officially assigned to the project, yet took time to provide guidance to us. Seeing as how MQP is the culmination of our academic careers thus far, it also seems fitting to take a moment to reflect on all the people who helped make our time at WPI a valuable learning experience. Thank you to all the professors who took the time to share their love of their subject with us and challenge us in fun and interesting ways. Thank you to all the friends who put in late nights that often turned into early mornings working on homework and projects with us. Most of all thank you to everyone who supported us along the way. We look forwards to taking what we learned in these four years into the future, wherever our paths may lead.

Abstract

Digging tunnels is a necessary part of many construction and infrastructure projects, but most small-scale tunnel-boring methods presently require a trench which can disrupt aspects of daily life on the surface. There are trenchless methods, but they require that the tunnel be completely or almost completely straight. The goal of this project was to design a self-correcting machine capable of producing complex tunnels without the use of a trench. Early stages of design considered many different alternatives for steering, anchoring, and boring. A prototype was constructed as a proof of kinematic concept with full electronic and computational systems onboard. Using the prototype, a report was generated detailing the improvements that would need to be made for a high-budget and marketable iteration of the device because, while the prototype is not capable of travelling underground, it provides proof that the kinematic designs are sound.

Executive Summary

The Modular Omnidirectional Lightweight Earthmover (MOLE) is designed to fill a need for a more adaptable and dexterous method of undergrounding cables. Compared to traditional methods, it disrupts surface conditions far less than conventional trench-based methods. MOLE operates in a manner similar to directional drilling methods currently used in the oil industry as well as other methods of trenchless boring. Boring is accomplished by displacing dirt around the body or including a system to pump the dirt out through the robot as a slurry. Selection of a boring head that is compatible with both the drilling paradigm and the soil conditions is important. Steering methods vary, but all of them work by adjusting the angle of the drilling head relative to the drill rod. What sets mole apart is that it is not mechanically tied to the surface, meaning it will not need a large engine on the surface, and it will be able to steer more effectively because it does not need stiff drill rod behind it, and it's built in obstacle avoidance system.

MOLE works by inch-worming its way through the soil. It is broken into two main halves. It extends the front while it bores, and then retracts the back cyclically in order to advance. Design of the robot was broken into separate subsystems: a boring system, an extension system, a steering system, an anchoring system, and a housing and chassis system. It was difficult to manufacture a boring head, and commercially available ones were out of budget. A model boring head was designed, but due to closures surrounding the SARS-CoV-2 outbreak it could not be manufactured. The extension system uses a custom-made linear actuator that is controlled by a single motor. The steering system is the primary innovation of this robot; it utilizes two pairs of sliding blocks to orient a “frustum” into position, with each pair of sliding blocks driven by a single motor and matched using gears. The anchoring system utilizes inflatable pneumatic tubes pressed against the soil, holding either module in position. All of the various motors and actuators receive power from an onboard power supply in the back module and are intelligently controlled by a Teensy microcontroller. The software side contains a server-based front-end so that an operator can control the MOLE from a standard browser and a pathing system that can generate paths in the complex, obstacle-ridden 3D space of the soil.

The team's prototype is unable to bore through the earth. At the time of writing, every component in the unit is fully mobile but it is unclear how the unit would fair in the ground. Future teams should improve the mechanical durability of the robot, optimize its components, and improve the ease-of-use and effectiveness of the software system.

Introduction

Stakeholders and Scope

Underground utilities have become commonplace, zigzagging through every city and town. Power lines, gas lines, sewer pipes, and many other utilities form a complex web under the soil of our cities. Unfortunately, installing and repairing these underground utilities can be costly, time consuming, and disruptive to nearby citizens and businesses as the conventional process to underground power lines requires large open-air trenches. Pedestrian and vehicular traffic must be diverted. Additionally, preexisting groundwork such as roadways, sidewalks, and landscaping must be ripped up for such trenches. This is both counterproductive and distressing for the landowners and businesses affected. With this project, our team seeks to propose an alternative to digging trenches, thus eliminating many of the challenges associated with burying cables. MOLE or Modular Omnidirectional Lightweight Earthmover is a small maneuverable tethered tunneling robot which can be deployed from a medium-sized truck and operated by 1-2 technicians. In this initial proposal and research summary, we will cover specification goals, estimates of the value propositions focused on the major potential stakeholders, and research of various options for design.

Design Goals:

Bore diameter:	4"
Turning circle:	10'
Weight:	Under 50 lbs
Minimum bore length:	100'
Digging material:	Dirt

Table 1 - Initial design goals decided during the planning phase.

Stakeholder Introduction and Background:

We have three target stakeholders for this boring product. First: power companies seeking to underground electrical power lines who are constrained by preexisting construction, natural obstacles, or space. Second: internet service providers (ISPs) seeking to install underground Ethernet, cable, or fiber lines again around or under already existing features, such as concrete slabs. Third: construction equipment rental companies such as Home Depot or similar who rent to homeowners and other small contractors. Conventional data and power line undergrounding costs \$4-\$12 per foot depending on site limitations and generally totals \$600-\$2100 per lot (based on 100-foot trench length). However much of that cost is tied up in the initial stage of hiring people and equipment.

MOLE can allow utility providers to navigate cabling under and around surfaces or obstacles that would otherwise have to be removed. One example of this is digging a tunnel under a homeowner's driveway to provide power instead of digging a trench which requires more repairs and is more disruptive to the surface conditions. Similarly, in dense city centers, where large excavating equipment is challenging to operate and causes significant disruption to traffic, MOLE can easily be deployed from a truck, causing minimal disturbance to surrounding people. Additionally, with a small-bore size of 4 inches MOLE will also cause far less environmental impact than trenches that are at least 2 feet deep and similar width. All of this combines into an effective cost savings on projects. As the number of onsite personnel is drastically reduced, heavy equipment

is unneeded, trench shielding and safety measures are unneeded, and public opinion is more favorable. For ISPs MOLE can simplify cable installation and reduce required labor while keeping homeowners and thus target customers happier. Finally for equipment rental companies MOLE can potentially enable otherwise challenging DIY projects for home improvement enthusiasts and give small contractors far more options for tracking difficult construction situations, one example being providing foreman trailers with power and internet access on large construction sites with smaller delays.

Background

Directional drilling was first implemented in 1934, when a well in Conroe Texas exploded and erupted into flames, filling a lake with flaming oil. John Eastman devised a solution where they would drill a second well from a safe distance, and then steer it towards the first well to drain the oil that had not yet ignited. Eastman's process involved cutting a straight hole, and then inserting a vertical bit with a universal joint at the end and an angled guide so that the drill cut at an angle (figure 1)[1]. The angled hole was monitored by removing the bit, dropping a camera and a level into the hole taking a picture and then replacing the bit in combination with a glass bottle full of acid which was lowered into the hole, and then when it was removed the angle of the acid-etched line was examined. From this, the position of the drill bit could be calculated. The technology Eastman devised to solve this crisis helped pave the way for advancements in both the oil industry and other areas. [1]

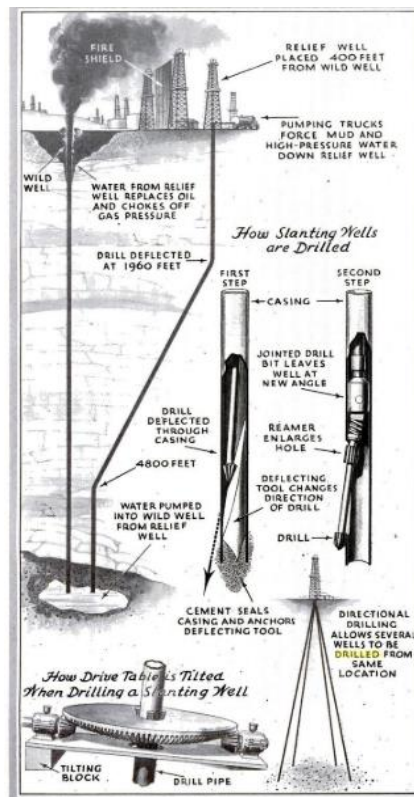


Figure 2 - A diagram of Eastman's tilting drill from the May 1934 issue of Science Direct magazine.[1]

Applications

Directional drilling is an integral part in the construction of modern infrastructure. It is commonly used in everything from transportation to oil drilling to laying utility cables. The benefits of this technology include access to remote areas, ability to drill a horizontal hole at a set depth, accessing areas underground without disrupting surface conditions, and even drilling straighter downward holes.

Oil Industry

The oil industry is one of the biggest users of directional drilling. Since petroleum resources are generally stratified, a single drill can be used far more effectively, because a single drill can be directed horizontally to follow the pay zone. By guiding the drill along veins of oil resources as opposed to stopping at one point in the well, oil can be removed far more efficiently.

Giant Tunnel Boring Machines



Figure 3 - A curved section of the underground tunnel dug under Seattle. [37]

The same technology scaled up was used in the installation of an auto tunnel directly under Seattle (figure 2). This was completed by a tunnel boring machine (TBM) named Bertha [37]. Some of the issues that came up were a result of the non-uniform geology of the region. Bertha had to drill through eight types of soil, ten different geologic zones, and even below the waterline. Bertha worked by drilling a section of tunnel with a modular cutting head, changing out different inserts to match the current geological conditions. The ground-up rock was removed by a screw mechanism, and then to a conveyer where it could be trucked out (figure 3). The bit was driven by a ring of hydraulically powered linear actuators. When the throw of these was reached, it installed in prefabricated tunnel sections behind it.

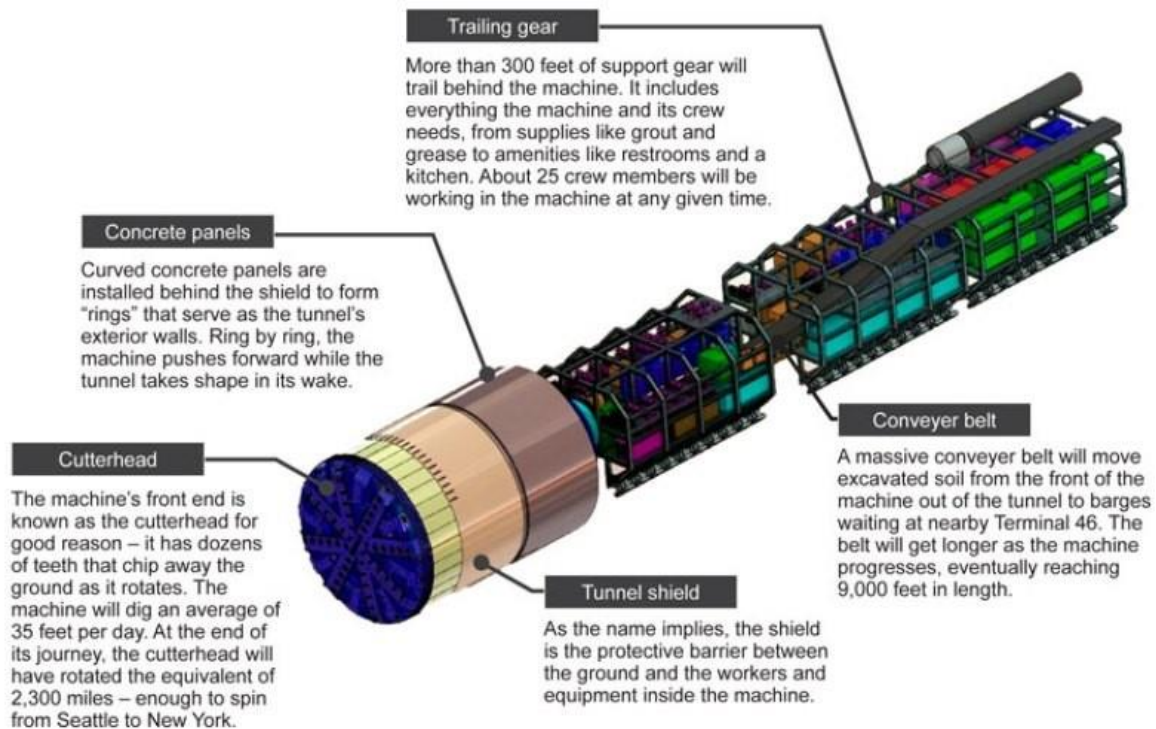


Figure 4 - A schematic showing the primary features of Bertha [38]

Underground Utilities

Directional drilling (figures 4 and 5) is also useful in laying underground utilities such as water and sewage lines and fiber optic cables. Traditionally, underground utilities are installed by digging a trench, laying the lines, and reburying them. This can pose issues, especially if the lines need to go under a road or sidewalk where stopping the flow of traffic would be not only expensive, but possibly damaging to public relations. Directional drilling provides an alternative that causes minimal surface disturbance. The basic process of directional drilling begins with drilling a small pilot hole through the ground with a spade shaped bit. This allows the operator to steer the hole, by aligning the bit with the desired direction, and then forcing it in that direction without spinning the drill rod. Once the hole is established, the drill rod is rotated again, and normal drilling is resumed. Since there is an exit hole, there are more techniques available to the operator than blind holes like in oil wells. Once the hole breaks the surface, a reamer is attached to the end and it is pulled back through to enlarge the hole [44]. After the hole is enlarged, a reamer is dragged through a second time, but this time the payload is pulled behind it.



Figure 5 - A directional drilling machine in action RMPS Solutions [39]

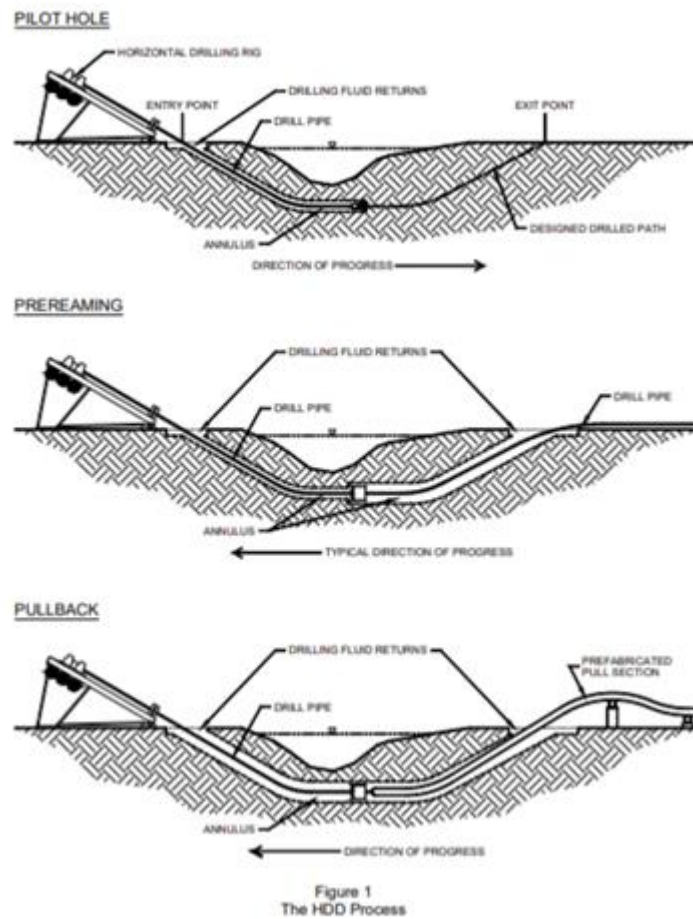


Figure 1
The HDD Process

Figure 6 - An outline of the directional drilling process for underground utilities [40]

Technology:

There are many existing techniques in the field of directional drilling and often multiple are used on the same job depending on the geological conditions, like in the case of Bertha in Seattle, where the conditions changed multiple times over the course of the hole. For example, in harder soil or rock, the drill rod cannot be used to transmit enough torque, so a downhole motor must be used.

Drill Rod and Bent Sub

Drill rod is used to connect the bit to the surface. It is made up of hollow threaded sections that can be combined to required length. They are specially designed to be able to transmit torque along their length, yet still be flexible enough to bend and follow a curved hole. They are hollow so that hydraulic fluid and electrical wires can be run through the hole. At the end of the drill rod is a section called a bent sub. The bent sub is critical to steering the hole. It is a slightly bent piece of drill rod that is either fixed at one angle or adjustable and used to control the direction of the hole. By adjusting the angle of the entire drill string, the bent sub will point in a different direction, the hole is advanced in that direction by removing material without changing the angle of the bent sub (figure 6).



Figure 7 – A fixed angle bent sub [59].

Mud Motor

Mud motors are useful in applications such as hard rock where torque cannot be transmitted down the drill pipe. A mud motor is a hydraulic device capable of producing incredibly high torques. It consists of an eccentrically rotating helically lobed shaft, in a housing with helical channels in a quantity one greater than the lobes. The eccentric helix rotates and changes where the hydraulic fluid flows in causing it to rotate both radially and eccentrically. This is closely related to the way a radial aircraft engine works.

The mud motor is connected to the drill stem at one end, and the bent sub at the other. It is either connected to the bit by way of a universal joint, and placed above the bent sub, or placed below the bent sub as seen in figure 7, and connected by a fixed shaft. The angle of the drilling head is guided by controlling the angle of the rod from the top, thus changing where the bent sub is pointing.

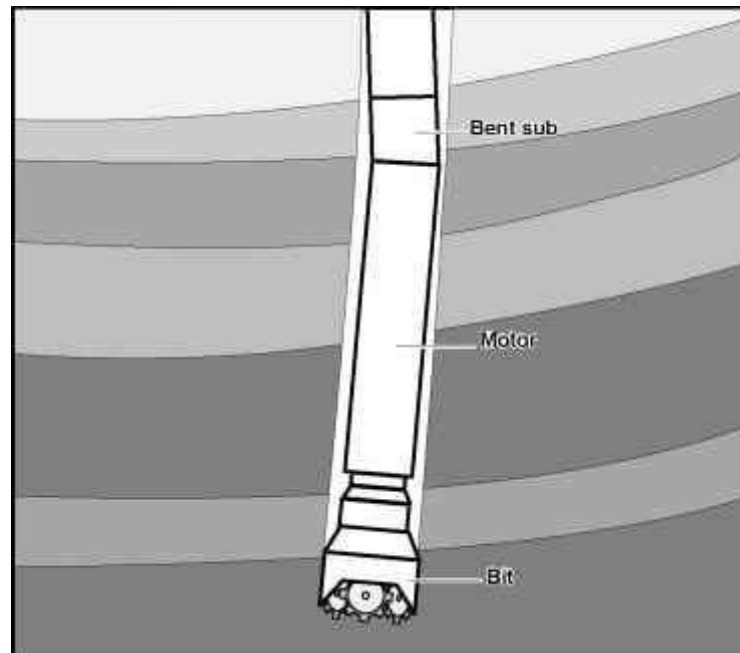


Figure 8 - An illustration of a mud motor in a directional boring machine. [41]

Jetting:

For softer earth, a jetting process can be used. Jetting works similarly to a downhole motor. The bit is directed by a bent sub, and then a jet of hydraulic fluid is forced out a nozzle in that direction. The bit is forced down without rotating, and it moves through the soil softened by the jet. Once the bit trajectory has been steered the desired amount, normal rotary drilling is resumed. Since the bit is driven from the top, and there is a bent stem, this means that the bit will be rotating eccentrically, and will therefore need to be able to cut with its side.

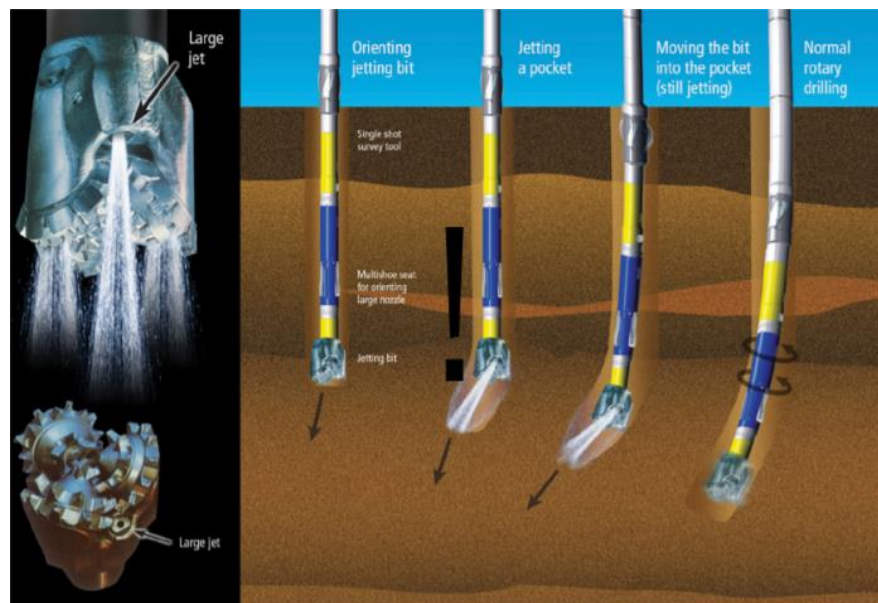


Figure 9 - An outline of the jetting process [42]

Rotary Steerable System (RSS)

A newer technology made possible by modern computing elements, RSS, allows for real time speed adjustment. It uses the same downhole mud motor but is able to bend the shaft in real time to steer the bit. This allows for more complex hole geometry, and cleaner curves to be attained, because the hole trajectory can be adjusted on the fly, and the desired angle can be slowly approached, rather than being limited to the angle of the bent sub. The three main parts are a nonrotating sleeve with actuator, a stabilizer, and the cutting end. The stabilizer acts as a fulcrum, allowing the rest of the shaft to be bent relative to the hole. The drive shaft runs through the non-rotating assembly, where linear actuators bend the shaft and steer the bit.

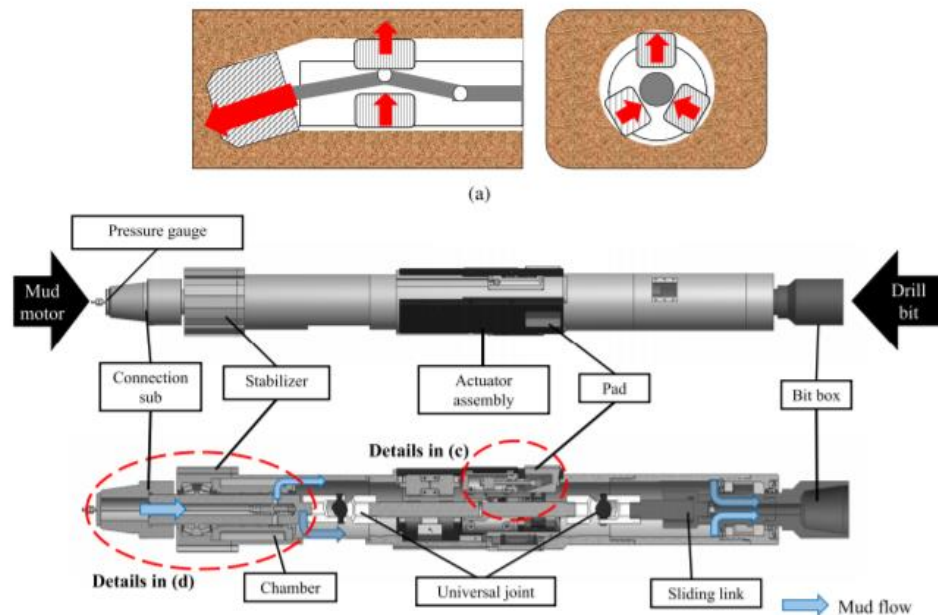


Figure 10 - A horizontal boring machine with real-time speed adjustment [43]

Drawbacks and Potential Issues

The main safety issue that has come up in horizontal directional drilling is blowouts. A blowout is when the bit breaks through the surface after wandering off course. It is much easier to break through the surface while drilling horizontally than it is in a vertical well. Also, because directional drilling is being applied to fields such as laying utilities, where the hole is much closer to the surface, there is an increased risk of blowouts.



Figure 11 - A directional drilling blowout [61]

Trenchless Construction Method	Pilot Equipment Type	Ream Equipment Type	Profile Type	Water Use	Bentonite Use	Loss of Circulation Potential	Inadvertent Return Potential
Conventional Auger Bore	None	Screw auger, motor drive	Flat	Used to cool cutter for bores through rock. Low volume, typically 1-2 gallons a minute in rock	Bentonite slurry applied to casing during ream to prevent binding.	None, returns enclosed by casing	None, returns enclosed by casing
Guided Auger Bore	Small diameter spoon cutter, machine or hydraulically driven, or self-propelled drill unit	Screw auger, motor drive	Flat	Used to cool cutter for bores through rock. Low volume, typically 1-2 gallons a minute in rock.	Bentonite slurry applied to casing during ream to prevent binding.	None to low varying by pilot type. None during ream, returns enclosed by casing	None to low varying by pilot type. None during ream, returns enclosed by casing
Cradle Bore	None, or small diameter spoon, machine or hydraulically driven	Screw auger, motor drive	Flat	Used to cool cutter for bores through rock. Low volume, typically 1-2 gallons a minute in rock.	Bentonite slurry applied to casing during ream to prevent binding.	None to low varying by pilot type. None during ream, returns enclosed by casing	None to low varying by pilot type. None during ream, returns enclosed by casing
Jack or Hammer Bore	Pneumatic or hydraulic driven hammer pilot	Pneumatic or hydraulic driven hammer hole enlargement	Flat	None	Bentonite slurry applied onto casing during enlargement to prevent binding.	None, hole enlarged by crushing tool at face and compression into surrounding subsurface	None, hole enlarged by crushing tool at face and compression into surrounding subsurface
Guided Bore	Small to medium size self-propelled drilling unit, spoon, jet head, or mud motor driver cutter	Open face rotated cutting tool	Flat to minor arc; surface or pit launch to pit or trench exit	Low to high pressure water, or water/bentonite slurry pumped through stem pipe to pilot head or reaming tool	Pressurized water, or water/bentonite slurry pumped through stem pipe to pilot head or reaming tool	None to high varying by surrounding soils or geologic strata, open annulus	None to high varying by surrounding soils or geologic strata, open annulus
FlexBor	Pneumatic or hydraulic driven hammer pilot; Small to medium size self-propelled drilling unit, trailer mounted drill unit, spoon, jet head, or mud motor drive cutter	High pressure air with low pressure water injection to open face injection cutter; enclosed drive stem, motor or drill unit drive	Flat, minor arc, or surface to surface radius	Low pressure water injection into high pressure air stream pumped through pilot casing	May or may not be used during pilot phase. Not used during reaming phase.	None to high varying by pilot type. None during ream, cuttings blown through enclosed annulus, no circulated fluids	None to high varying by pilot type. None during ream, cuttings blown through enclosed annulus, no circulated fluids
Horizontal Direction Drill	Medium self-propelled, or trailer mounted drilling unit, jet head, or mud motor drive cutter	Open face rotated cutting tool, drill unit drive	Surface to surface radius	Pressurized water, or water/bentonite slurry pumped through stem pipe to pilot head or reaming tool	Pressurized water, or water/bentonite slurry pumped through stem pipe to pilot head or reaming tool	None to high varying by surrounding surface soils or geologic strata	None to high varying by surrounding surface soils or geologic strata

Table 2 - Summary of trenchless boring methods

Principles of Boring Geology

The diverse history of horizontal drilling has led to a wide variety of principles and systems. In order to properly choose which principles and systems to apply to MOLE, a comprehensive understanding of the science of boring is necessary.

Types of Soil

In order to determine the range of difficulty of the materials that a boring machine must bore through, it is useful to have a systematic determination scheme for soil types. There have been multiple soil classification systems over time. Currently, the Unified Soil Classification System has been accepted for tunnel engineering because it can be used to predict soil behavior [2, pp. 4507–4518]. The system breaks soil down by size, liquid limit, and plasticity; there are eight groups of coarse-grained soil, six groups of fine-grained soil, and only one group of highly organic soils such as peat [3]. These classifications can be used to predict the compaction characteristics and stability of the soil, useful for estimating the ease of drilling and locomotion. Another useful system for classification of soil is using the “soil classification triangle” which describes soil by the concentrations of sand, silt, and clay (themselves defined by particle size) [4, pp. 213–286]. The soil classification triangle can be seen in Figure 12. The primary value of these soil classifications is estimating the friction and viscosity present and different stages of boring [5, pp. 360–371].

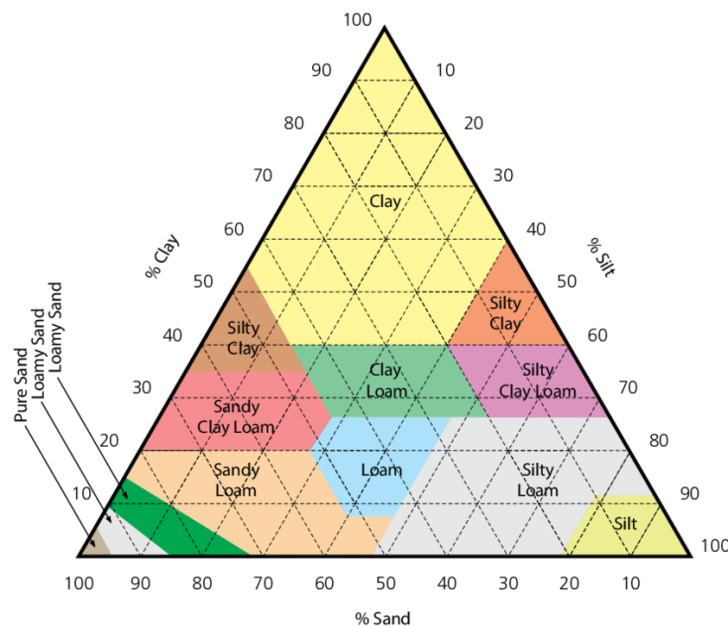


Figure 12 - The soil classification triangle [4, pp. 213–286].

Obstacles within Soil

The ground that boring machines travel through contains much more than simply soil. There are obstacles that the machine will meet that it must have ways to contend with, either by avoiding them or going through them.

Utility Lines

There are dozens of types of utility networks running underneath urban and rural areas, ranging from electrical lines and small residential plumbing pipes to fire safety mains and gas lines. Whichever type they are, avoiding them is a high priority, and there are multiple ways to do this. The first is through reference documentation provided by utility companies with locations of existing lines. The second is through subsurface imaging of potential obstacles. When the two strategies are combined, it is known as subsurface utility engineering. [6]

Voids

Below the ground's surface are spaces that contain neither soil nor rock. These are known as voids. Most voids are formed from a phenomenon known as Karst, in which some materials such as limestone or gypsum are dissolved by acidic rainwater and develop fissures. [7] While these voids pose a danger to surface operations in the form of sinkholes, they also present a risk to a boring machine. If a boring unit falls into a karst void, it could prove irretrievable. In addition, voids frequently contain water or even methane, so should be avoided. Luckily, however, voids do not occur in every material, rarely occur in soil, and can be detected using ground imaging.

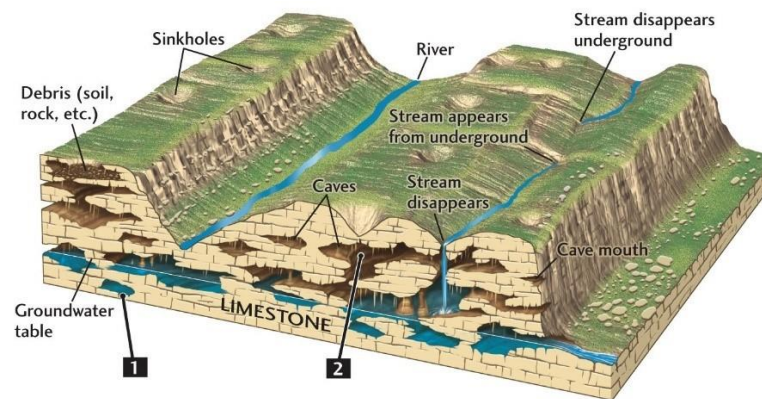


Figure 13 - A visualization of extreme karst void development in limestone [8, pp. iii–137] .

Rocks and Drillability

Boulders can either be drilled through or avoided if they are detected. Rocks are rated for drillability on a five-point scale from fast to slow based on their hardness, texture/density, fracture pattern, and structure [9, pp. 287–292]. The drilling operation can be described in terms of speed of drilling if the machine properties are known. Importantly, the drillability is not simply related to the brittleness of the rock, so specialized systems of characterization must be employed [10, pp. 406–414].

Bedrock

Underneath the layer of soil lies a layer of hard rock. Figure 15 shows the development process of soil [11]. As can be seen, the soil is nothing more than a layer of carbon-rich eroded rock above a layer of uneroded rock, so at a certain depth all soil layers—known as regolith—will give way to rock layers. The top of the bedrock layer, or the rockhead, is in most cases the maximum boring depth and so should be known. It can be found by imaging, core sampling, or referencing known bedrock depths [12].

Boring Methods

Overarching Methods of Boring

Boring can be done in many ways depending on the use case and type of hole desired. The following are some methods that are currently used in industry to create horizontal boreholes.

Drill and Compress

A drill and compress method breaks the soil using a cutter of some sort and then compresses the soil into the walls of the borehole. This method works best in soft soils and is less effective in sand or rock. The benefit of a drill and compress system is that there is no need for a removal system for the muck generated in the boring process. The disadvantage is that its use cases are limited, and any incompressible region will be impassable to the boring unit.

Drill and Displace

A drill and displace system uses a cutting head to break the boring material and then uses a muck removal system to translate the bored material back to a collection system or to the entrance of the hole. The systems do not rely on the compressibility of the drilling substrate but do require a much more complex machine and provide less room for a cable payload.

Pulverize

Certain systems use percussive methods of traveling through the ground. One such method is the pneumatic boring cylinder created in the 60s in the USSR, comprised of a containment rod and a hammering mechanism that both forced the rod through the ground and broke and displaced all dirt ahead of the machine; even back in 1964, the machines could travel horizontally 20 meters with a deflection of only 15-20 centimeters [13, pp. 165–168]. The system relies on a two-fold system of propelling the anvil and then the cylinder forward on separate strikes of the pneumatic hammer, as shown in Figure 16.

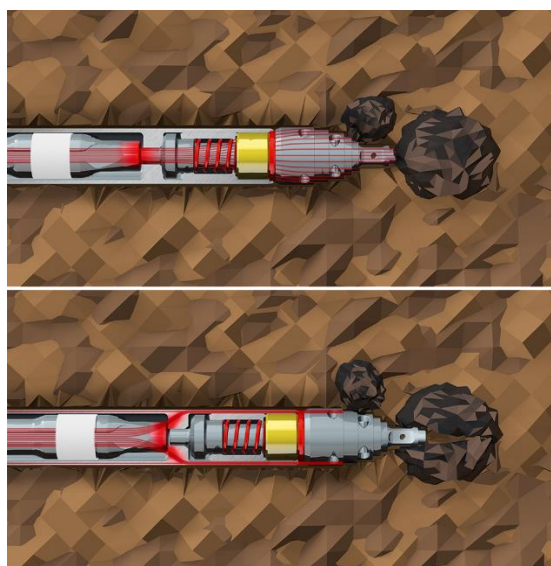


Figure 16 - A horizontal percussion boring machine going through a rock [14] .

Drill and ream

A drill and ream system uses progressively larger units to expand a hole that has already been drilled. In some systems, the reamer follows the drill as it is boring. In other systems, the reamer is pulled through on a cable through a borehole that has already been created by the tunneling machine. The drilling and reaming process is shown in Figure 17.

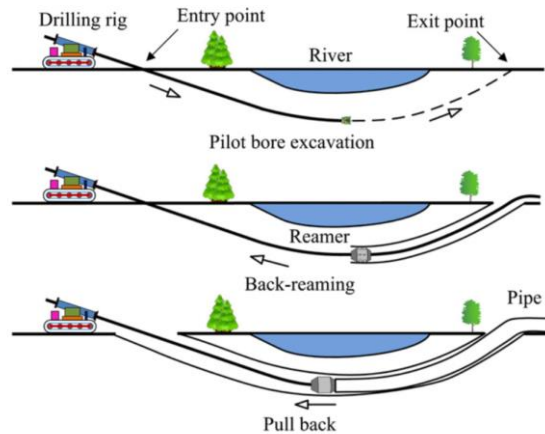


Figure 17 - A model of a drilling and reaming process [15, No. 8].

Boring Heads

Any boring method needs a boring head to create the hole that the rest of the unit continues through. The following are types of boring heads used in different boring industries. Not all will prove appropriate for the MOLE, but it is good to have an idea of the existing methods prior to selection of a specific bit. Some of the bits are used primarily in the petroleum industry, so they specialize in rock drilling; while the MOLE may not be designed for going through rock, it is possible that future iterations of the design will be so an accounting of rock-specific methods is still useful.

Roller Bit

Roller bits are designed for hard rock, crushing and fracturing the material as the cones roll across it. The torque is supplied only to the chassis of the bit, with the rollers being driven only with their interaction at the surface of the borehole [16, pp. 365–395]. Because the method of boring is fracture, roller bits are nearly ineffective for softer materials. Figure 18 shows single and triple roller cone bits.

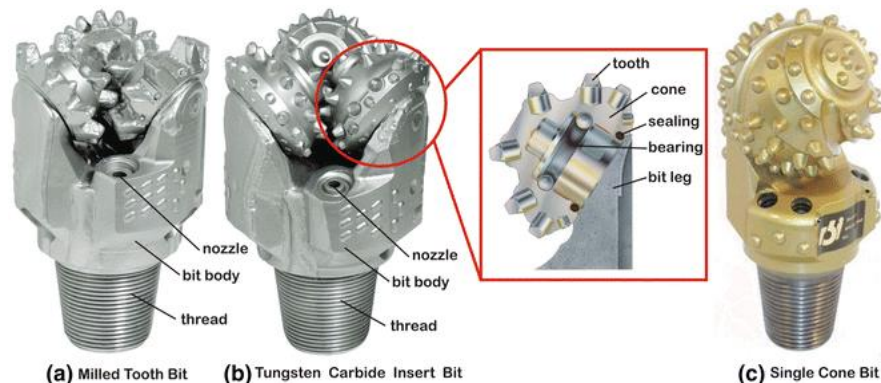


Figure 18 - Roller Cone Bits: Tricone and Single Cone [16, pp. 365–395].

Auger

An auger is a screw-shaped device used for transporting material linearly using a rotational motion. Current auger-based boring methods utilize an initial cutting tool followed by a chain of augers to remove material from the hole [17]. Augers themselves are not used as cutting heads unless the length of the desired borehole is less than the length of the auger. Augers cannot be used in hard rock.

TBM Cutterheads

The massive cutterheads on hard-rock TBMs are flat in design with small circular cutters arranged tangentially and radially, protruding several inches from the face of the cutter; these cutters can be exchanged from behind the cutterhead, so they are designed to be semi-disposable [18, pp. 125–126]. TBM cutterheads are designed for hard rock and would need to be heavily modified to bore through softer materials. Figure 19 shows the cutterhead from a hard-rock TBM.



Figure 19 - Cutterhead on a Robbins TBM for the Mumbai Water Tunnel [18, pp. 125–126].

Drag Bits

There are several designs of fixed cutter bits, each slightly different from the others. Drag bits are the simplest boring head design and have largely gone out of use for oil well boring due to their soft-soil restrictions. It is very effective at breaking soils of various types and is used in conjunction with a high-volumetric-flow hydraulic system for clearing excess material.



Figure 20 - Five different drag-bit designs. [19]

Modern Fixed Cutter Bits

Modern fixed cutter bits come in three primary types. The polycrystalline diamond cutter (PDC) bit uses diamond cutters in conjunction with normal soft-material blades to cut through soft and medium rock, while the impregnated bit and natural diamond bit both use diamond cutters on their surface to drill through hard and abrasive rock [16, pp. 365–395] .

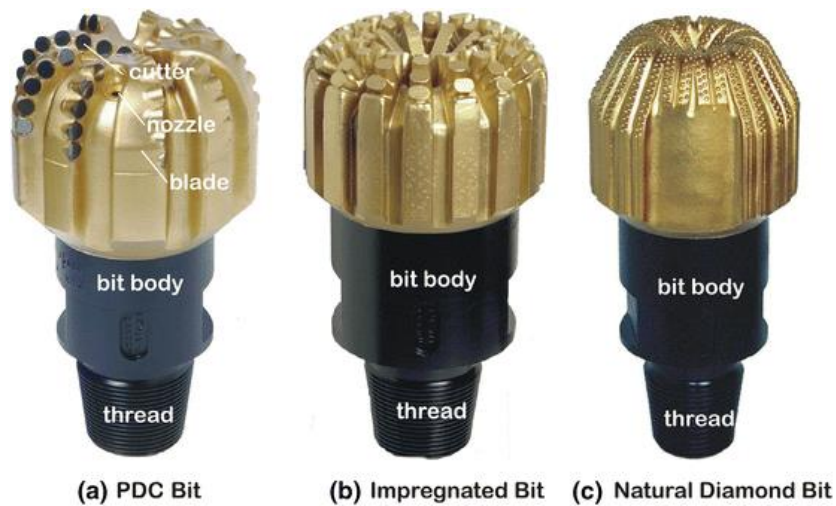


Figure 21 - Three ubiquitous designs of fixed-cutter bits. [16]

Hybrid bits

Hybrid bits are fixed cutter bits combined with roller cone bits. They offer excellent performance in both fracturing and cutting and easily accommodate directional operations. They can be used in soft, medium, hard, and abrasive rock [16, pp. 365–395].

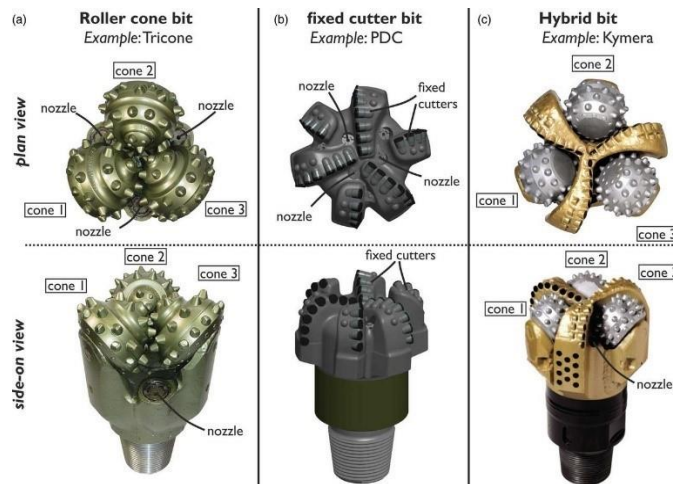


Figure 22 - An illustration of the shared components of a roller cone bit and a PDC bit together on a hybrid bit [19].

Forward Drive Methods

Locomotion Source at Surface

Most methods currently on the market use a source of locomotion at the entrance to the borehole. These are driven via a large pneumatic jack or other drive system and are connected the boring unit by a semi-rigid conduit. This method provides more reliable thrust but is less mobile underground.

Locomotion Source at Unit

A locomotion source at the unit will provide more mobility but less thrust. Current methods primary use and auger-propelled unit, but this requires a stiff shaft back to the borehole entrance. A truly self-propelled boring unit does not exist.

Steering Methods

One of the three steering methods below will work for the MOLE. These methods are not specific technologies; rather, they are umbrella methods than encapsulate many different systems. Selecting one of the three systems below will lead to a further investigation of systems.

Whipstock

The whipstock is the earliest method for steering a horizontal drill. It is shown in Figure 2 and consists of a single angled plate that deflects the drill in a specific direction. This method will not work for the MOLE as the drill is not multidirectional; rather than steer, it can simply turn once.

Push-the-Bit

The push-the-bit method uses some sort of system to deflect the front of the drill away from one of the sides of the walls. It provides a very large turning radius but relies on a simple and reliable system. This method will only work is a hard-walled boring substance.

Point-the-Bit

The point-the-bit method uses an internal curved shaft and eccentric rotary housing or a lever mechanism to point the drill bit in a specific direction. This method is notable for its freedom of movement of the drill bit but must be accompanied by a mechanism to turn the machine in the middle in order to get the small turning radii that it is capable of.

Angle-the-Machine

An angle-the-machine method makes use of a sub as shown in Figure 7 to angle the machine midshaft. This allows the drill bit to use its edges against the borehole with leverage from the back of the shaft.

Methodology

The overall goal was to design a system capable of laying cables underground without any mechanical tethering to the surface. This will require it to bore, steer, and advance. There are multiple methods of accomplishing this that can be split into two paradigms. One is a system for continuous advancement, the other is an inchworm type motion where it anchors and extends and bores. The inch worming method makes a lot more sense, because regardless of how it is accomplished, there will need to be an anchoring system, but for continuous advancement, it will need to both anchor and advance simultaneously which will be a complex system to design. This will also limit the ability to anchor with mechanical interference. On the other hand, anchoring and extending in steps allows for a more robust system to be developed. It is also not time sensitive, so anchoring methods that take longer can be used. After selecting the boring and anchoring as separate systems the robot will need a system to bore, a system to extend the boring system while it cuts that also is able to retract into the rest of the body, a system to steer the angle of boring, and two anchoring systems. One that works for advancement, and one that works to pull the rest of the robot towards the boring head. The chart in Appendix D shows the entire decision tree for the project, including the methodologies for the design of various components and systems.

The electrical components of the MOLE were chosen for their price, obtainability, and capability. Without the funding of a sponsor, many components onboard can be bought from commercial sources such as Amazon and SparkFun. Most of the interchangeable components of the robot such as the motors, motor controllers, and sensors were chosen with this in mind. There are definitely more advanced and expensive components which can provide the same fundamental operations the robot needs to carry out. As this is the first iteration of the project, the performance requirements of these components were not known, so optimizing for ease of access and price seemed reasonable.

Major components like the robot's computer and power supply were chosen for their capabilities. The power supply is an industrial grade component which was chosen because it provides the most power for still fitting within the MOLE's thin chassis. The Teensy 3.6 is a USB-based microcontroller featuring a 32-bit 180 MHz ARM Cortex-M4 processor.⁸ With 6, 16-bit PWM timers able to use 22 pins on the board, it has the facilities to run multiple PID loops simultaneously. The Teensy has 62 digital I/O pins all interrupt capable for limit switches, encoders, and hall effect sensors. The Teensy also features 5 PWM timers which allows for the control of up to 5 motors without the need for timer multiplexing.

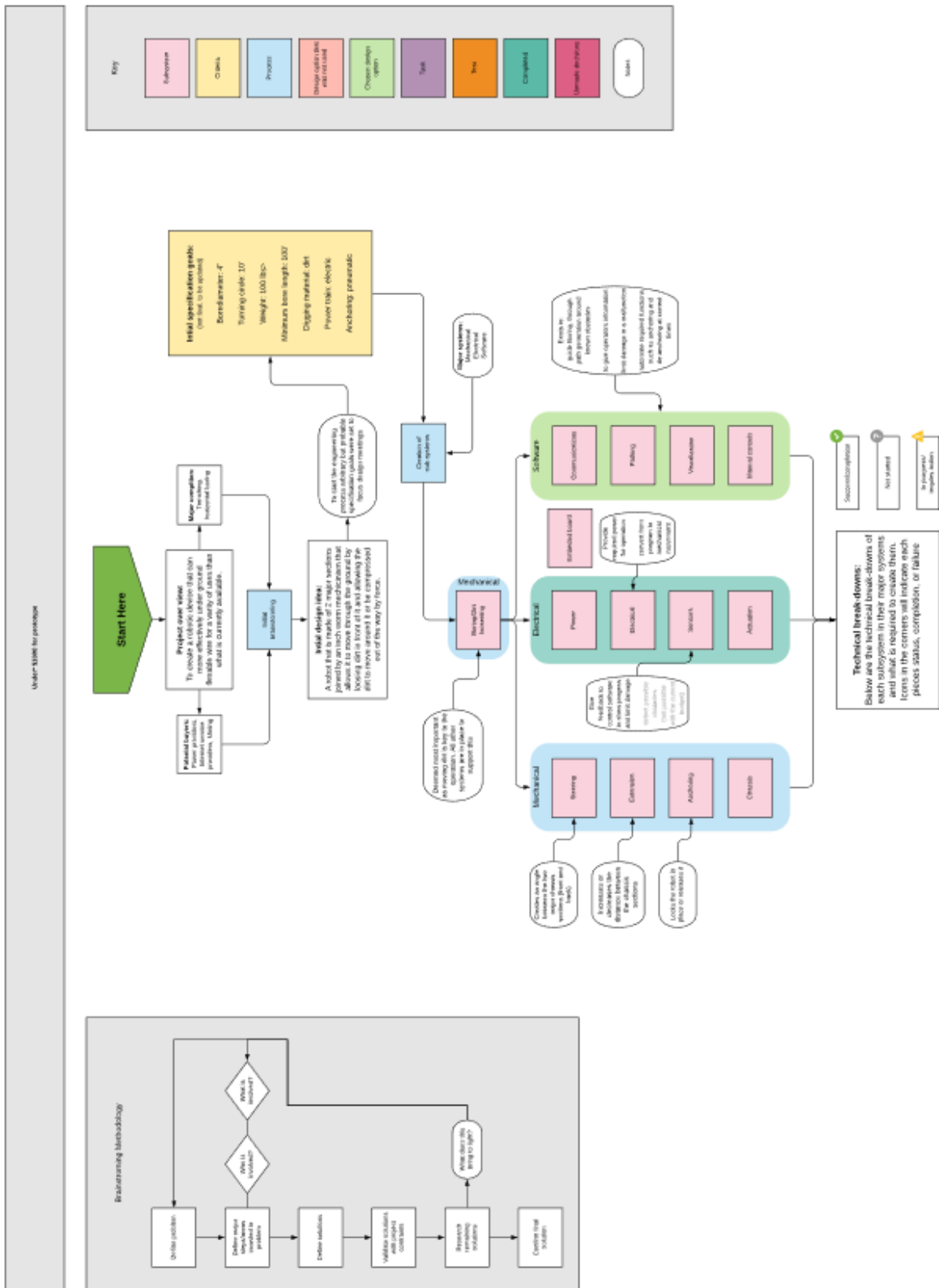


Figure 23 - A chart for the methodology of initial decisions. The expanded chart can be found in Appendix D.

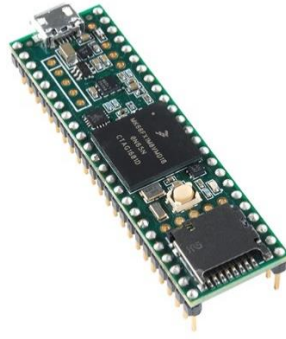


Figure 24 - The PJRC Teensy 3.6 [46]

The design of the software that drives the MOLE was broken down into four main subsystems: frontend/UI, server, data layer, and Arduino script. Each of these subsystems contains modules which interact with one another within the subsystem, and modules which interact with modules of other subsystems.

A JavaScript/HTML frontend user interface is the only software subsystem that an operator will interact with directly. The frontend contains modules for manual control, autonomous path planning, and status visualization. The frontend also contains static HTML web pages: a welcome page and about section. The path planning, controls, and status visualization modules are the edge modules of this subsystem and communicate with the server via HTTP GET and POST requests.

A Node.js server is a JavaScript environment that utilizes Chrome's V8 engine. An HTTP server is initialized and maintained with Node's HTTP module, and a TCP server with network sockets to connect to the Python data layer. The server also has a local database using lowdb which stores previous status reports in addition to controls commands sent to the Arduino. The Node.js server also acts as a gate to make sure that the commands sent to the robot are correctly formatted before they are passed along to the data layer.

The Frontend and Node.js server, along with their modules, are illustrated in Figure 25 below.

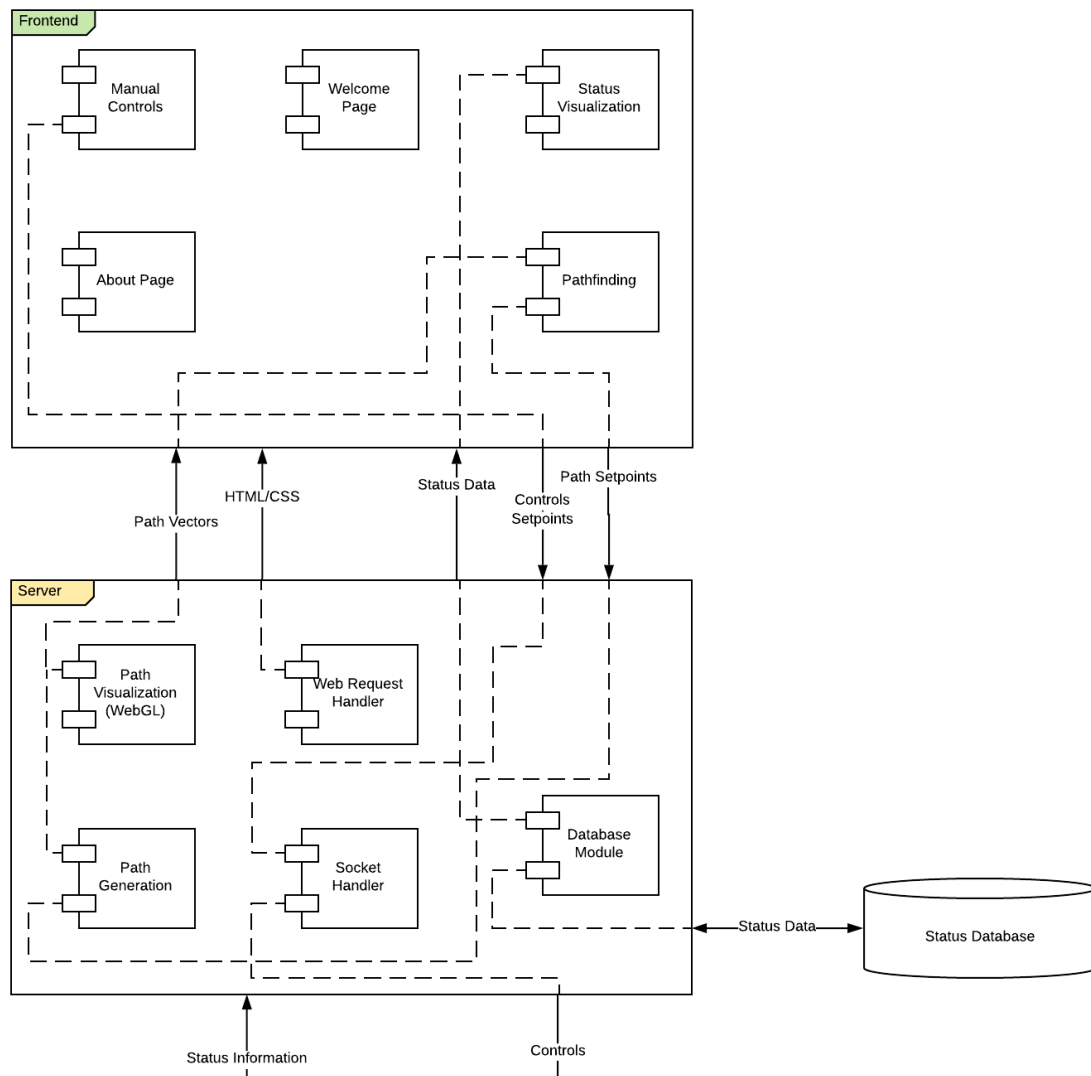


Figure 25 - Software Architecture: Frontend and Server

Between the Node.js server and the Arduino script lies a data/communication layer. This subsystem is written in Python 2.7 and contains three modules. A network socket edge module receives communications from the Node.js server in the form of JSON strings. These strings are passed through a translation module, which converts them into an Arduino-friendly format. The final module is the serial communication module, another edge system. This module opens a serial connection to the MOLE on-board embedded system, sends and receives string messages between the subsystems. This module was developed in Python due to the languages versatility in implementing various control protocols, including network sockets to communicate with the Node.js server, and serial protocols to communicate with the on-board microcontroller.

The final subsystem of MOLE software is the Arduino script, running on the Teensy 3.6 platform. A message parsing edge module receives messages from the data/communication layer, parsing, and storing relevant information contained within the message, and calling appropriate subroutines from the next module, the hardware controller. The hardware control module of the Arduino script interacts directly with MOLE's hardware (motors, sensors, pumps, etc.) through digital and analog IO, using software control protocols such as I²C, PWM and PID.

The data/communication layer, and Arduino script along with their modules, are illustrated in Figure 26 below.

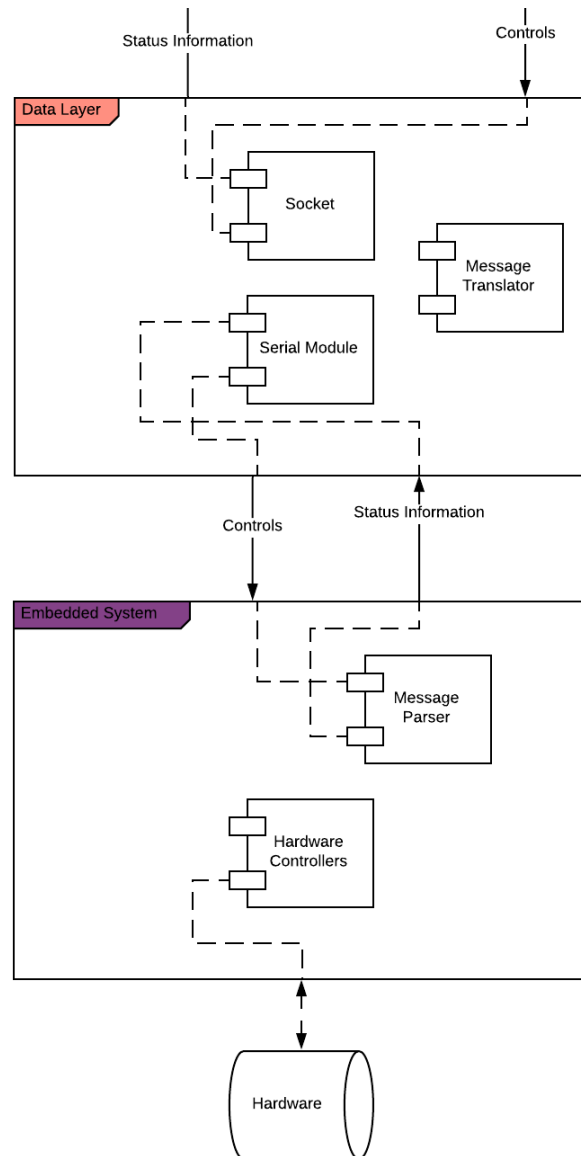


Figure 26 - Software Architecture: Data Layer and Arduino Script

Systems

Overall Design

Boring

Initial Design

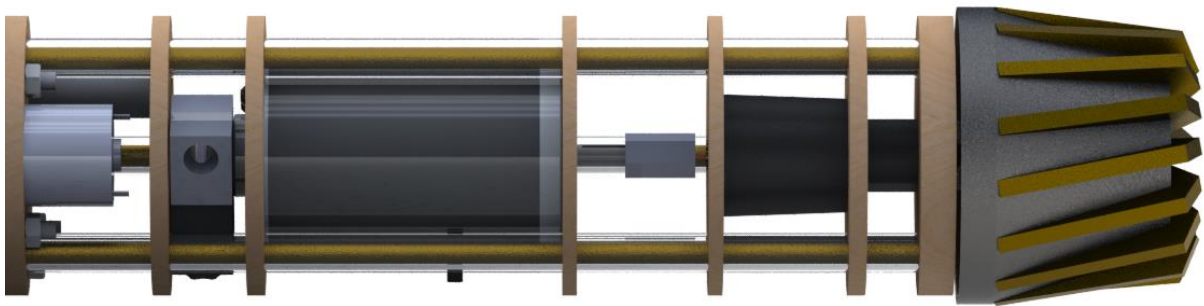


Figure 27 - A render of the boring module without exterior components.

The boring head is threaded onto a gearset. The gearset is taken from a drill, so the two ends need to be worked into the power train. The sun gear is pressed into a shaft that is then set onto the motor shaft. All components of the drive train are held into the chassis by fasteners, as in the case of the motor, or by geometry. The boring head shown in Figure 27 does not exist, as the SARS-CoV-2 epidemic prevented the final boring head production. The design shown is one option of many, the selection of which is encompassed by future work.

Fabrication

The first iteration of the boring module was primarily about fit and proved to us that the geometries we had chosen were suitable. However, the sun gear originally did not fit properly into the gearset and, if forced, produced uneven loading on the gears. The obstacle was clearly one of alignment and arose from one of two sources: concentricity in the adapter and concentricity between the gearbox and the motor shaft.

To address the concentricity of the adapter, we made shaft collars on a lathe to attach motors to the drive shaft. Starting with a cylindrical piece of brass, a small through hole was bored before the OD was turned to size, and then a chamfer was added to the outside. The part was then cut to length before being flipped. A large hole was then bored. Next, the part was moved to a mill where it was fixtured in a square collet block. The center of the diameter was found using an edge finder, and a set screw hole was added. To ensure that the threads were concentric to the hole, a spring center was placed in the drill chuck on the mill, and the hole was tapped using the spring center to align the tap. Using a lathe guaranteed concentricity, but access to the lathe is very limited and it is unlikely that any volume of parts can be created on the lathe. A CNC lathe would provide the capacity for increased production volume. This would benefit areas such as steering where duplicate parts are needed.

To address the issue of concentricity of the gear box to the motor shaft, the plate directly behind the gear box was replaced with an acrylic plate that had been cut to accommodate the very specific geometry of the gear box. Due to the imprecision of the laser cutter, this had to be attempted several times, but the final plate forces the center of the gearbox to within 1 thou of the exact center of the robot. This gives us concentricity and proves the superior tolerancing of the acrylic plates during cutting (a product primarily of the lack of warp in the acrylic sheets as opposed to the plywood sheets).

Electrical

Drilling could benefit from more power, not just slow-moving torque, so this motor, capable of outputting a nearly half a mechanical horsepower was selected. Figure 28 shows the motor curves which show how the motor will behave under a given set of circumstances from the manufacturer of the CIM Motor which will be used for drilling. The most efficient speed for the motor is 2625 rpm with 4.5 kg cm of torque, while the maximum speed is 4500 rpm with 17 kg cm of torque.

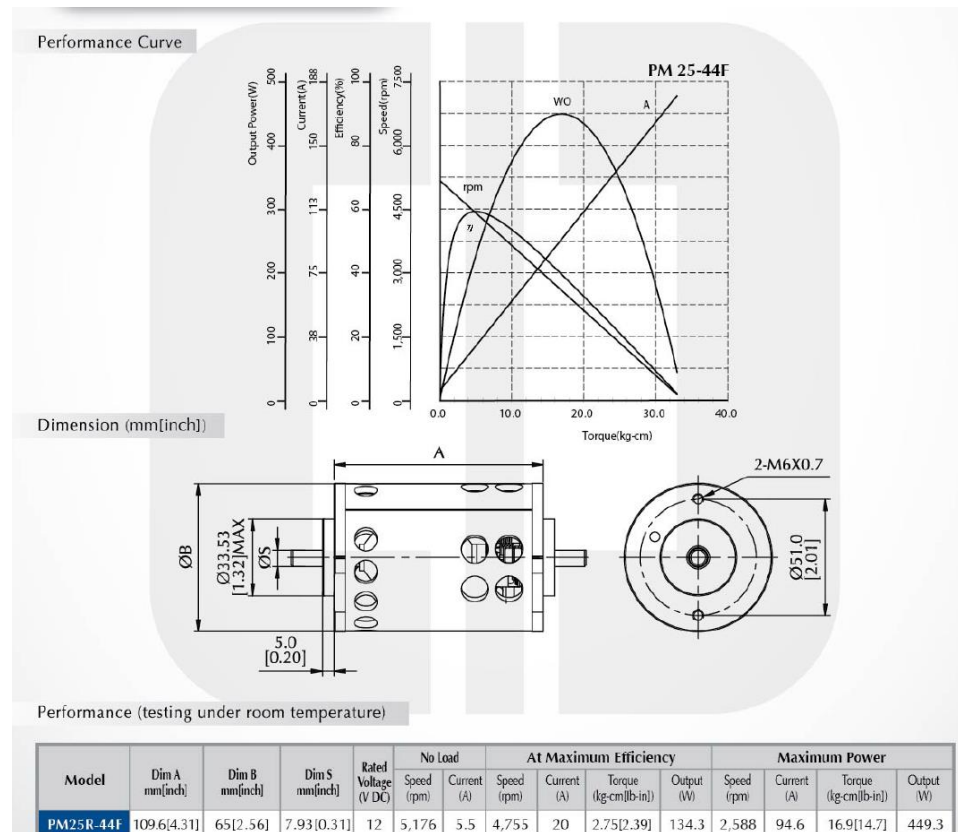


Figure 28 - Chiaphua Components Limited PM25R Motor Curves [47]



Figure 29 – The boring motor, the CCL PM25R [48]

A speed control loop running on the drilling motor is monitored by a non-latching Hall Effect sensor with a small magnet attached to the output shaft. By utilizing one of the I/O pins on the Teensy and attaching that pin to an interrupt, RPM calculations can be made every revolution of the driveshaft in real-time. The code which accomplishes this task is shown below.

```
void computeTachometer(ShaftTachometer tach){
    tach.detect_time = micros();

    tach.elapsed_time = tach.detect_time - tach.previous_detect;
    tach.previous_detect = tach.detect_time;

    tach.RPM = 1.0/(tach.elapsed_time/60000000.0);

    tach.detect = 0;
}
```

Figure 30 – Shaft speed calculation code

The drilling motor is the most power-hungry electrical component on the robot and is capable of drawing high current. According to its data sheet, the CIM motor draws 94.6 A at max power and about 175 A at stall. In order to correctly control this motor, a motor controller capable of handling comparable currents is required. The Victor SPX was chosen for this reason.



Figure 31 – CTR Electronics' Victor SPX [49]

The Victor SPX can manage 60A continuously and handle surges of 100A for 2 second intervals. This control signal from the Teensy to this controller is PWM, but the Victor SPX also features CAN protocols if a future team would like to utilize CAN daisy-chaining rather than running numerous, messy PWM wires.

Sensors

As the boring process will load the boring motor quite heavily, the motor is going to generate a considerable amount of heat. This thermocouple breakout board made by SparkFun is compatible with a standard Type-K thermocouple and is digitally interfaced with the Teensy which allows for thermal monitoring of the boring motor.



Figure 32 - SparkFun Thermocouple Breakout - MAX31855K [50]

The manufacturer of the boring motor, Chiaphua Components Limited Industrial Motors, states that our PM25R series motor, can be produced with insulation classes A, E, or B. As it is unclear which class our motor was produced to meet, a safe estimate is to assume the lowest class, Class A. According to NEMA (National Electrical Manufacturers Association), the maximum operational temperature allowed for a Class A motor is 221 °F. This means that the user interface, on top of showing continuous temperature feedback, will audibly or visually warn the operator when the temperature surpasses 200 °F. The Teensy will disable the CIM motor when the temperature reaches 220 °F and will not resume operation until below 150 °F.

A single AH1815 non-latching Hall effect sensor is used for tachometry on the boring shaft. The tachometry provided by the Hall effect sensor and a single magnet is a cheap solution to monitor shaft speed. This Hall effect sensor requires a minimum 175ms to sense a magnetic field and provide the corresponding output. This limits the maximum shaft speed the sensors can measure. The single magnet also means that the RPM reading is only updated once per revolution. This is a problem if the shaft is suddenly loaded and stopped entirely. This prevents the speed control loop from updating and will continue to command a low amount of power insufficient to move the shaft again. This means that the shaft will never rotate again unless unloaded. In order to prevent this behavior, the computer constantly predicts the shaft's speed based on how long it has gone without an update. If the predicted speed drops below the previous speed calculation, the predicted speed is used as the control loop's process variable. The process variable reverts to being the actual shaft speed calculation upon update.

Assessment

The boring motor can be accurately controlled by a computer. When set to 60, as shown in Figure 33, the boring motor will turn at 1 revolution per second or 60rpm. Because the motor is dynamically monitored based on the response from the Hall Effect sensor, the equal values demonstrate that the computational, electrical,

and mechanical systems are able to rotate at the proper speeds. Given the accuracy of the speed control and the 1:22 reduction across the gear train, the maximum torque generated by the boring motor is 374 kg cm, 27.05 ft lb, or 36.7 Nm. Given the lack of a physical boring head, the team was unable to ascertain whether the available torque is sufficient to bore through the soil.



Figure 33 - Left: the boring speed selection interface. Right: The boring output shaft with magnet and Hall Effect sensor.

Next Steps

The boring section is essentially finished. The next steps are to improve rigidity and concentricity, iterate through boring head designs, and finalize dimensions. It would also be beneficial to optimize the boring system to be shorter in order to reduce the lever arm of the cutting head and the steering forces.

Flow Simulation

In order to determine the ideal geometry for boring in multiple types of soil, a simulation could be done. Similar systems have been modeled using EDEM software [20, No. 194].

Electrical

There remains the possibility of using an AC induction motor for boring. An AC motor was not used in this prototype as its control would require a variable frequency drive which is much more expensive than a DC motor controller. An AC induction motor of the correct size would be more efficient and possibly more powerful than the CIM motor used in this prototype.

Sensors

In order to increase the reliability of shaft speed readings, incremental encoders or pairing a more advanced Hall effect with a trigger wheel should be used to increase the readings per rotation. This will allow the control loop to be tuned with more aggressive gains and therefore be more responsive to deviations in shaft speed.

Extension

Initial Design



Figure 34 - A render of the initial extension design with cutaway shell.

The original design for the extension system was based on the designs of electronic linear actuators. In those actuators, a travelling nut is rotated at the end of a shaft. In this design, a nut is pressed into a PVC shaft that rotates in a bearing. The later design replaces the PVC with a block of acrylic, as discussed later. The base of the shaft butts against a plate and, near the base, a gear is affixed to the outside of the shaft. This gear is driven by the extension motor at the highest reduction that reasonably fits inside the robot.

Fabrication

The initial fabrication of the extension region went poorly. A total of three shafts were manufactured and each had different problems. One was bent due to the application of heat for inserting the nut. Another had a crooked nut due to inaccuracies in forming the internal hex seating for the nut. All in all, forming the shafts proved to be time consuming and ineffective. Between the inaccuracies in the shaft and the overall non-rigidity of the assembly, the threads were binding on almost every revolution. The team decided to redesign the extension assembly.



Figure 35 - A photo of the two extension gears printed from the Form 2 SLA printer.

Luckily, the modular approach to prototype design meant that the extension region could be entirely redesigned without affecting the designs of the front or back modules. The first step was to decrease the total travel of the assembly to improve rigidity. Travel was reduced to three inches with a minimum of five threads engaged at any given time. The second step was to move the nut directly into the gearing assembly. Without the shaft, the assembly loses a source of imprecision and gains rigidity. Lastly, the nut was set into a pair of

bulky, high-accuracy bearings which were in turn set into nine consecutive plates of acrylic. The bearing alignment provided some difficulty, so the resulting block of acrylic maintains a high rigidity in the extension module.

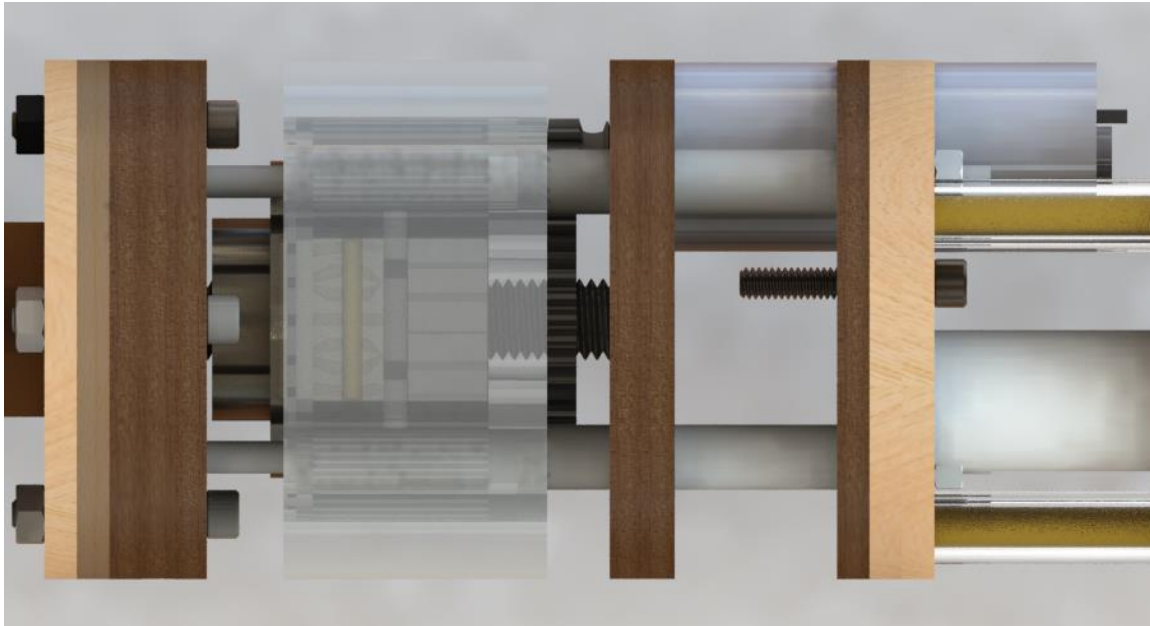


Figure 36 - A render of the final extension module.

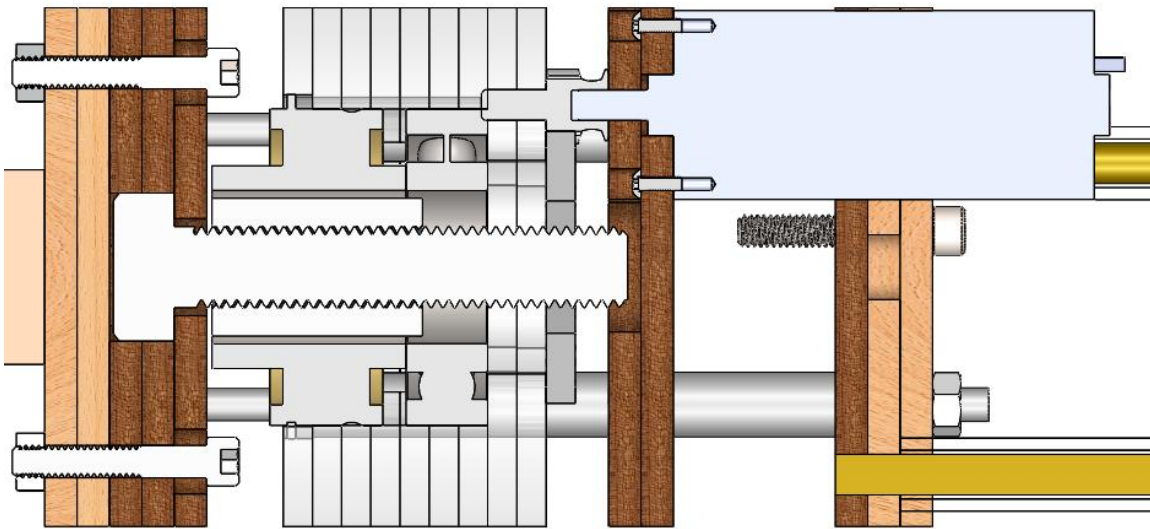


Figure 37 - A section view of the final extension module.

Electrical

For extension, a motor less powerful than the boring motor was chosen. Its rated torque is approximately 80 oz-in while consuming 0.1A. This gear motor features a 1:24 reduction which gives it this large amount of torque with low current draw. This limits its no-load speed to 335 RPM, but as extending the robot should be

a slow, strong, and steady process anyway, this motor is an excellent choice. Its shaft speed is also monitored with an identical setup described in boring. Pulses from a non-latching Hall effect sensor are timed by the Teensy to determine the RPM of the extension shaft. This allows the robot to accurately control its feed rate of the boring head. This motor has been discontinued and its specifications are no longer available.

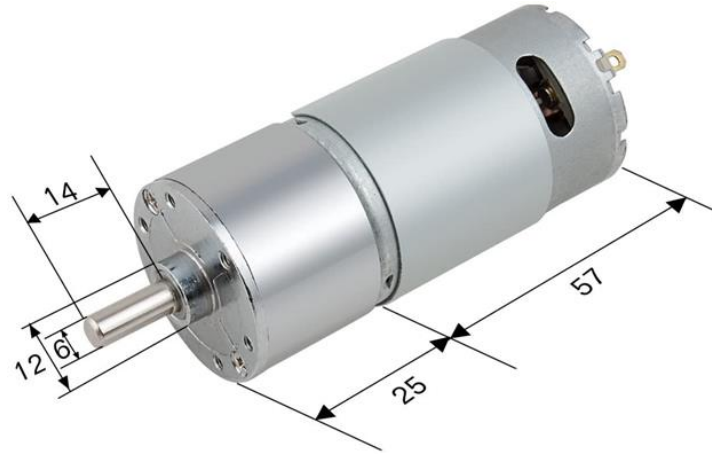


Figure 38 - uxcell DC 12V 335RPM Gear Box Motor with centric output shaft model no. a18030100ux0053 [51]

Specification:

No-load Speed: 300RPM at 12 Volts;

kg.cm: 0.8

Power: 7.2W;

No-load Current: 0.15A;

Load Current: 0.6A;

Reduction Ratio: 1: 17.4

D Shaft Size: 15mm x 5.9mm/0.59 x 0.23 inches (L*D);

Gearbox Size: 26mm x 37mm/ 1.02 x 1.46 inches (L*D);

Motor Size: 56mm x 37.3mm/2.2 x 1.47 inches (L*D);

Mounting Hole Size: M3 (not included).

Figure 39 - Specifications for similar 300RPM motor [51]



Figure 40 - Extension Tachometry Setup

This motor is driven by a L298 Dual H Bridge Motor Speed Controller. This motor controller is rated to output 7A on a single channel, which is much more than required for the expected loading on this motor. The motor controller also features under voltage protection so in the event that the 12V rail begins to falter, the controller is not damaged by sudden spike in current. This motor controller is based off simple H-bridge mechanics, so it only uses PWM control.

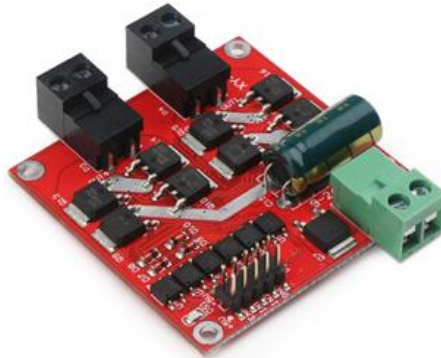


Figure 41 - DROK L298 Dual H Bridge Motor Speed Controller [52]

Although this requires more wires than the Victor SPX for the same number of motors, the code is simple.

```
void driveMotor(MotorController motor, int motorDirection, int throttle){
//direction = 0, forward
//direction = 1, reverse
    if(motorDirection){
        digitalWrite(motor.in1Pin, LOW);
        digitalWrite(motor.in2Pin, HIGH);
    }else{
        digitalWrite(motor.in1Pin, HIGH);
        digitalWrite(motor.in2Pin, LOW);
    }

    analogWrite(motor.enablePin, throttle);
}
```

Figure 42 – H Bridge control code

This code receives the output of the speed control PID for extension motor and drives it accordingly. In most conditions, if the rate of extension increases, the speed of the drilling motor will decrease. Even if the drilling motor is running at full power, it might not be able to spin at our desired speed. This means the control loop on the extension motor has two process variables. One is its own speed, and the other is the drill motor's speed.

Instead, if two process variables of the extension motor were its own speed and the error in the drill motor control loop, they could be combined.

$$P.V. = (\text{Extension Motor Speed}) - K_E * (\text{Drill Motor Error})$$

This means that if the drill motor's error is negative, the resulting process variable will be larger than the extension motor's real speed which will cause its PID controller to slow the motor to correct. In turn, this

should allow the drill motor to increase speed, reducing its error and the extension motor would increase speed again. In the above equation, K_E is a scalar constant to multiply the drill error by in order to change the magnitude of its impact on the extension motor. The limits of extension are detected with upper and lower limit switches. This prevents the Teensy from driving the extension past its operational boundaries.

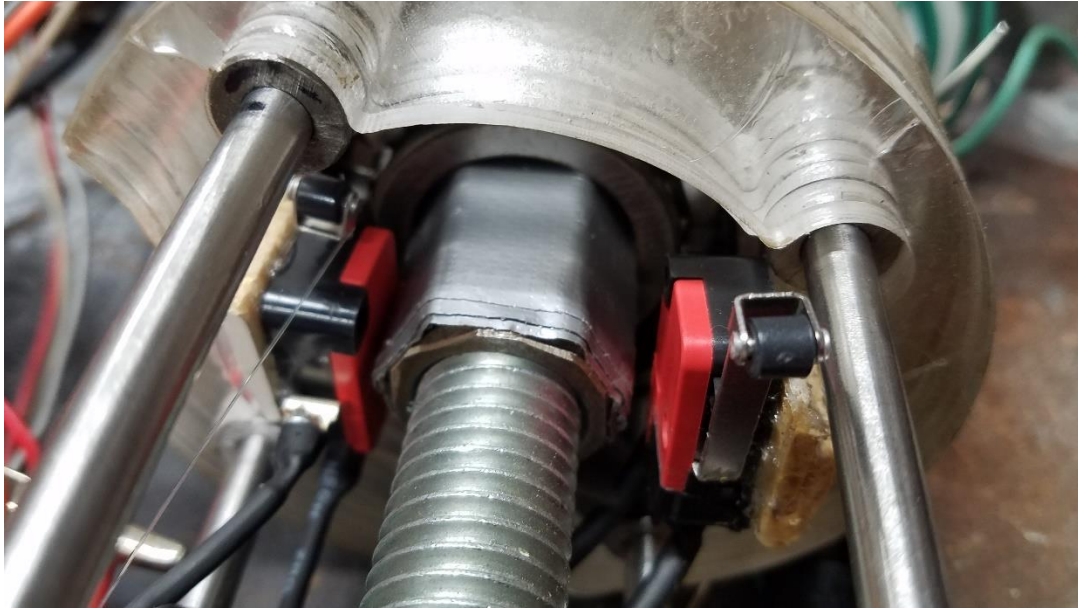


Figure 43 - Extension Limit Switches

These limit switches interrupt the computer and cause the extension motor controller to go into brake mode. This immediately stops the motor before simple logic takes over to ensure the motor can move when commanded, but only in the correct direction. This logic is shown below.

```
int extensionMotorLimitCheck(int motorDirection){
    int allowedMovement = 1;

    if(!digitalRead(fullExtendSwitch) && motorDirection){//if we are extended and want to extend
        allowedMovement = 0;
        // Serial.println("Already extended");
    }

    if(!digitalRead(fullCompressSwitch) && !motorDirection){//if we are compressed and want to compress
        allowedMovement = 0;
        // Serial.println("Already compressed");
    }

    return allowedMovement;
}
```

Figure 44 - Extension Motor Safety Logic

This function is checked when the extension motor is commanded to move by the user sending a speed setpoint. If the movement is not allowed, the motor controller remains in brake and the setpoint is overwritten to be 0 RPM. This prevents integral windup in its speed controller, avoiding overspeed when an allowed movement is eventually commanded.

Assessment

When the extension control is set to -40, it travels the full length and stops when the limit switch is triggered. It does so at a rate of 14 threads in 21 seconds, equivalent to 40 rpm or 4 in/min. The thread pitch of 10 makes it convenient for the operator to switch between linear and angular velocity. The equality of the input and output proves that all the intermediate systems are fully functional. With a maximum rotational speed of 350 rpm, the maximum linear extension rate is 35 in/min. At that speed, accounting for the contraction cycle but not accounting for the additional resistance due to boring through soil, it would take 69 minutes to go 100 feet.

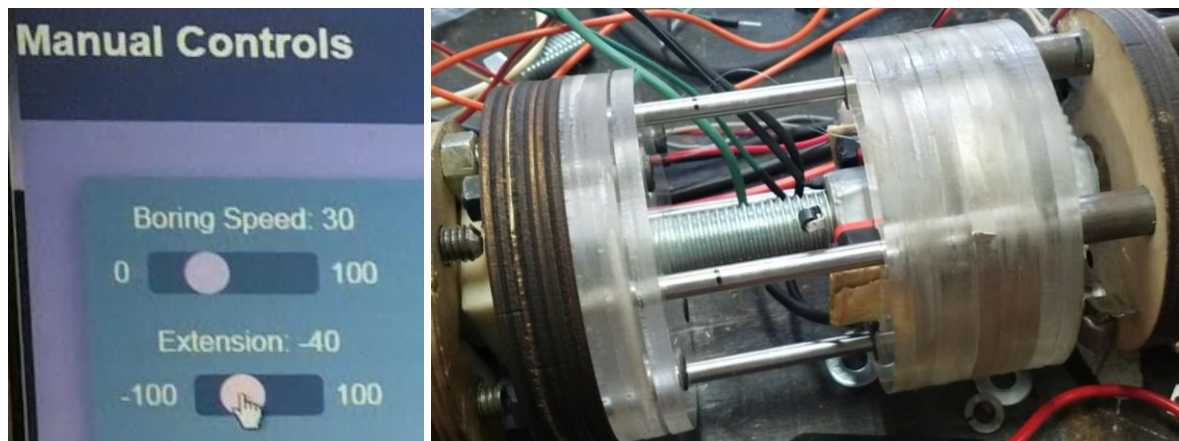


Figure 44 "Left: image of control interface with speed set to 60 rpm. Right: a still of the boring head attachment shaft rotating at 60 rpm."

Next Steps

The extension system is one of the most important subsystems in the robot, but it is also the one that will change the most for the non-prototype design. We are using a custom linear actuator primarily for reasons of cost, but we are recommending a hydraulic cylinder for the final version. A working extension region is needed to validate other elements of the design. Modeling soil provides difficulties so a working extension system is required to validate boring and extension empirically.

There are three main types of actuators. Ones using pneumatics, ones using electric motors with screws and followers, and hydraulic powered ones. Pneumatic actuators are not good in high force applications, so hydraulic, electric and hybrid systems will be considered. In a vacuum, hydraulic actuators are superior to electric actuators, they provide much higher force in the same footprint. A hydraulic cylinder that would fit in the same footprint as a 100lb electric actuator can produce 4420lbs of force. This increase in power can seem incredibly appealing, but there are certain design considerations that come with it. The failure point will likely be in the anchoring system so it doesn't make sense to have an extension system strong enough to break the robot loose from its anchoring. In selecting a new extension system, it is important to consider the overall length. Reducing the length of the robot will reduce the lever arm of the boring head, and reduce the force required to turn. The extension system would also need to be compatible with some sort of cutting force sensor in order to ensure that it feeds at the same rate that material is removed at.

One of the first considerations is whether to use a hydraulic actuator with an onboard pump, known as an electro-hydraulic actuator (EHA) or one with a pump on the surface. The benefit to an external pump is it takes up the least space per power because it does not need a motor or pump next to it. This allows for a bigger more powerful cylinder to be used in the same sized bore. Another benefit is the reduced length per stroke length. The pump on the surface will need to be stronger than an onboard pump to account for the pressure loss due to fluid friction, but this is less of an issue, because there are not tight space constraints above ground. This

type of hydraulic systems will require a circuit to meet the specific controls needs of measurable extension and load limiting. This would at the minimum need to include a fluid control valve and an overpressure system. This would allow for control of feed rate, which is important to ensure that cutting forces remain within a safe threshold. If cutting forces are increasing near an unsafe level, the feed rate should be backed off. The overpressure system would serve as a failsafe if too much pressure is built up in the lines. Since hydraulics are victim to sticking and slipping, a distance sensor will be needed to tell where in the extension it is for accurate path tracing.

Dash Size (1/16")	Velocity (Ft/Sec)	Hose Pressure Drop (psi/Ft)	Total Pressure Drop-Hose & Fittings (psi)	(1) Reynolds Number	(2) Heat Gain (BTUH)	(3) Horse-power Loss
5	62.8	28.8	789.6	7584	17483	6.87
6	43.6	12.2	399.1	6320	8838	3.47
8	24.5	3.2	81.3	4740	1801	0.71
*10	15.7	1.1	31.2	3792	691	0.27
12	10.9	0.5	12.7	3160	280	0.11
16	6.1	0.1	2.6	2730	58	0.02
20	3.9	0.0	0.8	1896	18	0.01

Table 3 – Pressure loss in a hydraulic line [45]

A system with an onboard pump, known as an EHA would offer the benefits of being easier to mount in the robot since it only requires electrical power which is easier to rout than hydraulic lines, and it would not suffer the same losses of long distance hydraulic power transfer. EHAs also often come with onboard sensor packages and are powered by servo or stepper motors so closed loop control is possible. Since all the sensing and power is in one component it would be simpler to install and maintain than a standard hydraulic actuator. In addition, it has the advantage of being a closed system, with fewer pipe fittings, and thus fewer points of failure. The tradeoff off is a decrease in power density, and an increase in length. The robot may need to be redesigned to fit larger components if this solution is to be used.

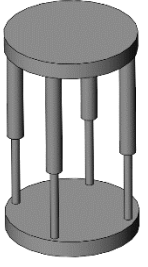


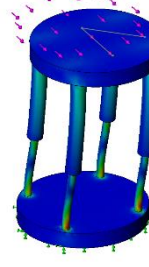
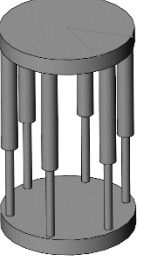
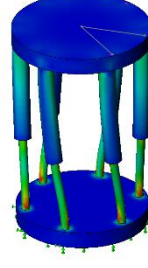
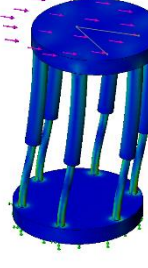
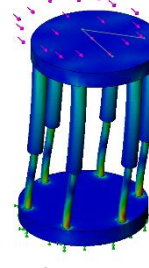
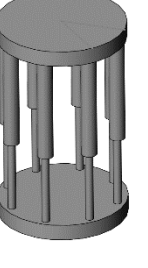
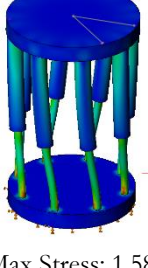
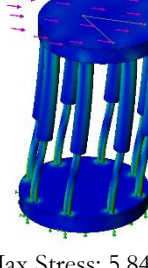
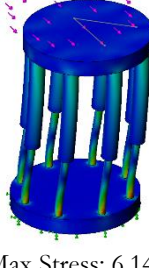
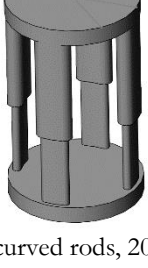
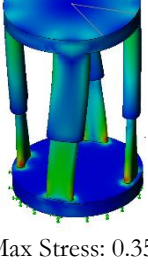
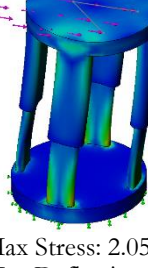
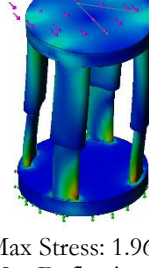
Electric actuators have the benefit of better fine control, their exact position can be calculated by the motor turns without the need for an additional distance sensor. Another benefit to electric actuators is they are cleaner. Hydraulic systems have seals that can spring small leaks which if unchecked might damage electronics. If an electrical system were to fail, it would be far less catastrophic than a hydraulic system. If hydraulic fluid leaks in the robot it is likely that additional components will need to be repaired or replaced whereas if the electric actuator breaks, only it will need replacement.

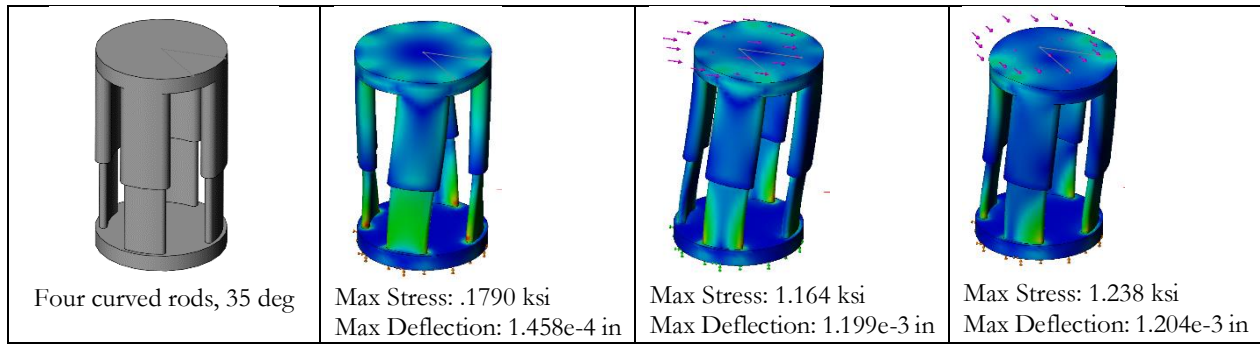
Another option to consider is a servo hydraulic system. This presents the best of both worlds. It is precisely controllable and powerful. They work by using a standard electrical actuator that is assisted by hydraulics. When the output piston begins to move, it actuates an NC valve that allows high pressure fluid to flow behind it and assist the motion. This works in both directions. While this is functionally the best option, they are difficult to find, and a higher complexity will translate into a higher overall cost.

Guidance Posts for Eliminating Torsion

The current guidance posts for extension use a thin internal rod the slides in an outer rod. The purpose of this design is to keep the extension axially aligned and to resist the torsion generated in extension in boring. The table below shows a few comparisons of potential shapes tested in three cases. The first is torsion at 30 in-

lb based on the maximum torque generated by the boring motor. This simulates a rear-anchored situation with the boring motor at full torque, completely locked in the boring substrate. The second is a 50lb force generated perpendicular to the robot, in line with the opposite posts. This represents the full weight of the robot, which is the highest pure turning stress it will experience. The third is the same force at 45 degrees to the second, the weakest orientation of the rods. The material is Aluminum 1060 Alloy. Stress is measured in ksi. Deflection is in inches and is exaggerated in the images.

 <p>Four cylindrical rods.</p>	 <p>Max Stress: 3.068 ksi Max Deflection: 5.831e-3 in</p>	 <p>Max Stress: 10.80 ksi Max Deflection: 2.06e-2 in</p>	 <p>Max Stress: 1.114e1 Max Deflection: 2.059e-2 in</p>
 <p>Six cylindrical rods.</p>	 <p>Max Stress: 1.937 ksi Max Deflection: 3.850e-3 in</p>	 <p>Max Stress: 7.675 ksi Max Deflection: 1.366e-2 in</p>	 <p>Max Stress: 7.620 ksi Max Deflection: 1.367e-2 in</p>
 <p>Eight cylindrical rods.</p>	 <p>Max Stress: 1.58 ksi Max Deflection: 2.917e-3 in</p>	 <p>Max Stress: 5.848 ksi Max Deflection: 1.029e-2 in</p>	 <p>Max Stress: 6.145 ksi Max Deflection: 1.026e-2 in</p>
 <p>Four curved rods, 20 deg</p>	 <p>Max Stress: 0.3518 ksi Max Deflection: 3.796e-4 in</p>	 <p>Max Stress: 2.055 ksi Max Deflection: 2.635e-3 in</p>	 <p>Max Stress: 1.961 ksi Max Deflection: 2.635e-3 in</p>



From a manufacturing perspective, the round rods would be desirable because they are far easier to make than the rectangular ones due to their symmetrical cross-section. In addition, during assembly, fitting the round pegs will be easier since only their position must be considered whereas the rectangular strips must also be aligned to the proper angle.

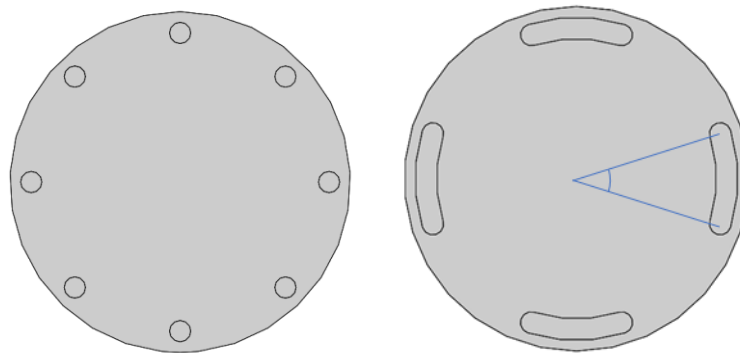


Figure 45 - Cross sections of two different post types. Left: Eight circular rods. Right: Rounded curved rods at 35 degrees (angle in blue).

Electrical

The next step for electrical in extension is to devise a more robust way of running wires and affixing limit switches. Under compression, wires are forced to bend out and around the bottom extension plate. This will not be possible on the real robot as wires should not leave the confines of the robot's outer wall.



Figure 456 - Limit switch wires under compression

The limit switches themselves barely fit in the area around the extension shaft. They are glued to small pieces of scrap wood that were diligently cut and sanded to mate the flat sides of the switches to the round outer wall.



Figure 47 - Limit Switch Prep



Figure 468 - Limit Switch Mounting

This rudimentary mounting worked for bench testing and is a product of the quarantine during this term. Future implementations should provide proper mounting points for limit switches and preferably utilize the mounting holes featured on the switches.

The tachometer setup is identical to the setup described for the boring system. This setup would also benefit from the same recommendations and improvements therein.

Steering

Initial Design

The initial design proved to be very effective, although difficult to manufacture. The design consisted of two motors driving two threaded rods via shaft adapters, with each rod in turn driving a sliding block forward and backward depending on the motor's rotation. Each driving rod is paired with a driven rod via four spur gears. The driven rod, ideally, matches the driving rod's rotation exactly so that the sliding block remains in contact with the frustum. The frustum is held in a specific orientation by the sliding blocks and is what provides the steering actuation.

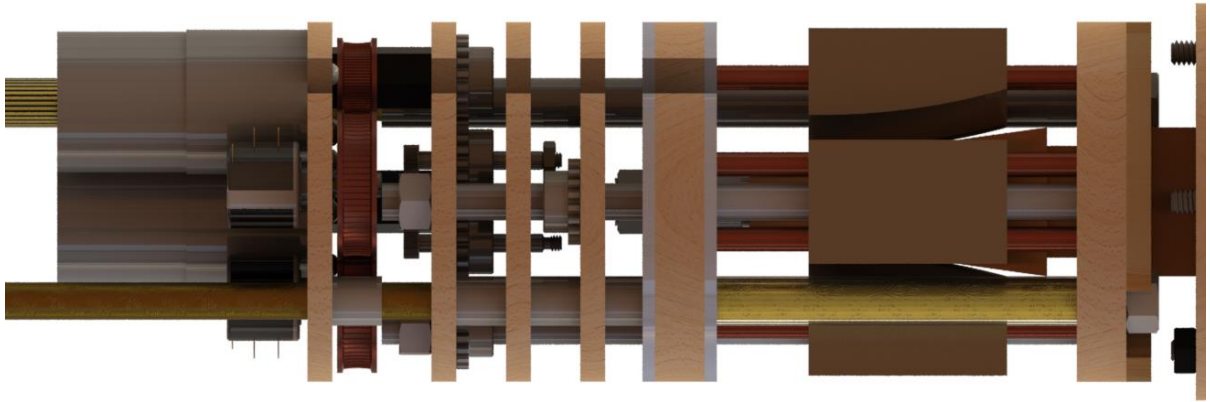


Figure 49a - A render of the steering module.

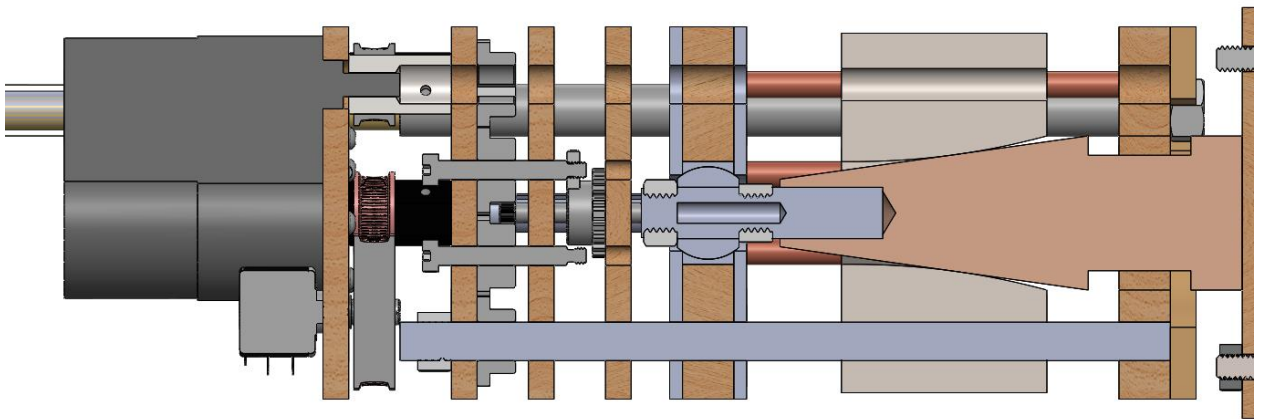


Figure 479b - A section view of the steering module.

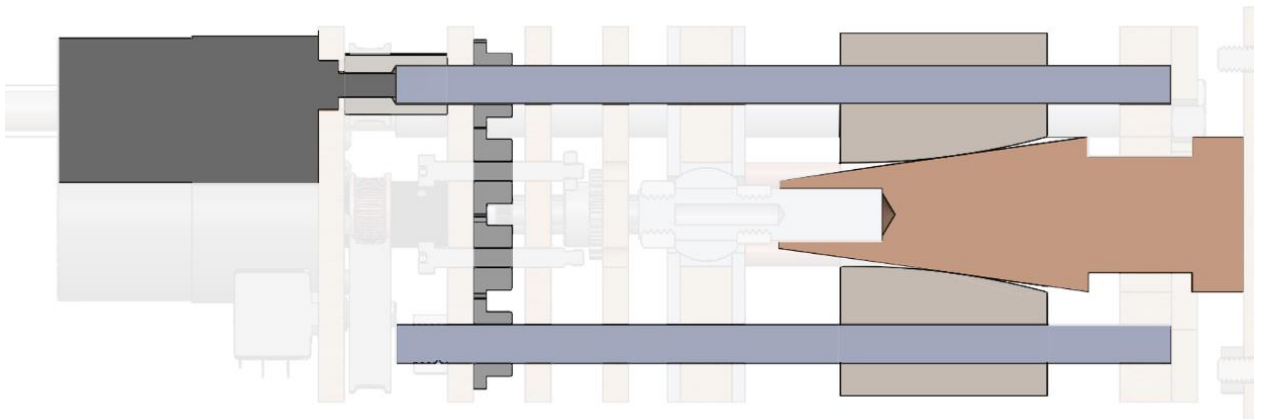


Figure 48c - A section view of the steering module with the dynamic components emphasized.

Fabrication

The slider blocks were machined, and test fit in the steering assembly. The sliders started as a block of aluminum cut to length on a band saw. One side was faced and then the part was flipped, and the outside was roughed out with a face mill. The holes were drilled, and the outside finished with a ball mill. The center hole was tapped using a spring center in a drill press to ensure centering. It was found that the guide rod hole size .257in (F drill) created clearance issues causing the sliders to occasionally bind when the steering module was assembled. After holes were enlarged to .261in (G drill), the blocks slid well and the slider mechanism could be operated with a hand drill. The holes are not perfectly square to the part, but it is a cosmetic issue and will not impact performance because they are only off-axis relative to the curved contactor. This will change the range of angles by shifting the minimum and maximum angle but will not impact performance because the part was designed with built in tolerance. This issue was created by stock not being sufficiently squared before machining. Because it was band saw-cut on both sides and initially faced with the cut side on the parallels, the opposing side was faced to be parallel to the side that was not perpendicular to the walls. This could be fixed in the future by fixturing the part such that the extruded sides are on the parallels and machining off the band saw-cut side with an end mill thus ensuring perpendicularity.



Figure 49 - A series of images showing the manufacture of the aluminum sliders. Clockwise from upper left: CAM of the part, cut stock, a minimill control screen, a finished part, and a milling operation.

Square parts show a keen attention to detail, and contribute to the overall professional appearance of the part, and, while the current slider blocks work, it is important to ensure that if the design is mass produced that the blocks are square to demonstrate well thought out design and machining skill.

The frustum was also supposed to be machined, but the tall narrow shape posed fixturing issues. The mill was able to unseat the part. After this, we elected to 3D print a facsimile for the prototype. It is still recommended to use metal on a more final version.

Our budget did not leave space for metal gears, resulting in use of gears made on a resin 3d printer. They worked well until they were tested under load, and teeth fell off. Hole spacing tolerance also created issues, and this was resolved by redesigning gears with larger teeth to allow for looser tolerance. To replace the 3D printed gears, we laser cut a batch of gears out of acrylic. This serves two purposes. They are stronger than the 3D printed gears and have an incredibly short turnaround time for replacement parts. Metal gears would be stronger, but they require specialized tooling such as a dividing head, or spindexer, in combination with a form cutter, or a gear hob none of which are easy to access at WPI. They could also be made less precisely by using a very small endmill.

Electrical

The motors used for actuating the steering assembly are also gearmotors with preinstalled a 1:22 reduction gearbox. These motors have a rated torque of approximately 14 oz-in which happens when they draw 0.6 A.



Figure 50 - uxcell 12V DC 250RPM Gear Motor with eccentric output shaft model no. A16071400ux0622 [53]

Specification:
No-load Speed: 200-250RPM at 12 Volts;
N.cm: 10
Power: 7.2W;
No-load Current: 0.15A;
Load Current: 0.6A;
Reduction Ratio: 1: 22
D Shaft Size: 14mm x 5.9mm (L*D);
Gearbox Size: 24.5mm x 37mm (L*D);
Motor Size: 33mm x 37.3mm (L*D);
Mounting Hole Size: M3 (not included).

Figure 51 - Steering Motor Specifications [53]

These motors were chosen as they have eccentric output shafts which allows them to be mounted as closely to the chassis wall as possible. This means the steering rods can maintain their close proximity to the outer wall as well without the need for another gear set.

The steering motors are also controlled by another L298. As the L298 is a dual H bridge, only one more is required to actuate the entire steering system. To sense steering position, industrial 10K, 10-turn potentiometers are used.



Figure 52 – Vishay 10 turn Potentiometer 534B1103JL [54]

These potentiometers made by Vishay Intertechnology have a linearity of $\pm 0.25\%$, ensuring reliable position readings. By using a simple pulley reduction, the near 40 turns of the steering rods can be brought down to the 10 turns these potentiometers can handle. The computer's onboard ADC provides 10-bit resolution to each steering shaft. There were a few reasons why using a potentiometer over an encoder made sense in this application. The shaft being measured is not spinning fast enough to worry about wearing out the potentiometer's internals and a potentiometer requires fewer wires for operation than an incremental encoder. An incremental encoder would have had the added complication of installing limit switches in order to carry out a homing routine. Purchasing absolute encoders capable of multi-turn encoding would have been more expensive than these precision potentiometers with no clear benefit.

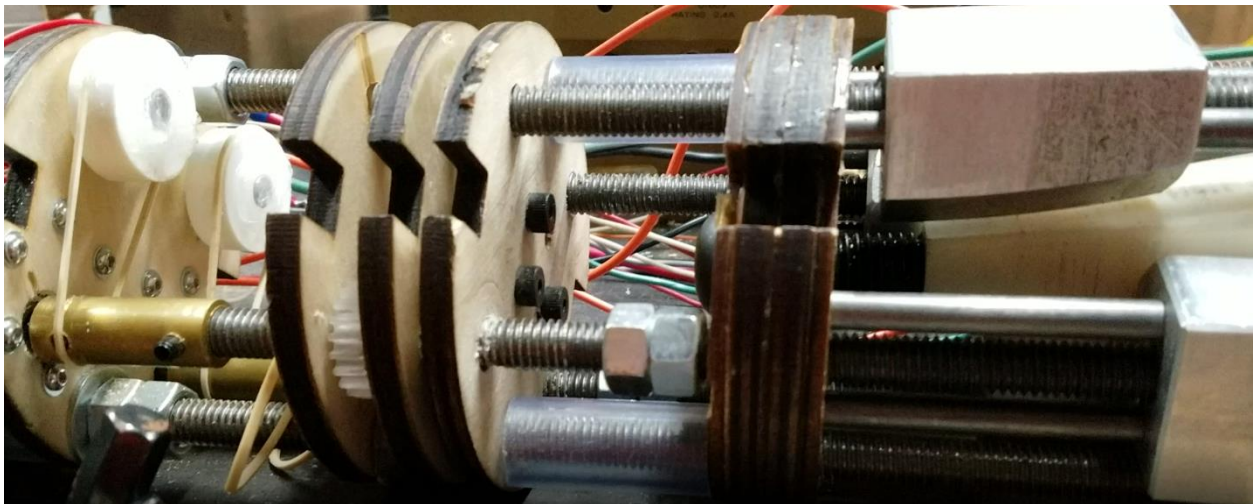


Figure 53 - Steering with installed bands

The rubber band also prevents damage to the potentiometer by slipping if the potentiometer ever reaches one of its hard stops. The independent control of the two motors also required each motor to have a devoted PWM timer on the Teensy. As the Teensy has 5 PWM timers, 4 of which are configurable, all motors on this prototype have independent timers configured to be the maximum frequency the controllers can handle. If more actuators requiring PWM control are added, timer multiplexing would need to be performed.

Assessment

Outside of the failure of the gears, the steering works as expected. Figure 54 shows change in the angle of the front module relative to the back module as a function of the progression of the slider block. The steering system is able to completely lift the front module without slowing so it stalls far above the 20 lb of the front module. The team was unable to test the maximum force because the SARS-CoV-2 closures meant that no part of the robot could be replaced and the risk of breaking components was too great.

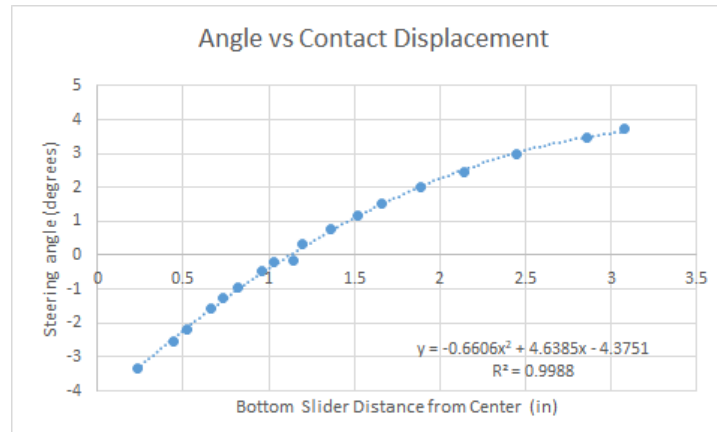


Figure 54 - Steering angle as a function of slider block progression.

Next Steps

The steering system has now been validated under light loading. It now has to become strong enough to be validated for heavy loading, especially because—unlike the extension region—the design for steering is one of the aspects that is intended to be very similar in the final version. To that end, stronger gears are needed, and it seems as though those will either need to be purchased (which we cannot afford) or made out of metal (which we likely cannot do). For the time being, the laser cut gears seem to be strong enough. A future team might want to attempt to make the gears on a CNC. The motor adapters could also be improved if desired, but they are currently fully functional if not very precise.

Gear Design Specifics

Currently, the point of failure in the steering system is the laser-cut and 3D-printed gears. In standard operation, the gears experience fracturing failure along the lines indicated in red in Figure 55. In addition, the gears experience smaller-scale fracturing in the teeth, which causes binding and skipping. While the final model will replace spur gears with a worm and wheel system, a prototype will need functional spur gears if it is to be tested in any sort of boring substrate. In order to design gears that are strong enough, two primary decisions need to be made. The first is the material selection. Acrylic was chosen because it could be laser-cut and was stronger than the 3D-printed gears, but it has proven too fragile for this application. The second decision is the geometry of the gears. Many decisions can be made around geometry, so a few common ones will be explored below.

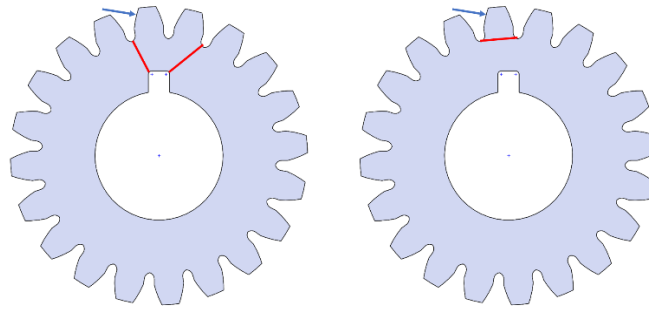


Figure 55 - Two different failure states of current spur gears. Red indicates fracture lines, while blue indicates mode of force.

The forces acting on the gear can be calculated from the motor torque using the equations below. In these equations, T is input torque in Nm, d is the pitch diameter, F_t is tangential force, F_r is radial force, and α is the pressure angle. The equations are in metric but converting to imperial is trivial. The values for the gear are shown in Table 4. In this case, the maximum motor torque is estimated as four times the rated torque of 14 oz-in.

$$F_t = \frac{2000T}{d} \quad F_r = F_t \tan \alpha$$

Input Torque	T	8.7032	Nm
Gear Diameter	d	22.098	mm
Pressure Angle	α	20	deg
Tangential Force	F_t	787.69	N
Radial Force	F_r	286.69	N
Tangential Force	F_t	177.08	lbf
Radial Force	F_r	64.451	lbf

Table 4 - Table of values for gear calculations and FEA.

The first FEA set was motivated by the need to find an adequate material. The limitations of mesh sizing—especially with the SolidWorks implementation, can lead to singularities that would misrepresent the maximum stresses. To avoid this, forces were distributed across a thin strip of material rather than a single line. The original CAD was based on a McMaster Carr gear which was not optimized for adjustment, so changes had to be made to the fillet between the teeth. This was an import change because the fillet is where the highest stresses occur, and the longer the tooth is the greater the bending moment applied at the base. It is advisable that a new equation-driven gear CAD is created, as the McMaster Carr version is not mathematically ideal and does not follow gearing standards when adjusted.

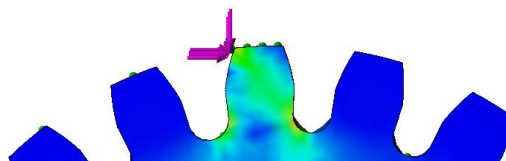


Figure 56 - An example of the FEA load setup and results.

Figure 57 shows the results of the FEA from two different angles. In both cases, force is applied left-to-right on the left side of the tooth. Compression on the back side of the failing tooth seems to be the cause in most cases, so a compressive yield strength should be used to calculate safety factors. The three primary

locations are unsurprising and resemble the real-world failure modes; the tooth breaks at the base when peak stress is at the bottom, but sometimes fractures off a piece along the tip instead.

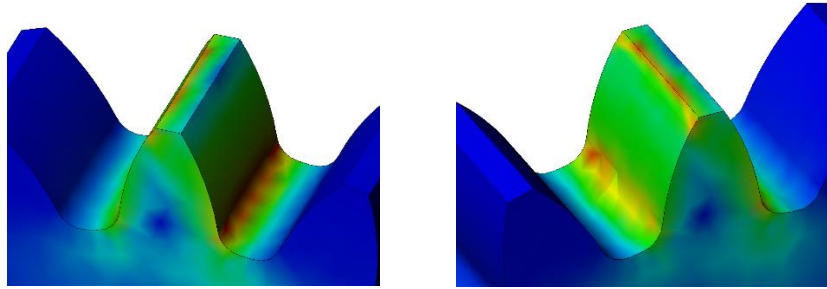


Figure 57 - Stress concentrations on the gear tooth. Note the three primary locations of stress. Force is applied from the left in both images.

The charts in Figure 59 and Figure 60 can be used to motivate a decision around gear material. This group selected acrylic because it was within budget, but a later project group may have a greater budget. The charts show that steel is the only material that can provide a safety factor above 1.5 based on the internal SolidWorks yield strengths. This can be modified by changing the geometry of the gear to reduce stress concentrations and provide reinforcement to the teeth. Alternatively, the next group could decide to skip the spur gears and move onto a better system, such as the one described in the next section. Lastly, Figure 58 shows the stresses the result from reversing the forces so that the key experiences the force while several gear teeth are held immobile. The stresses follow paths that resemble the left-hand failure mode in Figure 55. This can also be mitigated, and reducing stress concentrations will also reduce the likelihood of crack formation and propagation.

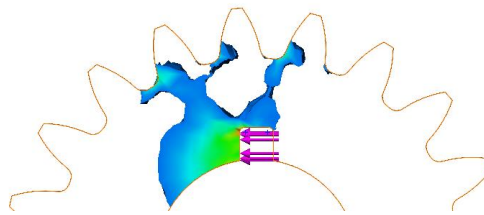


Figure 58 - A view of stresses in the gear created by isolating stresses above a certain point.

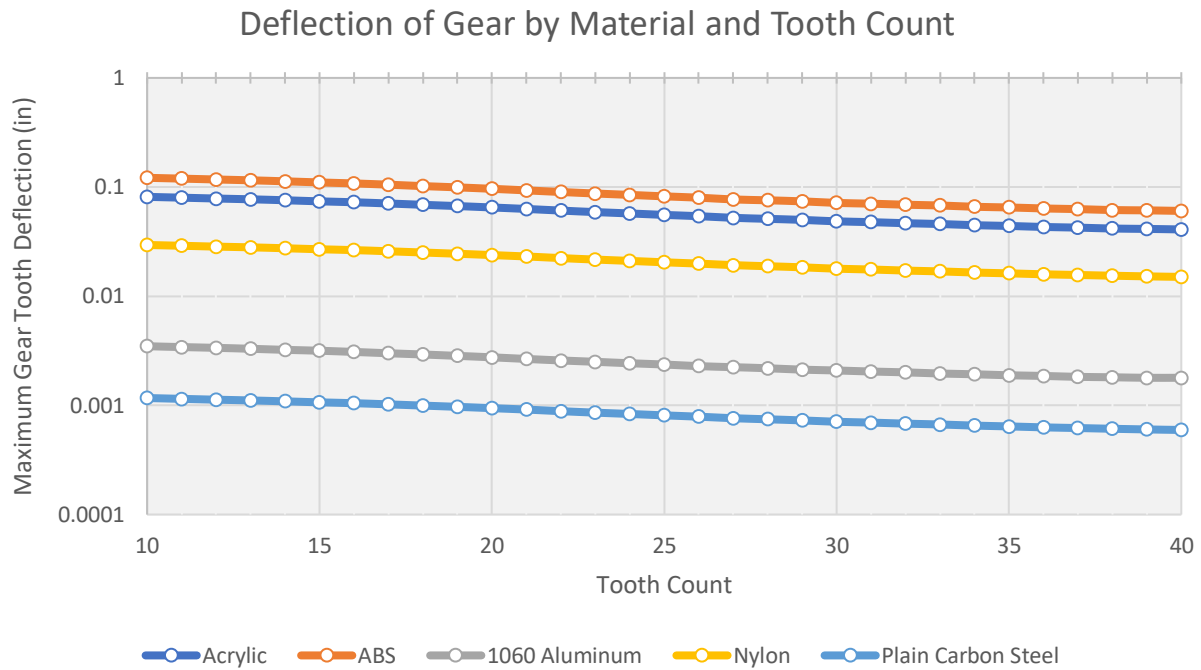


Figure 59 - Deflection of the gear tooth by material and tooth count.

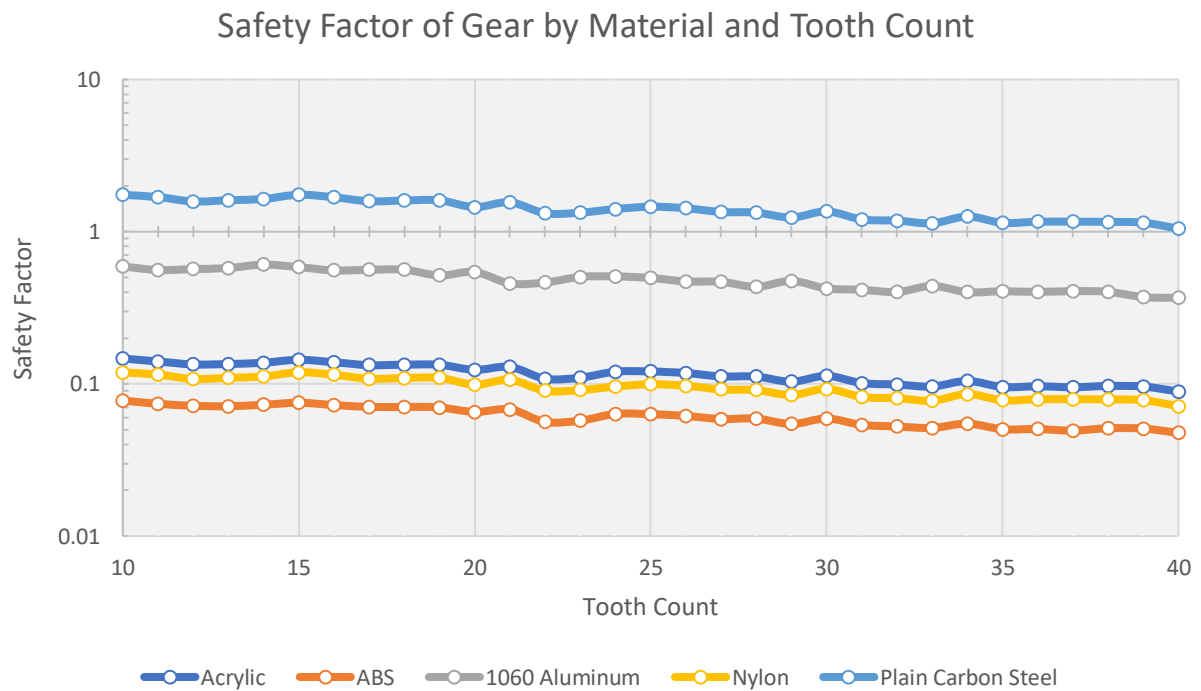


Figure 60 - Gear safety factors by tooth count and material.

Future Gear setup

A future iteration of the robot will ideally not use spur gears and will instead have a more robust and reliable gearing system. The current design consists of four spur gears in a line. It is the minimally effective solution given the financial constraints of the current prototype. Figure 61 shows the current design on the left and the proposed design on the right. The primary difference between the current design and the one on the right is the integration of a worm gear. This system provides several advantages. First and foremost, worm gears are almost insusceptible to back-drive. This allows the steering to be very deterministic and protects the motors from unexpected sudden loading. Second, it reduces backlash. This is not a large concern for the current iterations, but it will become one as the unit gets more rigid and precise. Third, the length of the worm gear provides much more flexibility in positioning the steering motors. While the spur gears have to be attached at one of the four gear center-points, the worm gear can attach anywhere along its length. The precise method of linking the motor to the gearing depends on the design of the motors, one method is the use of two bevel gears to transfer rotation 90 degrees, from axial to transverse relative to the robot.

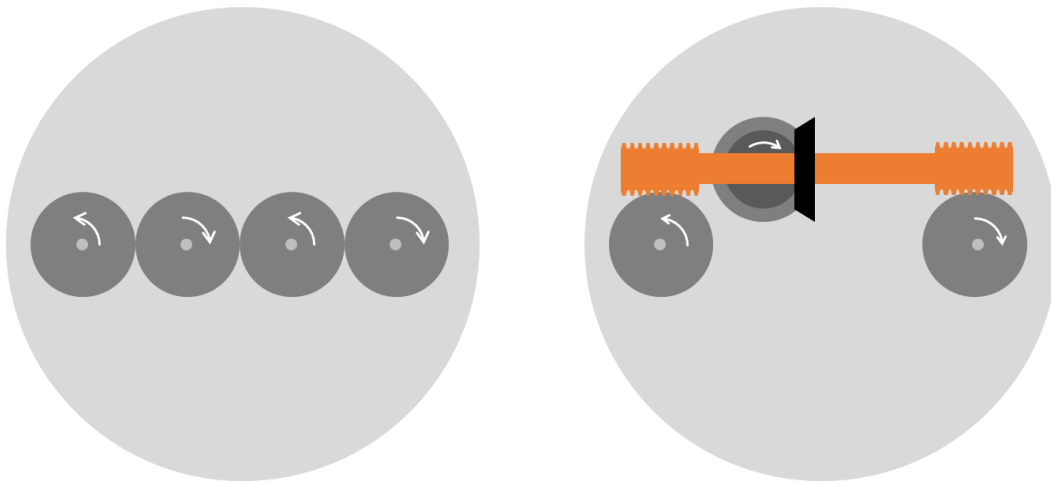


Figure 61 - Left: the current steering gearing on the prototype. Right: the proposed gearing for future iterations.

Frustum Geometry Considerations

The frustum is a very important structural component, and the current one will need to be replaced. There are some considerations in its design. Foremost, there is an obstruction that occurs if a flat sliding block is used with a flat frustum wall. This can be difficult to visualize, but it is the reason that the surfaces of the sliding blocks are curved. If the frustum is held in place by four flat sliding blocks, then the motion of one of the pairs of blocks will not drive it. It will be held in place by the other pairs, because moving them as flat pieces would require the frustum to phase through one side of the sliding blocks.

Potentiometer Pulley

The potentiometer that tracks steering needs a small redesign. Currently, the rubber band has a tendency to walk off of the two pulleys. In order to fix this, one of the pulleys needs a wall. This is a trivial fix, but it is worth mentioning. The only real design constraint is that the ratio between the pulleys has to be high enough that the potentiometer's maximum of ten turns is not reached.

Using Rollers Instead of Sliders

Currently, the steering blocks slide across the frustum to generate motion. The resulting friction is necessary with the current design to prevent back-drive. However, a future design could utilize rollers rather than simple sliding blocks. The addition of a worm gear drive as described above would do a lot to prevent back drive. Having a roller would prevent “skipping” or stuttering motion at high load, increasing the precision of the

steering. While it would run smoother, which would reduce the strain on the gears, it would introduce added design complexity and failure points. Of course, the wheel would have to be very strong. There are bearings that can withstand that strength, or the team could decide to use a single piece of metal turned down to shape.

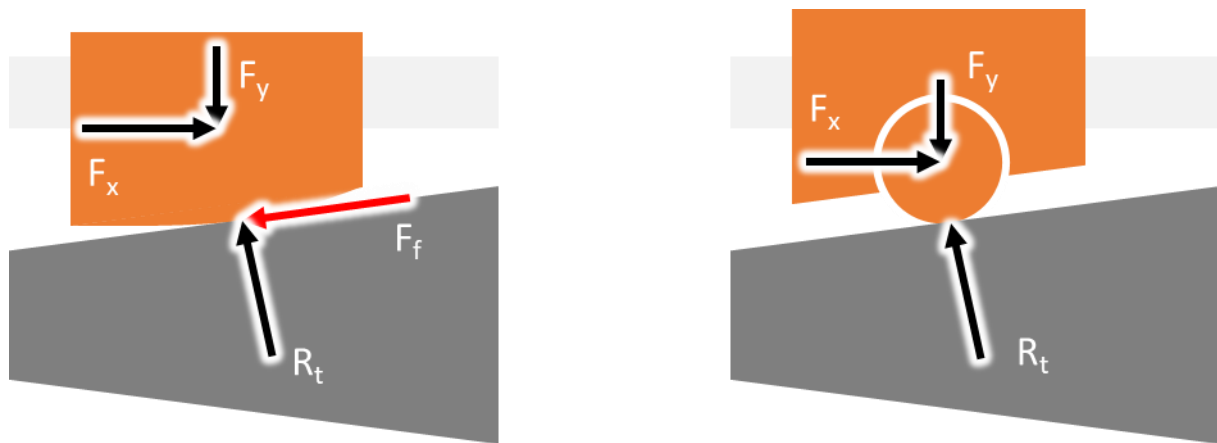


Figure 62 – An illustration of force components on the steering slider, left, and steering roller, right.

Anchoring

Initial Design

In order to ensure proper movement of the robot, the back and front sections must be able to anchor themselves to the surrounding earth. The anchoring system must be able to resist the backwards force of the extension, and the torque from drilling. In addition, they need to be able to be retracted flush to the robot to ensure that the front and back modules can slide easily during the inchworm motion.

Fabrication

For the prototype this will be accomplished by air bladders constructed out of lawnmower tire inner tubes. They will be inflated by an onboard pump, which can hold pressure while it runs, and then pressure can be released with a solenoid bleeder valve. These work by expanding in diameter and pushing on the outside of the tunnel wall. This will work predominantly through friction with the walls, and, in softer soils, it might deform them slightly and create a slight interference fit. While the anchoring tubes were never mounted on the prototype, the pneumatic system was connected to the power supply and functioned in the desired sequence of expansion and contraction.

Electrical

The anchoring system relies on air bladders inflating around the robot to press against the walls of the tunnel, holding the robot in place while boring. For inflating the air bladders, two small, low power air pumps were purchased. By using two air pumps, one for the front section of the robot and one in the rear, pneumatic lines do not need to be managed within the extension portion of the robot.



Figure 63 - DIMINUS DC 6V Mini Air Pump [55]

In order to reliably achieve the maximum amount of anchoring the bladders are capable of, air pressure sensors monitor the pneumatic line to each set of bladders.

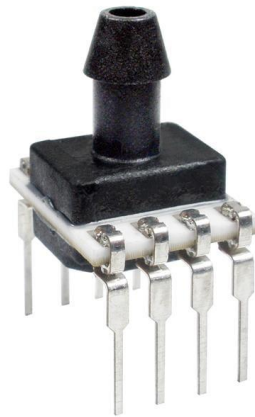


Figure 64 – Honeywell Pressure Sensor HSCDANN030PA2A3 [56]

Not only will these sensors ensure the correct pressures for anchoring in dirt, they allow for future iterations of this project to safely and accurately inflate to optimal pressures for other tunneling conditions such as rock or clay. The sensors also have integrated temperature sensing which can help detect ambient temperature within the robot.

Next Steps

The likely failure mode of this system is shear force within the inner tube, causing it to slip torsionally or axially when the boring head is in use. Because it is completely round, there is no interference between the tube and the tunnel wall. It is for this reason that it is recommended to upgrade the anchoring system for the final product. Given the goal of the final product is to be useful under multiple soil conditions, it is recommended that the anchoring be modular. The basic design will consist of retractable spikes that engage in whatever is being drilled to provide mechanical interference against torque and axial loads. Given this system, it makes sense to add an assortment of different anchoring attachments to be compatible with a wide range of soil conditions. For example, larger flat shovel bits would work well in sand, and shorter spike shapes would perform better in rocky soil. It is also possible that a single design could work, which can be researched.

The pressure sensors communicate through I²C and unfortunately as they are the same component, share the same address. Using them is possible by either using their enable pins like chip select and using the Teensy to enable them independently. Software can also be used to create another I²C bus on a software level.

Chassis

Initial Design

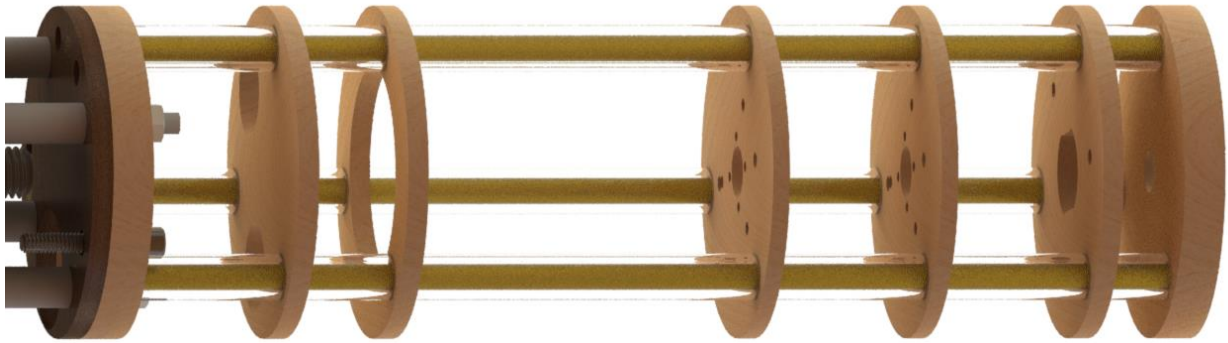


Figure 65 - A render of the chassis for the boring module.

The initial design for the chassis was very simple. Components would be primarily supported by $\frac{1}{4}$ " or $\frac{1}{2}$ " thick discs throughout the robot. The discs would be placed on threaded rods that extend the length of the robot. The discs would be spaced apart from each other by spacers made out of clear PVC for prototyping and metal for the final product. Similarly, discs were to be cut out of plywood on a laser cutter for the prototype, and steel in the final version. The discs would be placed on threaded rods that extend the length of the robot. The rods would be spaced apart from each other by spacers made out of clear PVC in the short term and metal in the long term.

Fabrication

Very little change was seen in the design paradigm for the chassis, but small changes had to be made throughout the process. The laser cut turned out to be far less precise than was originally thought (imprecision of up to 10 thou), and the software used to send files to the cutter scales each print by between 99.5% and 101%. Some iteration was required, but in the end it was determined that the laser cutter was still the best tool for prototyping the discs. Some discs had to be modified by hand, so the CAD was subsequently updated.

The threaded rods worked perfectly. A stiffer steel would provide better overall rigidity, but the slight elasticity is below a threshold that affects the prototype. The manufacture of the spacers proved more difficult than expected. PVC cutters deflect within the material, so a degree of variability occurs. The largest deflection was .025" and the resulting end surface is not flat. However, the spacers can be finished on a belt sander for better dimensioning. Sanding the spacers to square provides a better contact surface ensuring that when the threaded tie rods are tightened the disks remain square to each other. If there is any unevenness, the disks will rock. A lack of square disks can also cascade into the disks being parallelogrammed relative to each other because, while the specified dimension may be somewhere in the contact surface, there is no guarantee that it is actually what is making contact and thereby setting the spacing.



Figure 66 - A photo of the original chassis with many of the components in place.

Electrical Mounting

One difficulty presented by a small robot is that finding space for components that are not actuators or sensors is challenging. Although the Teensy 3.6 microcontroller onboard the MOLE is small, it is a critical component of the robot and needs adequate room for control wires. The motor controllers share this issue, and the space allocated for them and their wires needs to be of adequate size. Even just finding space for wires themselves proved to be difficult.

The backmost module of the robot holds most of the robot's essential electronics. It does come with the benefit of keeping all the motor controllers close to the Teensy, so abundant thin control wires do not need to be carefully run throughout the robot. By finding novel places to mount motor controllers close to the Teensy, only motor power needs to be routed. And as the power supply is also very close, the wires running motor power should be the only place for noticeable power loss. This can be mitigated by running slightly oversized wires to the motors. For example, the drilling motor, which draws the most current, has 12 AWG wire leads, although its factory leads are 14 AWG.

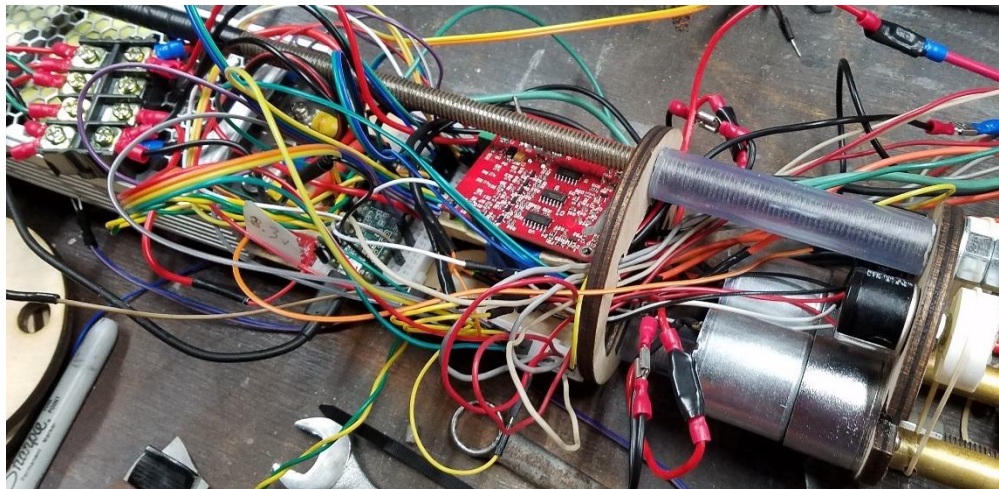


Figure 67 – Top view of wiring for Teensy, motor controllers, and power supply

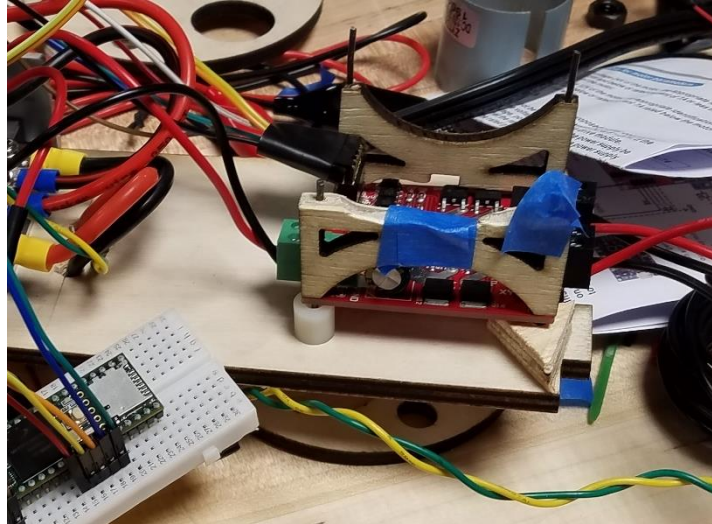


Figure 68 – Side view of motor controller mounting assembly with top controller removed

As shown above, motor controllers can be mounted in different orientations so long as they are not crushed by anything. The pair of motor controllers which operate extension and steering are mounted similar to a bunk bed and were fitted with right angled headers, so control wires are not dangerously bent to fit. These female headers were installed upside down in order to interface correctly with the controllers. The remaining pins, meant to be soldered into a PCB, did not leave adequate space for jumpers to securely attach. This means that pin extensions had to be manually soldered, allowing for secure connection.

The boring motor's controller is not shown in these images. It is mounted beneath the power supply, recessed into the mounting plate of the chassis.

Sensors

Knowing the chassis' orientation allows steering commands to be given in intuitive world coordinates and transformed into local coordinates. This means that regardless of the robot's orientation, the command to drill left will actuate the correct set of contactor blocks to ensure it turns to its left relative to its internal coordinate system.



Figure 69 – SparkFun's LSM9DS1 breakout board, aka 9DoF Sensor Stick [57]

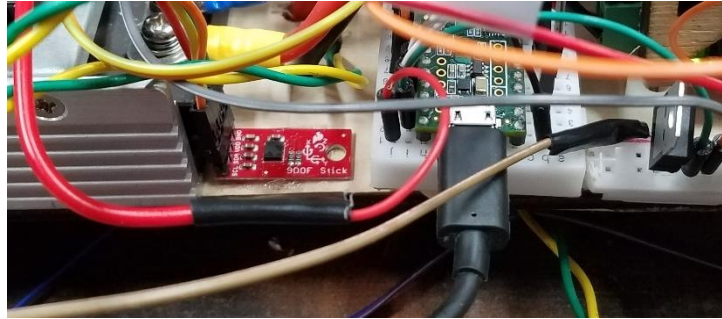


Figure 70 - The IMU affixed next to the Teensy

A dilemma emerged when bench testing of the SparkFun 9DoF Sensor Stick began. The IMU provides accurate and consistent pitch and roll readings, but its yaw readings were unstable and jittery. Upon rotating about the Z axis, the yaw reading would change as expected, but only for about 40 degrees before jumping back to its initial value. The yaw value of this IMU utilizes its onboard magnetometer to compute heading based off the Earth's magnetic field. Even after accounting for WPI's declination, the sensor still gave poor readings. The pitch and roll readings rely on the accelerometer to detect the direction of Earth's gravitational pull and can work out the IMU's orientation. The code which performs these calculations is shown below.

```
void printAttitude(float ax, float ay, float az, float mx, float my, float mz)
{
    float roll = atan2(ay, az);
    float pitch = atan2(-ax, sqrt(ay * ay + az * az));

    float heading;

    if (my == 0) {
        heading = (mx < 0) ? PI : 0;
    } else {
        heading = atan2(my, mx);
    }

    heading -= DECLINATION * PI / 180;

    if (heading > PI) {
        heading -= (2 * PI);
    } else if (heading < -PI) {
        heading += (2 * PI);
    }

    // Convert everything from radians to degrees:
    robotYaw *= 180.0 / PI;
    robotPitch *= 180.0 / PI;
    robotRoll *= 180.0 / PI;
}
//
}
```

Figure 71 – Attitude calculation code [58]

The poor yaw readings are either the result of a faulty chip, or from being a cheap sensor. Without correct yaw readings, performing accurate control transformations is impossible.

Next Steps

The chassis successfully housed all of the components and allowed for their motion. This helped us validate the design of other subsystems at a kinematic level. The next steps are to produce every spacer, as some are missing due to the fluctuating dimensions of various modules. In addition, some spacers and discs could be converted to metal. In order to measure all three axes accurately, a more expensive and better designed IMU should be bought, preferably an IMU which implements some sort of sensor fusion or Kalman filtering.

Ingress Protection

If the unit is to bore through soil, it will have to be protected from the various solids and fluids that can be found underground. The engineering standard for creating an enclosure that can prevent material penetration is the Ingress Protection (IP) rating system. The rating is denoted by two numbers following the letters “IP”, as in IP56. The first numeral represents the ability of the enclosure to protect against solid objects of decreasing size while the second number represents protection against liquids, with the lowest being IP00 and the highest IP68. Because the unit will be under solid and water pressure for most of its operation, it will most likely need to adhere to the IP68 or IP58 ratings, representing total or near-total dust protection and total liquid protection despite long periods of immersion under pressure. [60] It is possible that the specific machine design used in later iterations of the robot will allow a less strict rating for most of the chassis with a robust protection system around the sensitive components (e.g. electronics or gears). Regardless of the approach, meeting the desired IP ratings will pose a significant engineering challenge.

Power Delivery

With a handful of motors, and one of them having a large peak power, the highest wattage AC to DC converter that can fit inside our small chassis would allow the robot to maximize performance. A literature review showed that 220W is the highest power AC to DC converter in the consumer market that would fit within the robot. Using a less user-friendly search among industrial converters eventually yields a 500 W power supply that will fit inside the robot and is within budget.

Table 5 - Power Consumption Calculations

Device	Voltage	Current	Count	Wattage
Teensy 3.6	5 V	500 mA (max)	1	2.5 W
Steering Motors	12 V	0.6A	2	14.4 W
Air Pump	6 V	260 mA(max)	2	3.12 W
Air Solenoids	12 V	540 mA	2	13 W
Hall Effect Sensors	@ 3 V	8 μ A	2	48 μ W
10k Ω , 10-turn Potentiometers	@ 3.3 V	2.178 mA	2	14.375 mW
Pressure Sensors	3.3 V	4 mA	2	26.4 mW
9DoF IMU	3.3 V	50 mA	1	165 mW
Extension Motor	12 V	0.6 A	1	7.2 W
			Total:	40 W

Error! Reference source not found. above shows calculation of power draw for many of the robot's electrical components. The major entries which are missing are the motor controllers and the drilling motor. The motor controllers draw current for their respective motors and any current they require for control logic

can be neglected. With the maximum power draw of the robot coming to 40.425 W, there is 459 W left for the drilling motor. This equates to just over 38 A of current draw, which the Victor SPX motor controller can easily handle.

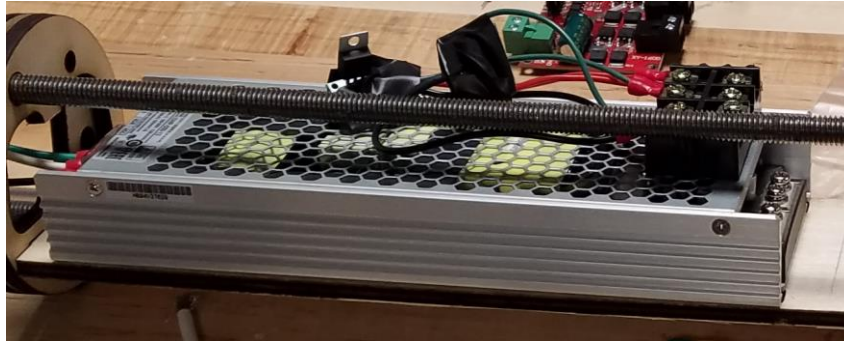


Figure 72 - Power supply with voltage regulators

Having this converter on-board replaces the need for much more expensive batteries which would limit the robot's tunneling distance as they would require charging or changing. Also, as the robot is powered by an AC line, power loss is considerably less than a DC tether, and the tunneling length limit is only dependent on the length or amount of extension cord available.

Software

Architecture

The MOLE software consists of four main components, each with individual modules to accomplish various tasks within the normal operation of the MOLE. These tasks include controlling the robot through user input, viewing the data collected by the robot's on-board sensors, communicating user input from the frontend to the MOLE's on-board computer, and driving electrical components via digital I/O. The software developed to accomplish these tasks will be discussed in this section. Controlling the robot and seeing its status are done in the first software module: the frontend and UI which are based in JavaScript utilizing a Node.js server. Communication between the frontend and the on-board computer is accomplished through a Python communication layer. Finally, the software on the on-board computer is an Arduino script running on the Teensy 3.6 platform.

The Node.js server communicates user input received from the UI with the Python data layer over a socket connection, using JSON objects to encode data. Similarly, the Python data layer communicates system status to the Node.js server via JSON objects which encode fields corresponding to the MOLE's on-board sensors. This information is then served to the user via a web browser. All communication between the Python data layer and the MOLE itself is performed over a serial connection. Encoded controls data are sent to the MOLE, and it replies with status information about MOLE operation. A full description of system architecture can be found in the Methodology chapter, and an illustration of system architecture is contained in Appendix B – Software Architecture.

Frontend

The first software module is the Frontend, which consists of user interface components for both controlling the MOLE and viewing its status information. Pathing or manual control information can be passed in by the user and sent to the MOLE. A lowdb database stores all status information received by the frontend and can be served to the user through a D3js visualization or exported into .csv data format. All the frontend modules are hosted on a local Node.js server and are accessed through a web browser.

Controls

In the controls section of the frontend user interface, an operator will be able to manually set the various control parameters of the MOLE. The parameters an operator is able to set include the MOLE's boring head RPM, extension rate (with negative values indicating to drive the system in reverse), turning angle in the X and Z directions, and inflation of the front and rear anchoring systems. These controls were created using a HTML form and styled with standard CSS. Figure below shows an early iteration of the controls UI.

The image displays a web-based control interface. On the left, there are four horizontal sliders: 'Boring Speed' (0 to 20), 'Extension' (0 to 20), 'Turning X' (-3.5 to 3.5), and 'Turning Z' (-3.5 to 3.5). Below these are two buttons labeled 'Inflate Front' and 'Inflate Back'. On the right, a section titled 'Command Log' contains a large orange rectangular area with the text 'Sent commands go here'.

Figure 74 - Early Controls UI Iteration

The controls system parses the input to the various control fields, and POSTs them to the Node.js server, which then formats and passes along the controls information to the Python communication layer. A running log of all sent commands is kept and presented to the user in a command log to the right of the control inputs. This command log also notifies an operator if a command they attempted to send did not change any of the already set parameters. After the functionality of the control system was verified, the team went on to apply CSS styling to this module to make it more user friendly and visually appealing; Figures 75 a and b Figure below show the final user interface controls and command log, respectively. The bounds of the various control systems were also adjusted as hardware limitations were understood and specific mechanical systems were implemented, and the system is implemented to allow for changes as the mechanical design evolves. Boring speed is set in RPM, on a 0 to 100 scale. Extension is configured from -100 to 100, representing a percentage of full speed backward or forward, respectively. Turning X and Z are on a 0 to 1 scale, with 0 representing no turning, and 1 representing full turning in one direction. Due to hardware and manufacturing constraints, the robot can only turn in one direction on each axis.

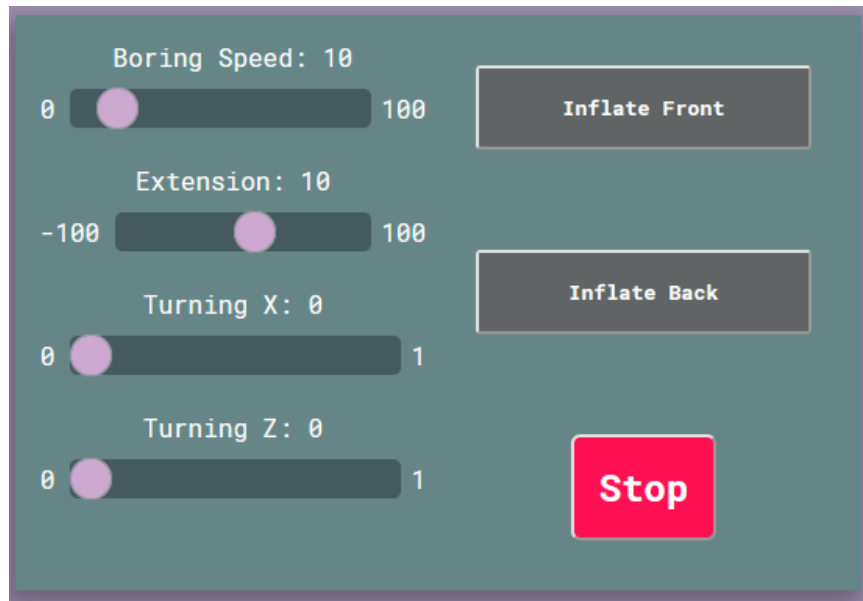


Figure 75a - Final Controls UI Iteration

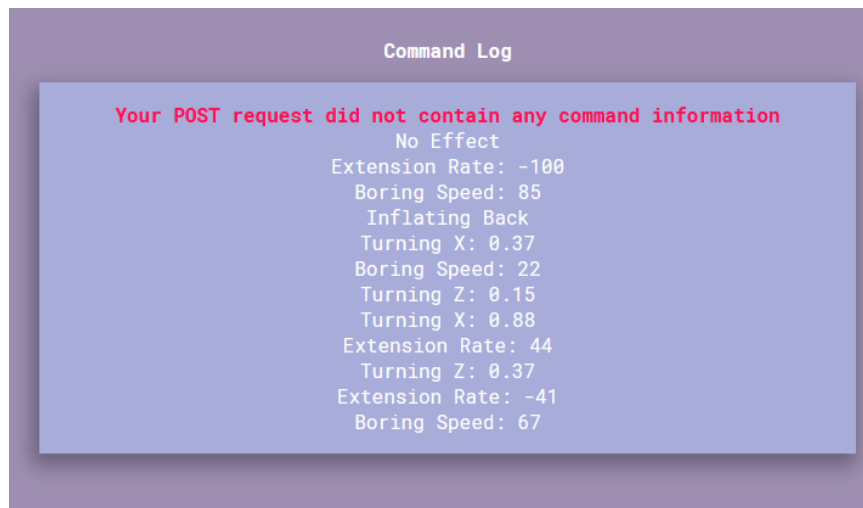


Figure 75b - Final Command Log Iteration

Status Visualization

Presentation of status information to the operator is accomplished with a data visualization created with the D3js library. D3, or Data Driven Documents, is a powerful framework that allows for the creation of dynamic and versatile data visualizations with real-time updating data and can be seamlessly integrated into web pages and the Node.js server. Data to be visualized are passed to the web application from the embedded system through a Python data layer (described below). The status visualization allows for dynamic switching between different datasets, which allows an operator to view the different sensor values of the MOLE.

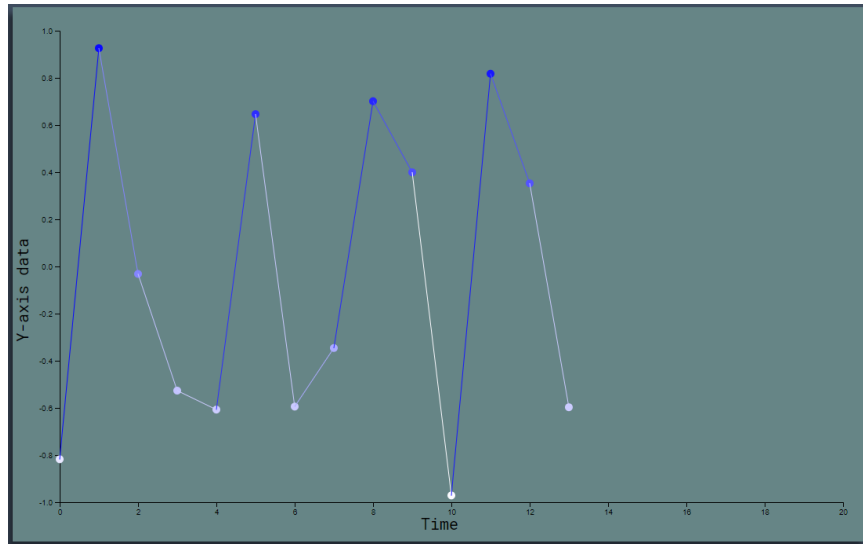


Figure 76 - D3 Status Visualization

D3 allows for the creation and manipulation of document objects using arbitrary data, which can dynamically change, updating the document in the process. The MOLE status visualization makes use of D3's enter, update, exit model [4]. To begin, an empty array of data is bound to an empty selection of SVG elements. When new data are added to the array, the enter routine runs. The MOLE visualization makes use of two types of SVG objects: circles for the data points, and SVG lines to connect the data points to each other. When new data are added to the array, the enter routine creates new circles and lines and appends them to the SVG object. Figure 76 above shows the D3 status visualization, displaying some sample data.

When data changes, (including when new data are inserted) the update routine runs. This allows for the document to quickly adapt to changing data. The update routine is also where the data-driven properties of an object are defined and applied. X and Y positions, as well as object color, are all defined in the update routine. X and Y position make use of D3's `scaleLinear` functionality is used to ensure that objects remain within the bounds of the visualization area. The `scaleLinear` functionality is also used to define the color scale that applies to the points, as a linear scale in D3 can map any numeric domain to an arbitrary range, including a color range that is automatically interpolated.

Finally, when data are deleted from the array, the exit routine runs. In this visualization, the exit routine simply removes the associated point and line from the visualization.

Data to be visualized are pulled from a local database, and all data are persistent across runs of the MOLE system or reloads of the visualization page itself. Any data that are loaded into the visualization from the MOLE must first pass through the database to be stored for future use; although this introduces a slight delay when compared to polling directly from the communication layer, allowing the data to be stored in a persistent manner allows for a more robust data infrastructure and easier integration with the other modules of the MOLE software.

D3's enter, update, exit pattern was determined to be the best solution for this project as the data being presented to the user are constantly updating and changing, and D3's dynamic nature allows us to present changing data seamlessly. Other tools considered, such as Python's matplotlib and NumPy libraries allow for easier creation of visualizations, but do not easily allow for live updates with changing data. The plotly library was also considered, as it would allow for easy integration into the Node.js server, due to it creating HTML

pages containing a visualization. Like the other alternatives explored, plotly fails to capture the behavior we required when dealing with data that are updating rapidly.

Path Visualization

In order to meaningfully display the pathing information, a 3D render must be made of the world that the robot is moving through. This is accomplished by using WebGL, which is a JavaScript API for rendering graphics on a webpage. WebGL was an advantageous choice for our team because it is the primary graphics language taught at WPI in CS4731: Computer Graphics, and it allowed us to keep the pathing visualization in the same place as the controls. In addition, working within the browser allowed us to include the pathing and the pathing visualization within the full stack. WebGL is a JavaScript API, meaning it is cross-platform which allows our software to be picked up by another team without too much effort. In addition, the pathing visualization depends on the MV library provided in CS4731.

Since the scale for the entire area that is encompassed by the MOLE's path can be extremely large, a way to navigate the space was required. The way to solve this issue was to implement a camera that could be controlled by the user to navigate the 3d space. We used the Gram-Schmidt method of composing a view matrix that creates a view space and transforms the objects according to its direction and position [24]. This allows a user to move through space via a "direction vector" and we can alter a camera's translation and rotation by affecting this "direction vector" and the camera's position [24]. The pathing visualization system keeps track of the camera's front-facing vector along with the camera's yaw and pitch, and the camera's right vector can be found at any time by the cross product of the direction vector and the world's y-axis.

Also, because the camera is composed with the Gramm-Schmidt method we cannot allow the user to rotate the camera's pitch to -90 or 90 degrees as that would cause Gimbal-lock, so we added the constraint to keep the camera's pitch within the bounds of -89 and 89 degrees. Gimbal-lock is an issue that arises when implementing 3D rotations with Euler angles; it is caused when one axis of rotation becomes parallel with another axis [29]. The common situation is that the pitch becomes -90 or 90 degrees, and the effect of this is that the yaw axis is collapsed into the roll axis and any yaw movement instead becomes a rotation on the roll or bank axis. In addition to this, we added a similar constraint to the camera's yaw as whenever the yaw reaches 360 , the yaw is set to 0 . Since the yaw is stored as a float, if it reaches large numbers imprecision in the yaw will cause distortion. Keeping the yaw in the boundary will prevent this.

The camera also can move along its forward vector, right vector, and the world's y-axis. It is preferable to move along the world's y-axis instead of the camera's up vector, because while a user knows what direction is forward and left, they are not guaranteed to be aware of where their up direction is. In addition, it is expected that when we wish to translate upwards it is an increase of height relative to other objects, and this is not guaranteed for every direction of the camera. The controls are listed below.

- 'w' - Translate the camera forwards
- 's' - Translate the camera backwards
- 'a' - Translate the camera left
- 'd' - Translate the camera right
- 'spacebar' - Increase the camera's y position
- 'control + spacebar' - Decrease the camera's y position
- 'Up' - Increase camera's pitch
- 'Down' - Decrease camera's pitch
- 'Left' - Decrease camera's yaw

- 'Right' - Increase camera's yaw

Obstacles are rendered in the world as cubes, and any path is rendered as a line segment. They are also stored in different arrays before they are drawn so that way they can be accessed or modified before they are drawn. For both the line segments and the cubes, we can specify the colors we wish to use when we make a call to “draw()”, which draws the entire scene. Figure 77 is an example of a rendering of the world axes, a basic example semi-circle as a path, and a random distribution of obstacles.

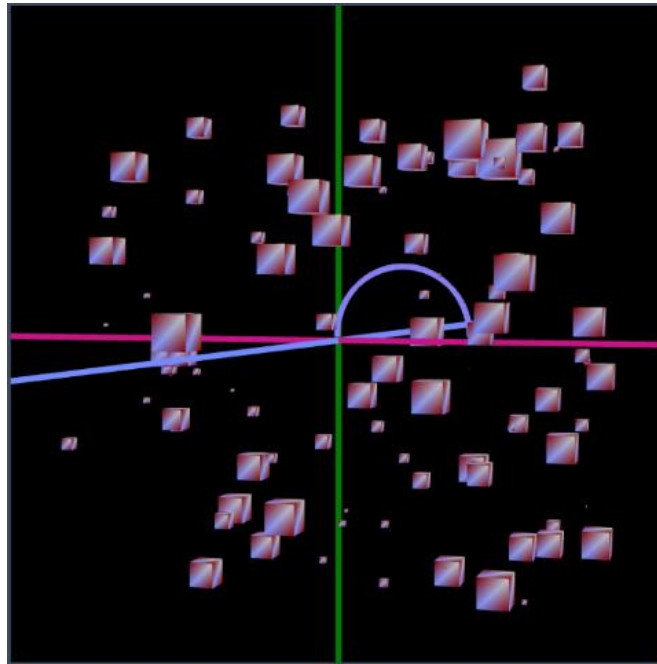


Figure 77 – Example Rendering

Next Steps

The MOLE user interface was developed and tested within the project team. To better understand the functionality of the user interface, a user study should be performed. Having an individual or group of individuals who are not familiar with the MOLE project will help discover flaws in the user interface design that the development team may have missed.

Better visualization of the MOLE as it performs a boring routine is an element currently missing from the system's frontend. Using information about the MOLE's orientation and position, determined through its on-board inertial measurement unit (IMU), a 3D rendering of the MOLE as it moves could be created. 3D object files of the MOLE were created in SolidWorks and can be exported to .stl file format, which can then be read into and drawn-to-screen using WebGL or another browser-based graphics library. In combination with the existing path visualization, a full scene of the mole as it moves along the planned path, with visualizations of drift or deviation from the ideal path, could be created. Another issue with the WebGL rendering of the pathing system is that it lacks a skybox or any sort of background. This can mean that when the user is trying to look around the scene, they can get lost against the default black background of the scene. In addition, an interface with the WebGL should be added to the HTML document like in the Controls page.

The current status visualization represents a one-dimensional array of numeric data. The y-axis encodes to the data values in the array, and the x-axis encodes to the index of that value into the array. This is useful for sensor data that are logical as a function of time, such as the temperature of the boring head, or the pressure in the anchoring air bladders. Other data points, like the acceleration in various directions require a different visualization. Some of these data, such as the boring head RPM, do not need to be viewed in a complex data visualization, as an operator is only interested in the current value, and if it matches up with their set points. Other data, such as acceleration along the 3 spatial axes, may be best served as instantaneous data points, but with a more complex 3-dimensional render of how those values influence one another and how they relate to the MOLE's surroundings.

Server

In addition to the Python Data Layer, the server acts as a main communication gate. The server is programmed in the Node.js environment for a multitude of reasons [26]. A server was chosen as part of the MOLE software as it allows the team to implement the various communication protocols needed to show and process the user interface and communicate with the embedded system. The MOLE required a serial connection, and since it is dynamically communicating with the user, we could not simply compile an instruction set and send it to the MOLE. Because Node.js runs on the Chrome V8 engine, it allows the software to be extremely portable which allowed our team to be able to run the server on all our local machines [26]. Node.js also has a minimal setup time which enabled us to immediately begin work on the design features of the robot. It uses an HTTP server to communicate with the front-end and a TCP server for communicating with the data layer.

The basis for the TCP connection with the Python Data Layer is the “Net” module provided by Node.js [3]4. The reason we used a TCP connection was it allowed us to use the JSON files we receive from the front-end to handle data for communicating with the data layer. JSON files are exceptionally accessible in terms of data types that are communicated across programming languages, so that way we do not have to worry about data type implementation. In addition to this, Node.js does not have an official serial module and we wanted to keep this aspect of the stack using modules that we know will be consistently updated. For instance, we considered using Browserify which would allow us to have access to Node modules on the client side of the applications [23]. One of these modules we had considered was “net-browserify,” and it seemed to solve a few issues with being able to access TCP connections through a browser interface [25]. However, it has recently been archived, and when using custom frameworks or Node packages on long running projects we thought that trying to keep with supported modules would allow teams in the future to be able to add on to the structure of the server without having to refactor components that have become out of date.

One of the key components of the server is its local database. This allows the user to navigate between pages and still retain the data and information from MOLE's operations. There may be situations wherein an operation with this robot may take a few days, and it is important that all the relevant data from the operation is persistent. The database is implemented with ‘lowdb,’ and is stored in JSON format as well [27]. This allows us to quickly save any information from either the client or the data layer without having to worry about type conversion. Moreover, since lowdb uses JSON files this allowed us to import the module ‘json2csv,’ which a module that is frequently maintained by its author [28]. With this module, the user can download a summary of their data in the form of a csv file. CSV files can be opened in almost any spreadsheet program, and because of this it allowed us to maintain the theme of cross-platform portability with our application. Figure 78Figure is the schema of the database.

```

1  {
2    "sentControls": [],
3    "controlsStatusResponses": {
4      "imuAccX": [],
5      "imuAccY": [],
6      "imuAccZ": [],
7      "imuYaw": [],
8      "imuPitch": [],
9      "imuRoll": [],
10     "boringRPM": [],
11     "extensionRPM": [],
12     "drillTemp": [],
13     "steeringYaw": [],
14     "steeringPitch": [],
15     "frontPSI": [],
16     "backPSI": []
17   },
18   "idealPathPoints": [],
19   "sentPaths": [],
20   "obstacles": [],
21   "correctedPathPoints": [],
22   "pathStatusResponses": {
23     "driftX": [],
24     "driftY": [],
25     "driftZ": []
26   },
27   "errors": [],
28   "lastUpdated": 1588826256875,
29   "createdAt": 1588826256875
30 }

```

Figure 78 - Database Schema

As seen above, many of the fields of the database are related to the status of the MOLE itself. The database also keeps track of any requests sent to the MOLE. This allows a user to keep track of the decisions that they made during the operation in case any of that data would be relevant to future operations. In addition to that, it provides future teams with the capability of tracking any operations that they execute for any reason concerning testing. Another important thing to notice about the pathing information is that it is broken up into multiple fields. The first field is “idealPathPoints,” and this is the list of points that is initially generated by pathing module. This is because we wish for the data to be persistent across page navigation, and it also allows us to be able to re-render at any point an earlier path. This is helpful because it allows us to compare different solutions to earlier pathing situations, and because of this we can evaluate previous pathing routes in order to judge their efficiency. For the fields ‘sentPaths,’ and ‘correctedPathPoints,’ we wished to differentiate between what the MOLE following the initially generated path and the corrections that the MOLE must make. This again is a metric that could be a measure of the MOLE's pathing efficiency, and, depending on how well the MOLE can determine environmental factors, could be used to develop a stronger understanding of the field of operation for future operations. Both of the 'sentPaths,' and the 'correctedPathPoints,' contain the end point destination, Euler angle representations of the movement with distance, and the index within the series of movements executed.

The Node.js server uses a typical HTTP server to communicate with the front-end, with the common REST API. This is achieved through Node's HTTP module which provides a variety of API calls that allows for flexible design with the server [35]. Because the user can perform a multitude of actions with the MOLE, and the Data Layer can also respond in multiple ways, the server acts as an API for possible operations. It makes sure that the communication between layers is handled properly and notifies both ends of the stack if there is an error. The typical pattern is that the user sends an HTTP POST request when they wish to interface with the MOLE or the database. The server then checks whether the data is formatted correctly, if it is not the server sends the client a 400 error, and the request is not sent to the Data Layer. A 'type' field which allows us to define our API with a basic comparison to find the proper layout of the data and the correct operation. If the

data is valid the server then processes the data, saves it to the database and sends it over the TCP socket to the python Data Layer. The interaction with the Data Layer is based around the JavaScript Promise API. Node.js cannot return information from an event-handler, so in order to access that information outside of the event-handler, we implemented the interaction with the TCP socket with asynchronous Promises [36]. The received data is then saved to the database and passed up to the user.

At any point the user can send a GET request to the server to ask the database for information. The convention for distinguishing the query to the database and general file requests (HTML, CSS, or JavaScript files for the client side application) is to append the url with '/db/' and then specify what field in the database is required. Like the POST requests mentioned above, if the request is not formatted correctly, then the server will send an error response to the client without passing the information to the Data Layer. If at any point the server receives a request with the JSON type specified as 'error,' the server will save the error message to the database, and then pass the error to the other party. Figure 79 shows the server's protocol in a visual format.

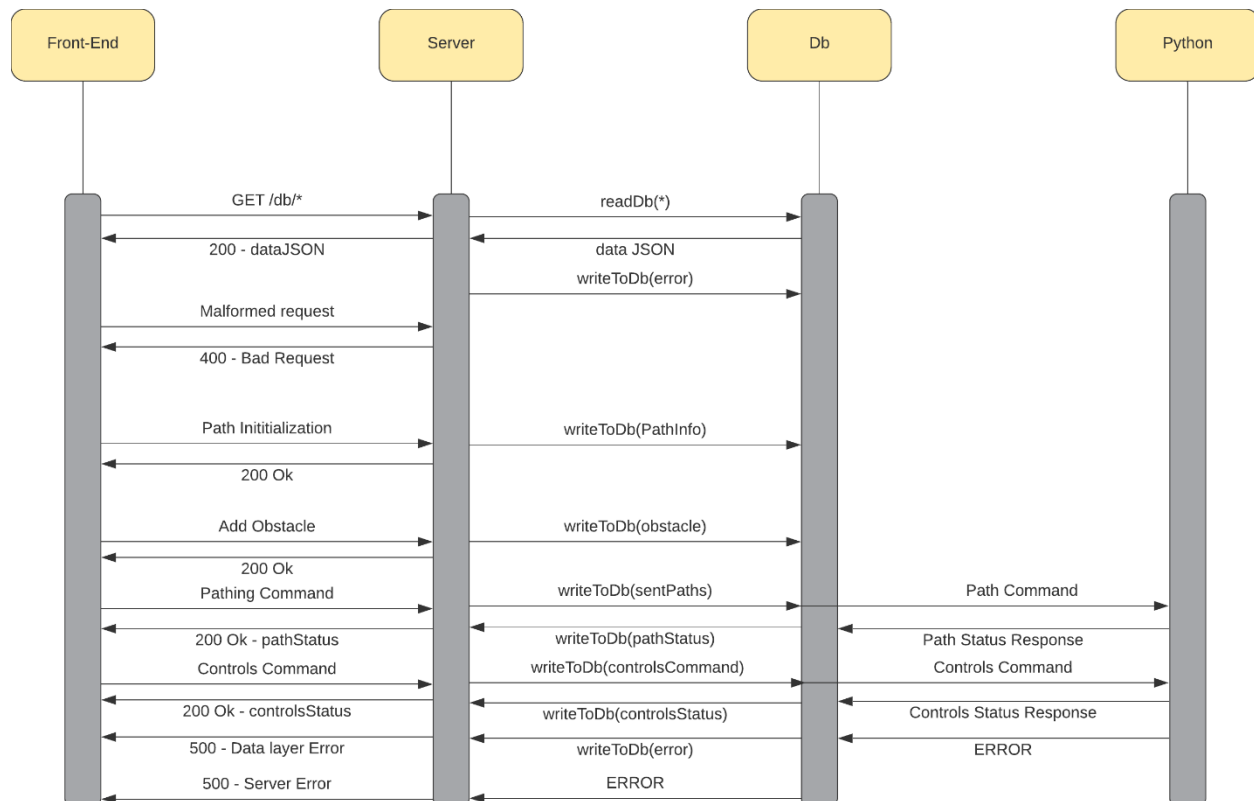


Figure 79 - Protocol Diagram for the Server

Next Steps

The server is one of the more flushed out modules of the stack, but it is still not without its issues. Currently, the API does not need to process most of the data received from the user or the Data Layer. The only API command that has its data altered is the pathing commands, and this is because the MOLE does not need to know the cartesian coordinates nor the index and so they are trimmed from the JSON file. As the MOLE's

complexity grows to incorporate more features to better suit it to its task the API and the server will have to accommodate these changes with different options and actions. In addition to this, the API lacks a behavior for receiving an error except to pass along the error and robot could occur it is difficult to properly define the correct course of action in advance. One other feature that the server (and API in general) lacks is a ‘finished’ or ‘done’ type of command, and this feature will be important to have once the MOLE enters its testing phase.

Pathing

One of the core aspects of the problem that our software is designed to accomplish is to decide how to move the MOLE throughout the underground space it is a part of. A particularly difficult aspect to this problem is that the MOLE has two axes of rotation that it can act on at any time: pitch and yaw. However, the medium of the space is different from traditional pathfinding problems. Typically, a robot is constrained to be able to move along a plane like a normal object in above ground space, and in typical situations like this the robot cannot move along by pitch without the ground changing beneath it. The MOLE is also characterized by its difference from aeronautic pathfinding strategies because it lacks the capability of changing its bank/roll. This limitation of having to move through a 3d space without ground as a reference plane, nor the full six degrees of freedom that air crafts utilize. The MOLE also cannot turn in place and it has a hard limitation on any rotation it makes. What this meant was that the pathing system had to focus on a method to avoid any obstacle with these limitations on movement and be able to get to desired points that the client requires to complete their task.

Since the space is not a simple 2D grid, we could not revert to previous graph or grid-based algorithms such as A* or Dijkstra’s algorithm. Both of these algorithms assume that it is easy to determine the weights for the grid squares or graph edges, but it is not easy to determine what movements are possible or not possible with the MOLE and so that meant we had to focus on determining proper ways to move through the space with our constraints. The space in which MOLE is operating is 3D and is not always going to be of consistent size, so it was appropriate to not represent the world space as a 3D array. So, in order to address this problem, we decided to instead represent the space through mathematical functions because while they were difficult to implement, we never had issues with the representation of the world itself.

Implementation

The first implementation was the idea that we should subdivide a semi-circle by inner angles, and, while this method was abandoned, the idea of using semi-circles as a path “unit” was kept. The issue with this method was twofold. First, the internal angle division does not correspond to the MOLE’s actual turn angle, and because of that we could not ensure that the semi-circle’s path was possible. Second, because the angle was not determined relative to the MOLE, we could not append a second path to the original. This is a design issue for many reasons, but primarily because it prevents any capability of avoiding any obstacle and especially those that have not been discovered on the path yet. However, the idea of creating semi-circles as path units did stay until the final design.

The second implementation of the curve generation algorithm was closer to our current implementation, but it was still riddled with some of the same issues that occurred in the previous implementation. They were both developed around the same time, but we were unsure of the validity of either implementation, and so we tested both and the results between these two methods were wildly inconsistent with each other. This second implementation still had the same issues as before, because there was no clear way properly append new paths to a generated one. Both implementations were also incredibly difficult to translate to the 3D space in which we were working. However, the equation for this implementation was still valuable, and some of the later equations we used were related to this one. So, the equations for the first and second implementation are shown

below. Note that “ θ ” refers to the maximum turning angle that the MOLE can accomplish, and “ d ” refers to the maximum extension distance the robot can achieve.

Angle Subdivision Implementation

$$\theta = \frac{180}{\vartheta}$$

$$x = r \cos(\theta)$$

$$y = \sqrt{r^2 - x^2}$$

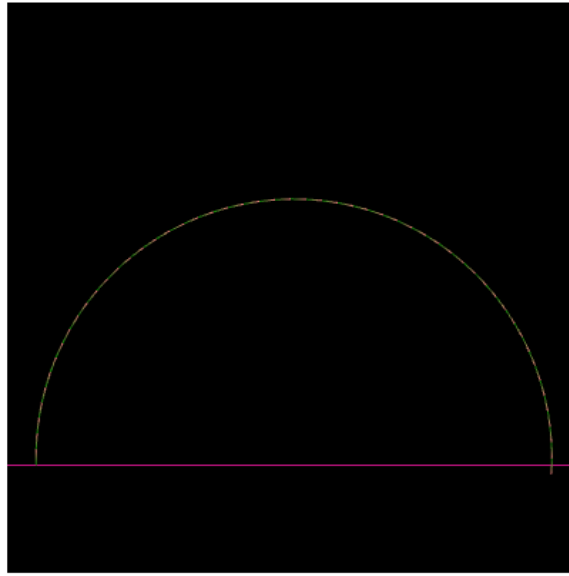


Figure 80 - Angle Subdivision where $\vartheta = 3.5$

Incremental Angle Implementation

$$a = \frac{90}{\vartheta}$$

$$x_n = \sum_{n=0}^{n=a} d \cos(90 - n\vartheta) + x_n - 1$$

$$y_n = \sum_{n=0}^{n=a} d \sin(90 - n\vartheta) + y_n - 1$$

The third and final major implementation of the problem is the one we ended up using because, unlike the other two implementations, the movement is localized to the line of sight of the robot itself. While the second implementation does not require a known radius, unlike the first, dependent upon the angle relationships of a circle. The third implementation is conceptually based around the robot’s line of sight, and because of that the path is not limited to simple semi-circles and can be used to move through the world in very complex paths. We continued with semi-circles for this initial phase of the project, but this is extendable to a multitude of contexts.

First, at any time if we know our previous and current locations, we can determine our line of sight as a vector that is found from the difference between these two points. With this vector we can rotate it to any location using a series of yaw and pitch rotations. This rotation is accomplished by first extending the vector by the distance of the movement we wish to perform, and then multiplying it by a rotation matrix in order to find the next point. This vector then is appended to the previous vector. The 2D implementation of this is shown below. d represents distance, v represents the line-of-sight vector (the intended direction without changing the MOLE's turning), x and y represent the new location coordinates.

$$u = ||v||$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} u_x \\ u_y \end{bmatrix}$$

$$x = du_x + x_0$$

$$y = du_y + y_0$$

To determine a direction vector from yaw and pitch, we use the transformation below.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \cos(yaw) \cdot \cos(pitch) \\ \sin(pitch) \\ \sin(yaw) \cdot \cos(pitch) \end{bmatrix}$$

However, since the rotations are different for 3D, we took this concept and used the Euler angles of yaw and pitch. This allows us to not have to start on the X-Y axis and branch out to other orientations, but there are limitations to this method we will discuss soon. The conversion of the yaw and pitch to a unit vector is the same way in which we update the front of the camera when we change the pitch or yaw [24].

Through this method, we rendered a view of five possible movements a path could take, where at each point the robot could increase pitch, decrease pitch, increase yaw, decrease yaw, or simply head straight. The renders of this method are shown in Figure 81 - Rendering of all paths with 10° turns and Figure 82.

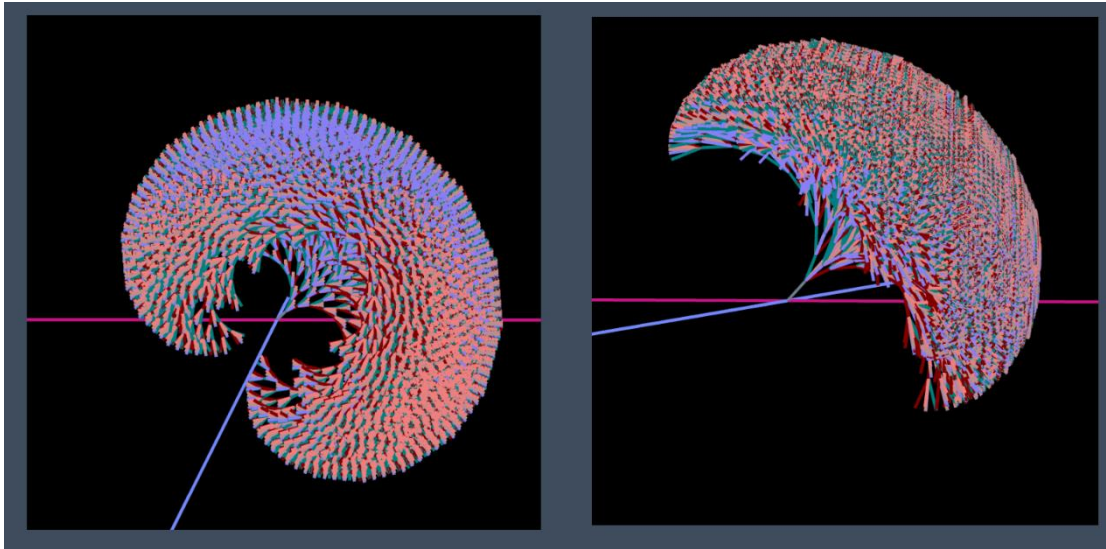


Figure 81 - Rendering of all paths with 10° turns

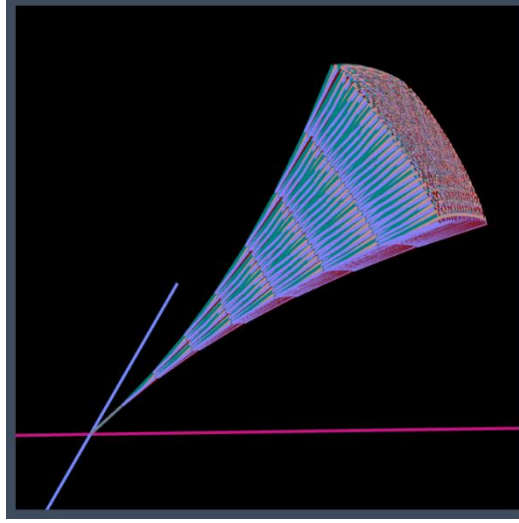


Figure 82 - Rendering of all paths with 3.5° turns

While these renders are incredibly informative about the importance of the turning angle and why an increase in the maximum turning angle would lead to a considerable improvement in the MOLE, the computation for this calculation is extremely expensive due to it being a recursive function. Discounting matrix multiplication in this process, the rendering of all possible paths is $O(5^n)$. If we do count the matrix multiplication, then this render is $O(80^n)$ where n is the number of iterations.

One of the last major problems we solved was how to append paths to each other, and this is one of the key features of the MOLE's software if in the future the MOLE can correct its own path. The first step in merging two curves is being able to detect the change in yaw and pitch from turn to turn, so that we can validate if two curves are capable of merging. The equations used to determine these values are shown below.

$$u_1 = ||v_1||$$

$$u_2 = ||v_2||$$

$$yaw = \arctan2(u_z, u_x)$$

$$pitch = \arcsin(u_y)$$

$$\delta yaw = u_{2yaw} - u_{1yaw}$$

$$\delta pitch = u_{2pitch} - u_{1pitch}$$

In addition to these equations to show the change in yaw and pitch, we also needed to be able to determine whether any arc will collide with a known obstacle. Obstacles are represented as boxes in the world space to simplify collision detection. The collision detection is implemented with Plücker coordinates which are a reliable way to determine when a line segment will intersect with a plane, and since there are six planes that define a cube we need to perform this calculation six times per segment [30]. First given three points, we need to both determine the fourth point of the bounding square and the plane coordinates that represent that face of the cube. The fourth point can just be determined by $O = P - Q + R$, where P , Q , and R are in counterclockwise orientation. When a bounding box for an obstacle is drawn, it is represented as twelve triangles, which all follow this counterclockwise orientation.

$$S = P - Q$$

$$T = R - Q$$

$$[a, b, c] = S \times T$$

$$d = -ax - by - cz$$

In order to find d , you can simply plug in one of the three coordinates used to determine the other plane coefficients. The Plücker coordinate representation of the plane is $[N:n]$, where N represents a tuple containing a , b , and c , while n is the coordinate d [30]. In order to represent line segment in Plücker coordinates, an additional amount of vector math is necessary. A line in Plücker coordinates is defined by $\{U: V\}$ where, $U = A - B$, and $V = A \times B$, where A and B are the points defining the line segment [30]. From these two coordinate sets, the intersection between the line and the plane is determined by $(V \times N - nU: U \cdot N)$ where this tuple is a set of homogenous coordinates [30]. The Cartesian coordinates are then found by dividing the three coordinates that define the first tuple of this representation by the coordinate that is the second tuple. This gives us a coordinate $[x, y, z]$ where the line intersects the plane. However, this does not check if the line intersects the plane within the given triangle or square. In order to do that, we need to check if the intersection point is within the range specified by the coordinates defining the cube. Finally, the intersection point can still be outside the points defining the line segment. In order to test for this, let AC be the vector between the intersection point and the furthest point, and AB be the vector defined by the furthest point and other defining point of the line segment. The following conditions must be true if the point is in-between both ends of the line segment [31]:

- $\|AC \times AB\| = 0$
- $AC \cdot AB > 0$
- $(A_x - B_x)^2 + (A_y - B_y)^2 > AC \cdot AB$

Any time a line intersects an obstacle, unless the obstacle is at the start or end of the line, it must have an exit point. We test for collision over the entire series of line segments and note the two line segments that enter and leave an obstacle. The line segments help us to define where on the line the obstacle lies. From this we use the equation defined below to determine the distance of the endpoints of these two line segments.

$$d = \sqrt{2r^2(1 - \cos \theta)}$$

From this we find the distance that needs to be traveled by MOLE every time it extends in order to reach the desired radius. The radius between these two points can be found by the distance between these two points, and with the radius we can solve the equation for the distance. These provide us enough information to construct a curve before we specify yaw and pitch. From this we can construct three curves, all three with the same change in pitch, but the difference between the three curves is that the yaw values for the curves are the zero, and the negative and positive max yaw values that the robot can turn on.

The way the path correction works for appending a dynamically created curve around an obstacle the mole will collide with is that we check the change across the four points forming the connecting line between both curves, and from here will be referred to as P0, P1, P2, and P3. P1 and P2 lie on the initial curve, and P3 and P4 lie on the newly generated curve. In order for the curves to be able to merge, the lines bounded by P1, P2, P3 and P2, P3, P4 must change their yaw and pitch within the turning range of the MOLE. It is clear that not every two curves can be merged together at desired points, and so the way to deal with that is to iterate over the initial curve and test sets of line segments until we either have no more points to test on the inner curve or have found a proper place to merge the two curves. Figure 83 is a visualization of this testing method.

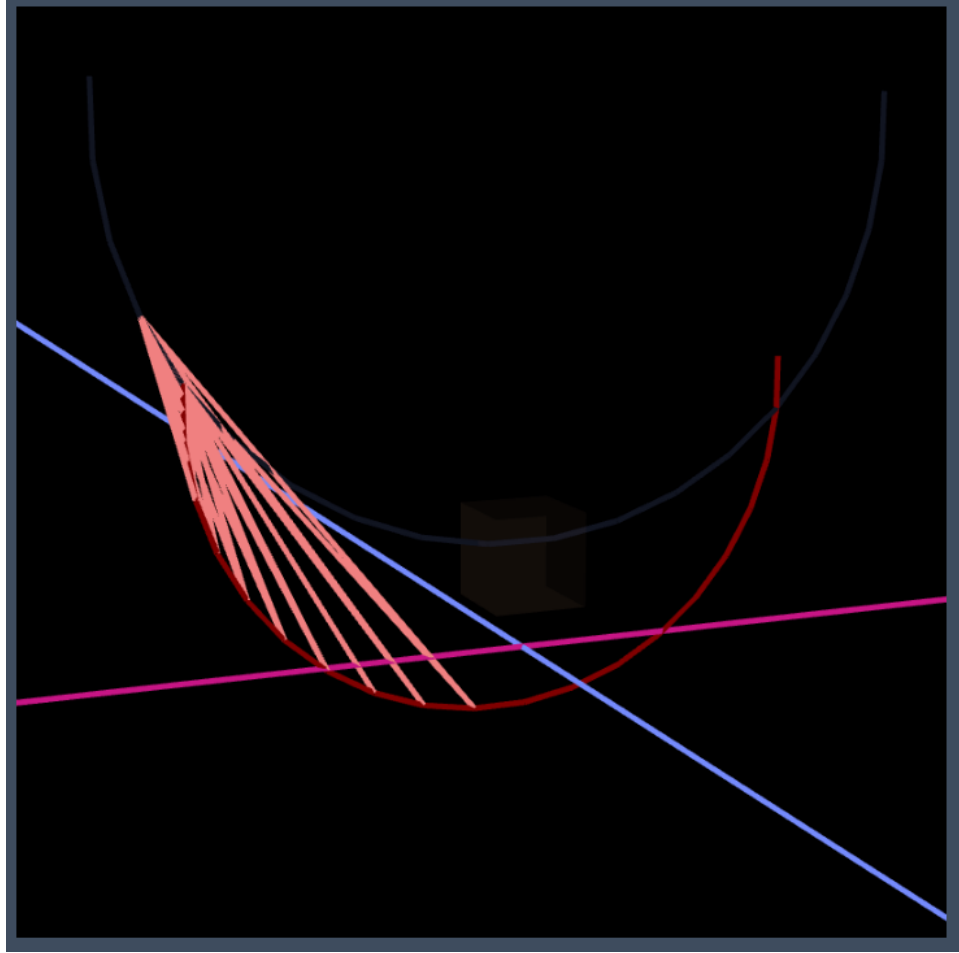


Figure 83 - Yaw and Pitch Testing

If there are no valid merge points on the inner curve, we test the next set of points along the initial curve, slowly expanding our radius. Figure 84 shows the series of smaller curves that can be drawn from the initial curve. This solution is a brute-force method as it checks every combination and will need to be improved upon in the future.

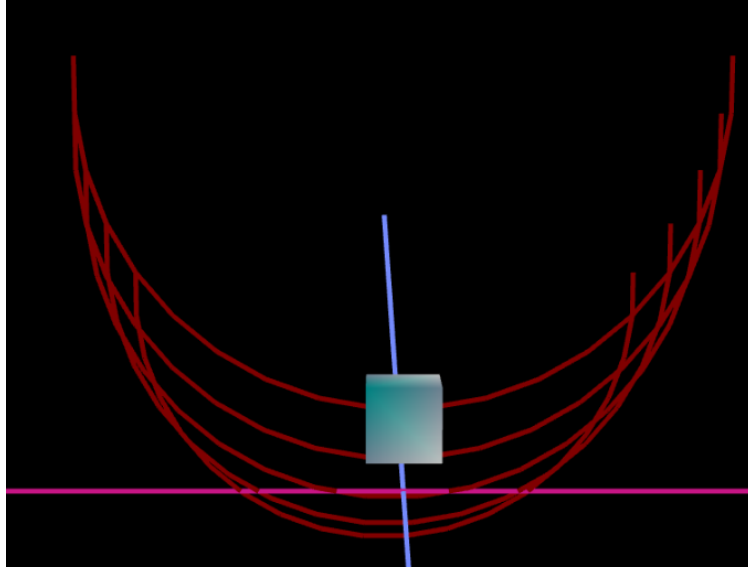


Figure 84- Drawing Sub-curves

However, once two curves are deemed as valid to merge, we treat the fully merged line as a series of splines and use a Centripetal Catmull-Rom Spline to smooth out any possible sharp turns that may remain within the curve. A Catmull-Rom Spline is a form of a Cubic Hermite spline that was developed by the leading engineers at Pixar for the specific purpose of developing a way of producing smooth curves [32]. Initially, we wanted to use Bezier curves to merge series of points together, but Bezier curves' slope is hard to control and can react strongly to a change in a control point causing the slope to vary wildly.

The Centripetal Catmull-Rom Spline is formed by taking a series of four controls points that the curve must pass through and interpolating across these using a knot formation. The equations defined for the interpolation across $P1$ and $P2$, are defined as follows [33].

$$t_0 = 0$$

$$A_1 = \frac{t_1 - 1}{t_1 - t_0} P_0 + \frac{t - t_0}{t_1 - t_0} P_1$$

$$A_2 = \frac{t_2 - t}{t_2 - t_1} P_1 + \frac{t - t_1}{t_2 - t_1} P_2$$

$$A_3 = \frac{t_3 - t}{t_3 - t_2} P_2 + \frac{t - t_2}{t_3 - t_2} P_3$$

$$B_1 = \frac{t_2 - t}{t_2 - t_0} A_1 + \frac{t - t_0}{t_2 - t_0} A_2$$

$$B_2 = \frac{t_3 - t}{t_3 - t_1} A_2 + \frac{t - t_1}{t_3 - t_1} A_3$$

$$C = \frac{t_2 - t}{t_2 - t_1} B_1 + \frac{t - t_1}{t_2 - t_1} B_2$$

$$t_{i+1} = \left(\sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2 + (z_{i+1} - z_i)^2} \right)^\alpha + t_i$$

$\alpha = 0.5$ for centripetal Catmull-Rom splines

$\alpha = 0$ for chordal Catmull-Rom splines

$\alpha = 1$ for uniform Catmull-Rom splines

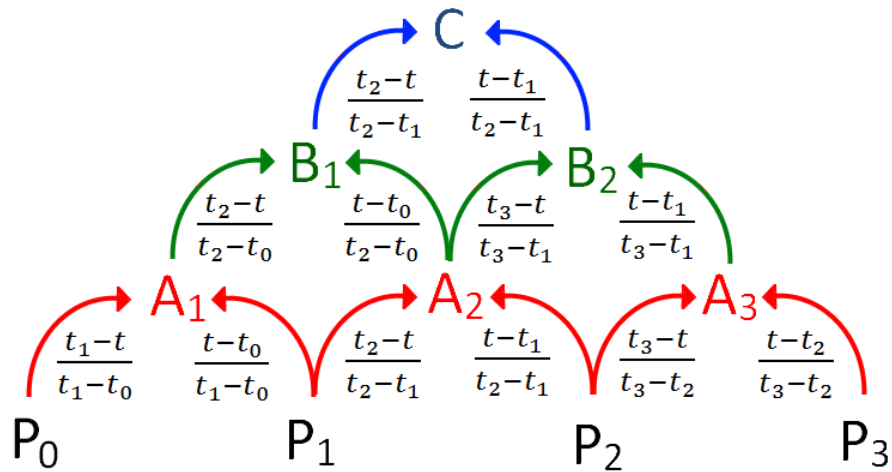


Figure 85 - Knot formulation [32]

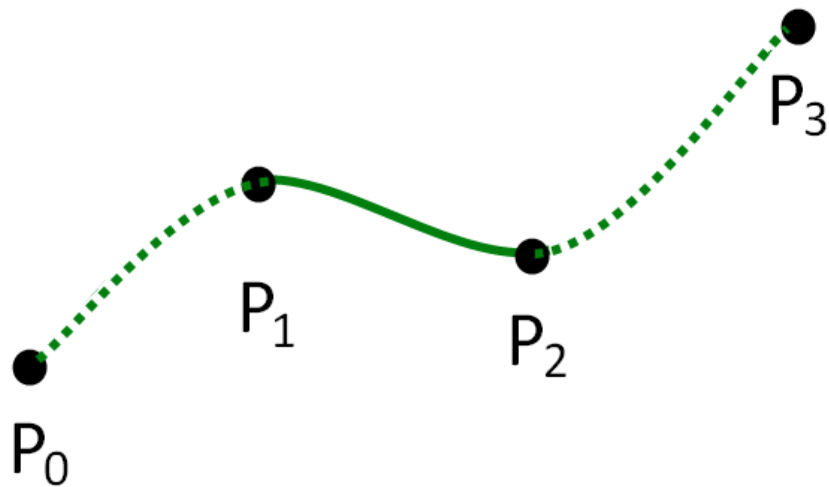


Figure 86 - Catmull-Rom Spline [33]

We use this spline across the entire series of points of the two fully merged curves to produce an exceptionally smooth curve. For example, Figure 87 shows the result of the pathing correction algorithm from the equations above and Figure 85 and Figure 86.

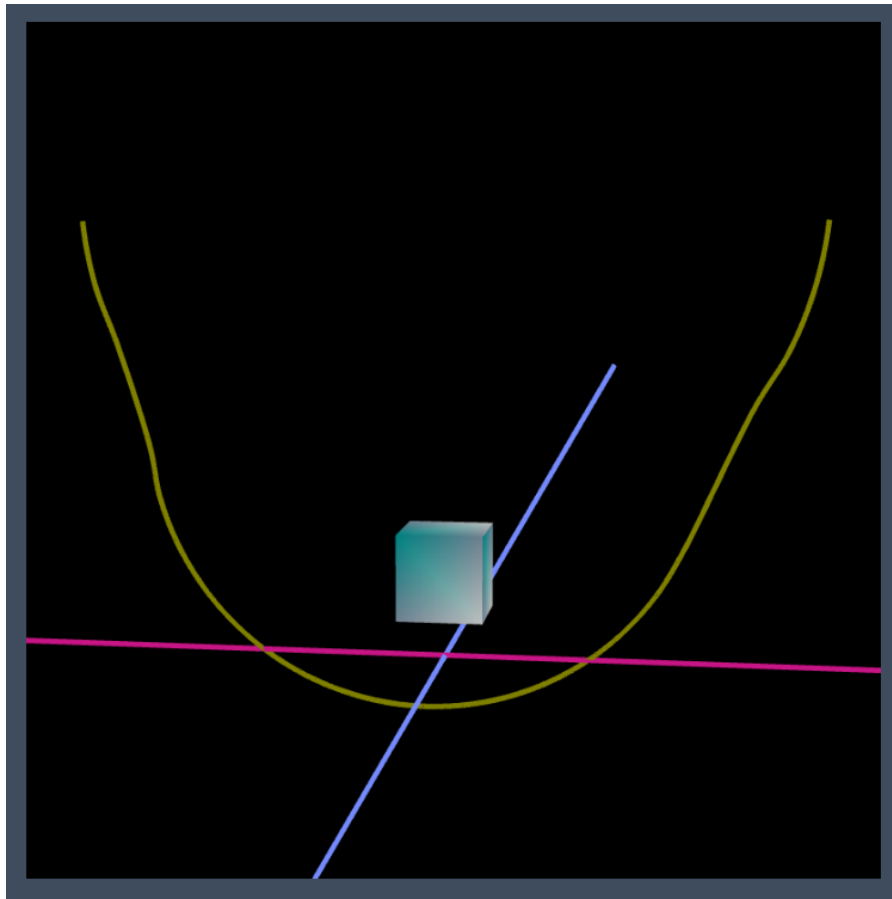


Figure 87- Result of Path Correction and Centripetal Catmull-Rom Spline

Next Steps

This design of navigation through the world is far from perfect and needs to be improved in several areas. First, Euler angles will not continue to suffice for these operations as they tend to be inconsistent with calculations and require many matrix operations to utilize effectively. Moreover, they are prone to Gimbal-lock as discussed above [29]. Our recommendation is to switch to a Quaternion based rotation system to both minimize the possible errors with Euler angles, and so that way the next team may implement as robust a form of this navigation system as they desire.

Second, while the merging two curves algorithm works, it is a brute-force solution, and thus is not a satisfactory solution to the unique issue presented by this robot. Instead, the next team should still utilize the Catmull-Rom Spline, but for it to become more useful, there needs to be a way to meaningfully discern proper control points for the path. This could even mean discarding the semi-circle idea altogether.

Third, a full control loop with dynamic path correction and generation still needs to be implemented, at the current state most of the pieces are assembled. However, in order for the robot to be fully functional, the robot needs to be able to both generate a path with a start and end point that navigates around any known obstacles in a scene and not just one obstacle.

Finally, one of the most important aspects of the pathing system that was not developed was a proper way to test or validate the functions and data we are generating. Since it was crucial for this iteration to demonstrate

a proof of concept and lay out a pathway to a solution to begin with, the validation and testing was not a feasible avenue to pursue. Upon further iterations of the pathing system, comparisons with other known pathing methods could be a fruitful examination.

Data Layer

A Python-based data translation and communication layer lies between the Node.js server and frontend, and the software running on the embedded system on-board the MOLE. This component acts as a middleware, freeing up resources on both the server and embedded system by handling message translation and ensuring data completeness. The data layer communicates upstream with the Node.js server via a socket connection and communicates downstream with the embedded system over a serial port connection.

Downstream Communications

The Python data layer communicates with the Node.js server over a socket connection. Both of these components can build and parse JSON objects and send JSON strings over this socket connection to communicate. When an operator sends a request containing controls instructions to the server via the controls UI (described above), a JSON object which encodes the control fields is created and sent downstream to the data layer. This JSON object contains seven fields, summarized in Figure 88.

```
{
  type: 'controls'
  boringspeed: int
  extensionRate: int
  inflateFront: boolean
  inflateBack: boolean
  turningX: float
  turningZ: foat
}
```

Figure 88 - Controls JSON Message Format

These fields are received as a JSON string and parsed into a Python dictionary using the `json.loads()` function from Python's JSON library. A similar process takes place when the data layer receives a path message. These messages have a simpler format, with only five fields, shown in Figure 89.

```

{
    type: 'path'
    yaw: float
    pitch: float
    roll: float
}

```

Figure 89 - Path JSON Message Format

After parsing these messages into Python dictionaries so they can be handled by the data layer itself, the translation into an Arduino compatible format is performed. When specifying the format of the Arduino message, the team wanted to ensure a format which was lightweight, extensible, and robust. Although the JSON string format could be pushed directly to Arduino over the serial connection, that format makes liberal use of whitespace, and is not as efficient as possible. The team specified a simple, yet robust format for communicating with the Arduino: comma-delimited strings with the appropriate fields in the order they appear in the respective JSON string, terminated by a ‘!’ character. These messages also encode the message type as a single character: 0 for controls, 1 for path, and 3 for error, to increase efficiency. The Arduino script running on the embedded system reads these messages character-by-character and has the ability to quickly parse messages or handle malformed messages. Once the message is transformed from a JSON string to an ‘Arduino string’ they are passed over a serial connection in a call to `Serial.write()` from the `pyserial` Python library. After sending a message downstream to the embedded system, the data layer waits for a response from it, which will contain status information, or, in the event of an error, error diagnostic information. Figure 90 below shows the information flow through the different modules.

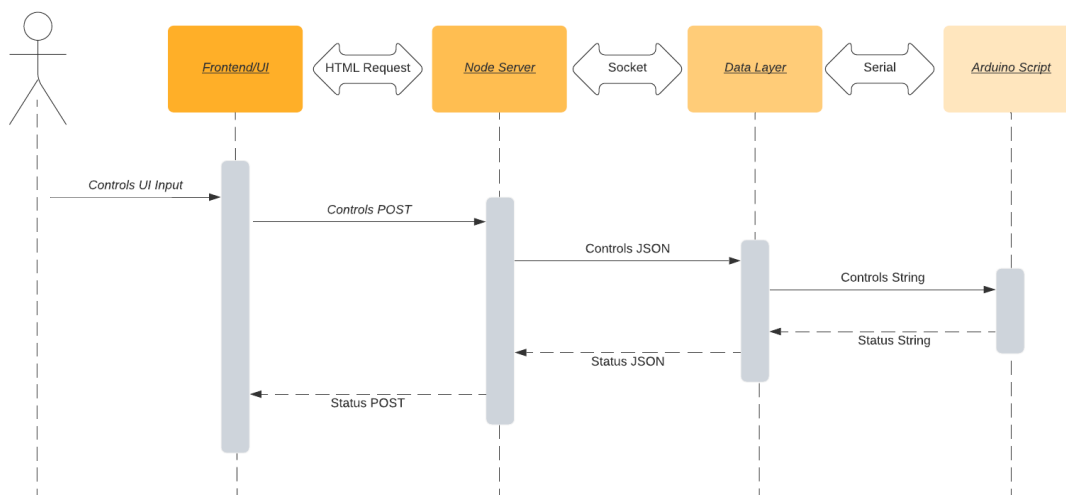


Figure 90 - Main Module Communication Loop

Upstream Communications

After a message is sent downstream to the embedded system, the data layer waits for a response using `serial.readline()`, again from the `pyserial` library. Messages sent up to the data layer from the embedded system

take a similar format to the messages sent from the data layer down to the embedded system. Again, status messages are encoded as comma-delimited strings.

After processing a manual control type message, the embedded system will compile all available sensor data into a comma-delimited string, with a type of 'status' prepended. This message is then sent over the serial connection to the data layer.

Upon receiving a 'status' type message from the embedded system, the data layer checks the message for completeness, ensuring that the correct number of values (14) are present, as these strings contain only data values and a type field, no other field indicators or headers. If the message passes the correctness check, it is parsed into a Python dictionary with 14 fields, which hold various sensor readings from the MOLE.

```
{
  'type': 'status'
  imuAccX: float
  imuAccY: float
  imuAccZ: float
  imuYaw: float
  imuPitch: float
  imuRoll: float
  boringRPM: int
  extensionRate: int
  drillTemp: float
  steeringYaw: float
  steeringPitch: float
  frontPSI: float
  backPSI: float
}
```

Figure 91 - Status JSON Message Format

The resulting Python dictionary is compiled into a JSON string using `json.dumps()`, which is then sent to the Node.js server via the socket connection. The full format of this JSON object is shown in Figure 91 above.

When the data layer processes and sends a 'path' type message to the embedded system, it performs a similar process, but expects a different message format. Upon receiving a 'pathStatus' type message from the embedded system, the message is parsed and the number of values (3) is checked. If the correctness check is passed, the 'pathStatus' message is parsed into a Python dictionary with four fields, summarized in Figure 92.

```
{  
    type: 'pathStatus'  
    driftX: float  
    driftY: float  
    driftZ: float  
}
```

Figure 92 - pathStatus JSON message format

The type field indicates to the server what this message contains, while the other fields indicate how far off the ideal path the MOLE has deviated. The resulting Python dictionary is compiled into a JSON string and sent upstream to the Node.js server over a socket connection.

Error Handling

If, at any time during the data layer's main communication loop, a message fails to send, or an incorrect message is received, the system will transition into an error handling mode. In the event of a failed communication with the embedded system, the data layer will attempt to re-establish connection and send an emergency stop message, which triggers a routine shutting down all motors and pumps. Additionally, when an error of this type occurs, an 'error' type JSON message, with the format shown in Figure 93 is sent upstream to the Node.js server.

```
{  
    type: 'error'  
    message: string  
}
```

Figure 93 - Error JSON message format

The type field indicates to the server that an error has occurred, and the message is printed to the user, indicating that there was an error communicating with the embedded system, and that the system is now in an error state.

Next Steps

As the MOLE functionality changes to add more abilities or sensors, the data layer will need to be updated to match newly defined message formats. This may take the form of entirely new message 'types,' such as the result of an on-board sensor mapping out the area around the MOLE. These changes could also take the form of adding new fields to the existing 'status' or 'pathStatus' messages, if a new sensor is added, or the precision of one is improved such that a floating point number is no longer precise enough.

The system's error headline is barebones, and simply stops the robot's operations and alerts the user when an error state is detected. More graceful error handling and recovery is an area the system could expand on. Replacing the simple error message format, which currently only contains a string to describe the error, with a more comprehensively defined system of error states or status codes could improve the system's ability to recover from errors. For example, if the system detected an error with the serial connection to the embedded

system, that error code could be processed by the server to advise the operator of a disconnect, while the data layer runs a routine which attempts to re-establish the connection.

Finally, the data layer is written for and runs on Python 2.7. During the MOLE project lifecycle, Python 2.7 reached end-of-life. The data layer will need to be updated for compatibility with Python 3.7 or later versions. Additionally, Python 3.7 versions of included libraries will need to be utilized or alternatives will need to be found or developed. The software will continue to run on Python 2.7, and Python 2.7 will continue to be available for download from the Python Foundation, but the software must be updated to Python 3.7+ to avoid potential security vulnerabilities in the no longer supported version of Python [22].

Embedded System

The final software component is an Arduino script running on the Teensy 3.6 platform. This module has three main responsibilities. First, it must parse messages coming from the data layer over serial connection and take action corresponding to the message contents. Second, it must interface with hardware components to drive motors, run pumps. Finally, it must read various sensors and adjust the system according to their values.

Communication

The Arduino script listens for messages over a serial connection with the Python data layer described above. This serial connection operates at a 9600 bits/second, one of the suggested values when using the Teensy 3.6 platform. To prevent message parsing from slowing down system operations when there is no message to be received, a call to `Serial.available()` is made to determine if there is a message waiting to be read.

Messages are parsed one character at a time, using `Serial.read()` to build up an Arduino/C++ String object, rather than a C-style character array. After the value is read in its entirety, (determined by reaching the comma delimiter or the '?' end of message character) the String message fragment is converted to the appropriate data type, using `atof()` to convert the Strings to an floating-point value that the string represents. Although the inflation of the front and rear modules are represented as a binary inflated or deflated, converting all the message fragments to floats keeps the processing in a more general form, and the storage of 2 extra floating point values has no meaningful impact on the usage of system memory. After parsing the entire message, a function is called which runs the routine to set various control setpoints. This functionality was initially developed as a series of stub functions with only prints to the serial communication line or blinking LED as validation that the controls were parsed correctly. As the system developed, and motor controllers and other components were acquired and installed, the stub framework made for easier integration of communication code with hardware control code.

Conclusions

This group succeeded in creating what could be called a prototype but should more accurately be called a “full-stack proof of kinematic concept.” The background research and design process were a significant portion of the project and the mechanical team proposed that materials and linkages of sufficient strength would not be possible with the given budget. Because of this, the present assembly has three primary purposes: (1) it can be used to validate that the motion is generated correctly, (2) it provides a full-stack foundation in which any portion of the stack can be improved upon, and (3) the prioritization on modularity means that retrofit is an easy task. The team accomplished the above three objectives, and successfully created a robot to serve as a kinematic proof of concept.

The current platform does not function in soil, but there is no hypothetical reason that the platform the current team has provided would be unable to be retrofitted to operate in soil. The specific form of retrofit will depend on the nature of the next team to take over the project. A mechanical team could focus on any of the individual systems or could focus on a single upgrade, such as creating a fully metal chassis or replacing the electrically driven systems with hydraulics or off-the-shelf actuators. An electrical team, depending on their specialty, could introduce a dynamic sub-surface sensor system for detecting obstacles and pinpointing the system's location underground. A computer science team could create a much more robust front end with full integration of pathing, potentially with live path-generation around detected obstacles. Specific next steps can be found in the Next Steps sections throughout the report. Regardless of the direction the project takes in the future, the team looks forward to a viable technology for this completely unfilled niche and wishes the best of luck to the next team to take on the mantle.



Bibliography

- [1] M. . Doula and A. Sarris, Soil Environment. Germany: Bartens, 2016, pp. 213–286.
- [2] T. Ma, P. Chen, and J. Zhao, “Overview on vertical and directional drilling technologies for the exploration and exploitation of deep petroleum resources,” vol. 4, no. 2, pp. 365–395, Dec. 2016, [Online]. Available: <https://link.springer.com/article/10.1007/s40948-016-0038-y>.
- [3] J. Rostami and S.-H. Chang, “ScienceDirect,” no. 10, pp. 125–126, 2017, [Online]. Available: <http://lib.cqvip.com/qk/85265X/201710/72747566504849554948484955.html>.
- [4] M. Bostock, *D3.js - Data Driven Documents*, 2019. [Online], Available: <https://d3js.org/>.
- [5] “Life cycle of a soil.” <https://museum.isric.org/content/themestation/life-cycle-soil/landscape2>.
- [6] “Auger Boring - Frequently Asked Questions.” Earth Boring Company Limited. <http://66.226.137.91/faqs/auger-boring/>
- [7] D. Watson et al., “Raiders of the Lost Mud: the geology behind drilling incidents within the Balder Formation around the Corona Ridge, West of Shetland ,” 2/7/19.
- [8] J. C. Obi, “Geophysical Imaging of Karst Features in Missouri,” Doctoral Dissertations. Missouri University of Science and Technology, 2016.
- [9] “Quarrying and the Environment: Caves and Karst,” 2017. http://www.bgs.ac.uk/mendips/caveskarst/karst_3.htm.
- [10] B. Singh and R. K. Goel, Chapter 22 - Rock Drillability. Engineering Rock Mass Classification. Boston: Butterworth-Heinemann, 2011, pp. 287–292.
- [11] E. Cooley, F. Madison, D. Frame, and A. Wunderlin, “Mapping Bedrock: Using models to determine bedrock depth across a large area,” University of Wisconsin, 2013. Accessed: 10/14/19. [Online]. Available: <http://www.uwdiscoveryfarms.org/UWDiscoveryFarms/media/sitecontent/PublicationFiles/mappingbedrock/Using-models-to-determine-bedrock-depth-factsheet.pdf?ext=.pdf>.
- [12] T.-T. G. & C. KG, ““GRUNDOMAT,” 2019. <https://www.tracto-technik.de> (accessed Oct. 15, 2019).
- [13] J. B. B. Ucgul Jacky M. A. Desbiolles, John M. Fielke, Mustafa, “Development and field evaluation of a high-speed no-till seeding system,” no. 194, Jul. 2019, [Online]. Available: https://www.researchgate.net/publication/33gure94560_Development_and_field_evaluation_of_a_high-speed_no-till_seeding_system.
- [14] I. T. Association, “The drillability assessment of rocks using the different brittleness values,” vol. 26, no. 2, pp. 406–414, Mar. 2011, [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0886779810001677>.
- [15] S. Gleason, “Slanted Oil Wells.” Popular Science. May 1934.
- [16] S. N. Warren, R. R. Kallu, and C. K. Barnard, “Correlation of the Rock Mass Rating (RMR) System with the Unified Soil Classification System (USCS): Introduction of the Weak Rock Mass Rating System (W-RMR),” vol. 49, no. 11, pp. 4507–4518, Nov. 2016, [Online]. Available: <https://link.springer.com/article/10.1007/s00603-016-1090-1>.

- [17] S. Alam, E. N. Allouche, C. Bartlett, A. Sherpa, and B. Keil, “Experimental Evaluation of Soil-Pipe Friction Coefficients for Coated Steel Pipes,” 2013, pp. 360–371, doi: 10.1061/9780784413012.034.
- [18] J. R. Allen and N. P. Zembillas, “Power Engineering,” Penn Well, Sep. 2013, [Online]. Available: <http://www.tshtbe.ca/download/PowerEngSept2007.pdf>.
- [19] B. V. Sudnishnikov, A. D. Kostylev, and K. K. Tupitsyn, “Pneumatic boring tools in constructional engineering and mining,” vol. 6, no. 2, pp. 165–168, Mar. 1970, [Online]. Available: <https://link.springer.com/article/10.1007%2F02502153?LI=true>.
- [20] C. J. Yang, W. D. Zhu, W. H. Zhang, X. H. Zhu, and G. X. Ren, “Determination of Pipe Pullback Loads in Horizontal Directional Drilling Using an Advanced Computational Dynamic Model,” vol. 140, no. 8, Jul. 2014, [Online]. Available: https://www.researchgate.net/publication/264041657_Article_Determination_of_Pipe_Pullback_Loads_in_Horizontal_Directional_Drilling_Using_an_Advanced_Computational_Dynamic_Model.
- [21] “ASTM D2487 Unified Soil Classification System - ANSI Blog,” 2018. <https://blog.ansi.org/2018/03/unified-soil-classification-astm-d2487-17/> (accessed Oct. 14, 2019).
- [22] P. Foundation, *Sunsetting Python 2*, n.d., [Online]. Available: <https://www.python.org/doc/sunset-python-2/>.
- [23] Browserify.org, *Browserify*, n.d., [Online]. Available: <http://browserify.org>.
- [24] J. de Vries, ‘LearnOpenGL - Camera’, 2014. [Online]. Available: <https://learnopengl.com/Getting-started/Camera>
- [25] S. Ser, ‘emersion/net-browserify,’ *Github*, 2016, [Online]. Available: <https://github.com/emersion/net-browserify>
- [26] OpenJS Foundation, ‘Node.js’, *Node.js*, n.d., [Online]. Available: <https://nodejs.org/en/>
- [27] Typicode, ‘typicode/lowdb’, *Github*, 2019, [Online]. Available: <https://github.com/typicode/lowdb>
- [28] Z. Mirco, ‘zemirco/json2csv’, *Github*, 2020, [Online]. Available: <https://github.com/zeMirco/json2csv>
- [29] A. Popa, ‘What is meant by the term gimbal lock’, *MadSci Network: Engineering*, 1998, [Online]. Available: <http://www.madsci.org/posts/archives/aug98/896993617.Eg.r.html>
- [30] T. Jones, ‘Introduction to Plücker Coordinates’, *flipcode*, 2000. [Online]. Available: https://www.flipcode.com/archives/Introduction_To_Plcker_Coordinates.shtml
- [31] C. Ka, ‘How can you determine a point is between two other points on a line segment’, *Stack Overflow*, 2008. [Online]. Available: <https://stackoverflow.com/questions/328107/how-can-you-determine-a-point-is-between-two-other-points-on-a-line-segment>
- [32] E. Catmull and R. Rom, ‘A Class Of Local Interpolating Splines’, *Computer Aided Geometric Design*, pp. 317-326, 1974.
- [33] P. Barry and R. Goldman, ‘A recursive evaluation algorithm for a class of Catmull-Rom splines’, *Proceedings of the 15th annual conference on Computer graphics and interactive techniques – SIGGRAPH 88*, 1998.
- [34] ‘Node.js v14.2.0 Documentation’, *Net | Node.js v14.2.0 Documentation*. [Online]. Available: <https://nodejs.org/api/net.html>
- [35] ‘Node.js v14.2.0 Documentation’, *HTTP | Node.js v14.2.0 Documentation*. [Online]. Available: <https://nodejs.org/api/http.html>

- [36] 'Node.js v14.2.0 Documentation', *Events | Node.js v14.2.0 Documentation*. [Online]. Available: <https://nodejs.org/api/events.html>
- [37] Metcalfe, J., Explore Seattle's Bertha Tunneling Machine In Awesome 360 Video. [online] CityLab. 2017. Available at: <<https://www.citylab.com/life/2017/03/take-a-360-degree-dive-with-seattles-massive-tunnel-borer/519447/>>
- [38] Szondy, David, Hard grind: The epic journey of the world's biggest tunnel boring machine [online] May 01, 2017 Available at: <https://newatlas.com/bertha-boring-machine-seattle/48862/>
- [39] RMPS Solutions, Directional Drilling, 2018 , available at: <https://rmpsolutions.com/directional-drilling/>
- [40] J.D.Hair&Associates,Inc., Consulting Engineers SITE INVESTIGATION REQUIREMENTS FOR LARGE DIAMETER HDD PROJECTS, available at: https://www.fws.gov/Midwest/endangered/permits/hcp/nisource/2013NOA/pdf/NiSourceHCPfinalAppndxJ_HDD.pdf
- [41] Mamedbekov, Oktay., Directional Drilling. Deflection Tools. Available at: http://drilling-engineering.com/lec/4_Deflection_Tools_final.pdf
- [42] IADC, Evolution of directional drilling since 1900, IADC Drilling Manual, 2015 available at: <https://www.iadc.org/wp-content/uploads/2015/08/preview-dd.pdf>
- [43] Kim, Jongheon, and Hyun Myung., "Development of a Novel Hybrid-Type Rotary Steerable System for Directional Drilling." IEEE Access, vol. 5, 2017, pp. 24678–24687., doi:10.1109/access.2017.2768389.
- [44] Brandt, M., Johnson, K., Elphinston, A. and Ratnayaka, D., n.d. Twort's Water Supply. 7th ed. pp.659-652.
- [45] Hosecon.com. 2020. Hydraulic System Pressure Drop. [online] Available at: <<https://www.hosecon.com/PDF/Engineering/HydraulicPressureDrop.pdf>> [Accessed 16 May 2020].
- [46] Sparkfun.com. 2020. *Teensy 3.6 (Headers) - DEV-14058 - Sparkfun Electronics*. [online] Available at: <<https://www.sparkfun.com/products/14058>> [Accessed 16 May 2020].
- [47] *cclmotors.com*, 2020. [Online]. Available: <https://www.cclmotors.com/public/assets/common/images/product/product/product-81/spec.pdf?1589597594>. [Accessed: 16- May- 2020].
- [48] VEX Robotics. 2020. *CIM Motor*. [online] Available at: <<https://www.vexrobotics.com/217-2000.html>> [Accessed 16 May 2020].
- [49] *Cclmotors.com*, 2020. [Online]. Available: <https://www.cclmotors.com/public/assets/common/images/product/product/product-81/spec.pdf?1589597594>. [Accessed: 16- May- 2020].
- [50] Sparkfun.com. 2020. *Sparkfun Thermocouple Breakout - MAX31855K - SEN-13266 - Sparkfun Electronics*. [online] Available at: <<https://www.sparkfun.com/products/13266>> [Accessed 16 May 2020].
- [51] Amazon.com. 2020. [online] Available at: <https://www.amazon.com/uxcell-Torque-Reduction-Eccentric/dp/B07CGJDDNP/ref=sr_1_17?dchild=1&keywords=eccentric+output+shaft+gear+motor&qid=1589668655&s=hi&sr=1-17> [Accessed 16 May 2020].

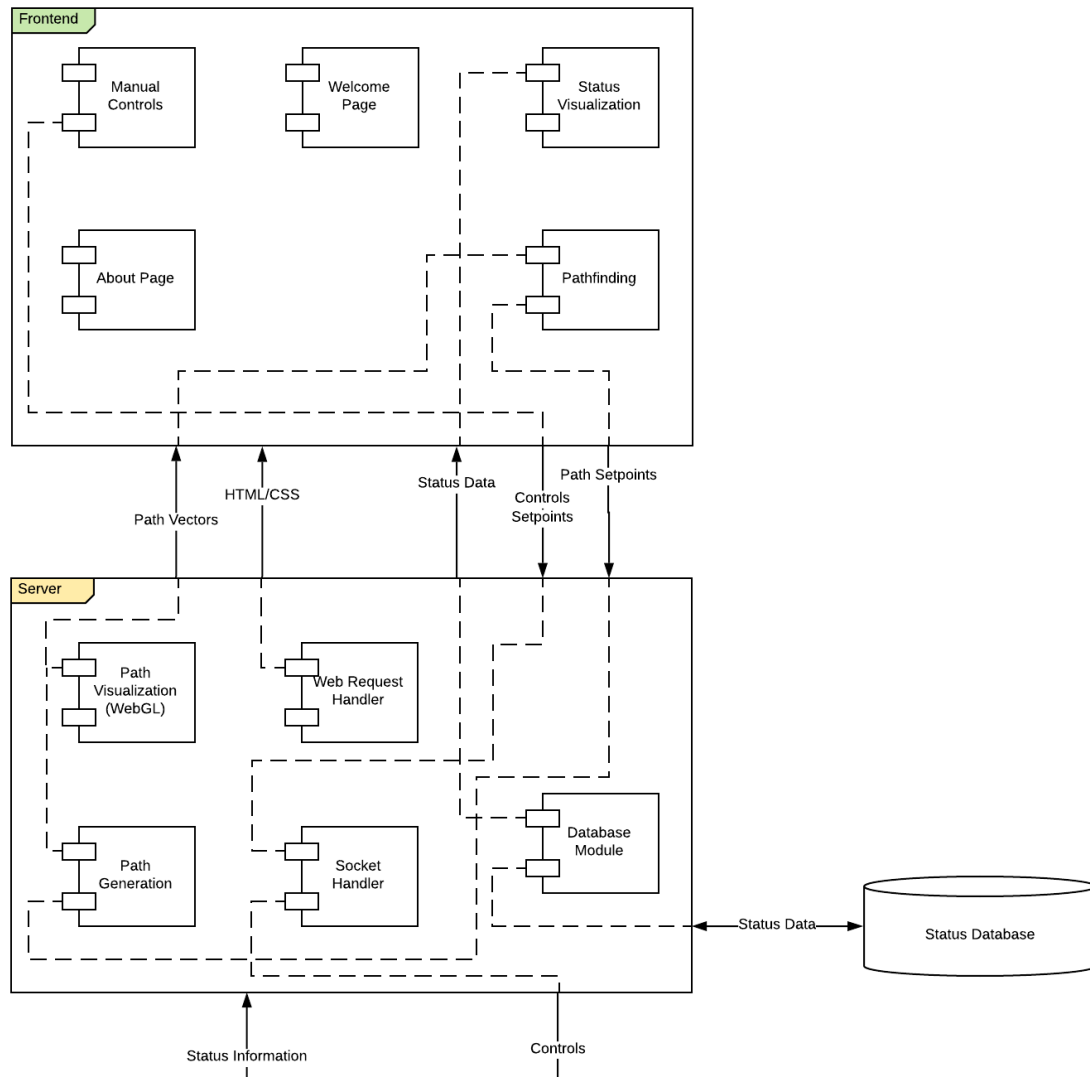
- [52] *Amazon.com*, 2020. [Online]. Available: https://www.amazon.com/DROK-Controller-Regulator-Industrial-Optocoupler/dp/B06XGD5SCB/ref=sr_1_1?dchild=1&keywords=drok+1298&qid=1589598198&s=hi&sr=1-1. [Accessed: 16- May- 2020].
- [53] *Amazon.com*. 2020. *Uxcell 12V DC 200-250RPM Gear Motor High Torque Electric Micro Speed Reduction Geared Motor Eccentric Output Shaft*. [online] Available at: https://www.amazon.com/uxcell-220RPM-Electric-Reduction-Eccentric/dp/B01KTYONKO/ref=pd_di_sccai_1/146-6143624-1298927?_encoding=UTF8&pd_rd_i=B01KTYONKO&pd_rd_r=f33b0d03-3cd4-4c4f-9bef-c433c8504257&pd_rd_w=t4tNg&pd_rd_wg=wICVW&pf_rd_p=e532f109-986a-4c2d-85fc-16555146f6b4&pf_rd_r=6QD2DSVF7Z3JJ0N5MDDV&psc=1&refRID=6QD2DSVF7Z3JJ0N5MDDV [Accessed 16 May 2020].
- [54] Mouser Electronics. 2020. *534B1103JL Vishay / Spectrol | Mouser*. [online] Available at: <https://www.mouser.com/ProductDetail/Vishay-Spectrol/534B1103JL?qs=sGAEpiMZZMtC25lF4XBUwJCNzrDHISWvHH0ha%252BsgXDVjcKLfUzSmA%3D%3D> [Accessed 16 May 2020].
- [55] *Amazon.com*. 2020. *DIMINUS DC 6V Mini Air Pump Motor, DIY And Replacement Accessories, Best For Aquarium Tank Oxygen Circulate (4 Pack)*. [online] Available at: https://www.amazon.com/DIMINUS-Replacement-Accessories-Aquarium-Circulate/dp/B06Y2CXZ67/ref=pd_sbs_60_2/146-6143624-1298927?_encoding=UTF8&pd_rd_i=B06Y2CXZ67&pd_rd_r=cd16babd-bca8-443e-b8d7-c16453e04753&pd_rd_w=XYP6Z&pd_rd_wg=cD1JB&pf_rd_p=12b8d3e2-e203-4b23-a8bc-68a7d2806477&pf_rd_r=JWCRT5YJW2YGSJCKEZF0&psc=1&refRID=JWCRT5YJW2YGSJCKEZF0 [Accessed 16 May 2020].
- [56] Mouser Electronics. 2020. *HSCDANN030PA2A3 Honeywell | Mouser*. [online] Available at: <https://www.mouser.com/ProductDetail/Honeywell/HSCDANN030PA2A3?qs=pcUO8jIlt0a2xNetHRJOJw%3D%3D> [Accessed 16 May 2020].
- [57] Sparkfun.com. 2020. *Sparkfun 9Dof Sensor Stick - SEN-13944 - Sparkfun Electronics*. [online] Available at: <https://www.sparkfun.com/products/13944> [Accessed 16 May 2020].
- [58] Lewis, P., 2020. *Sparkfun/Sparkfun_LSM9DS1_Arduino_Library*. [online] GitHub. Available at: https://github.com/sparkfun/SparkFun_LSM9DS1_Arduino_Library [Accessed 16 May 2020].
- [59] Holeproducts.com. 2020. Bent Sub - HDD50. [online] Available at: <https://www.holeproducts.com/Bent-Sub-HDD50> [Accessed 16 May 2020].
- [60] Bloch, H., 2009. Ingress Protection code explained. *World Pumps*, 2009(11), p.26.
- [61] Hedmond, S., 2020. [VIDEO] Horizontal Directional Drilling Goes Off Course And Into Innocent SUV — Construction Junkie. [online] Construction Junkie. Available at: <https://www.constructionjunkie.com/blog/2017/8/10/video-horizontal-directional-drilling-goes-off-course-and-into-innocent-suv> [Accessed 16 May 2020].

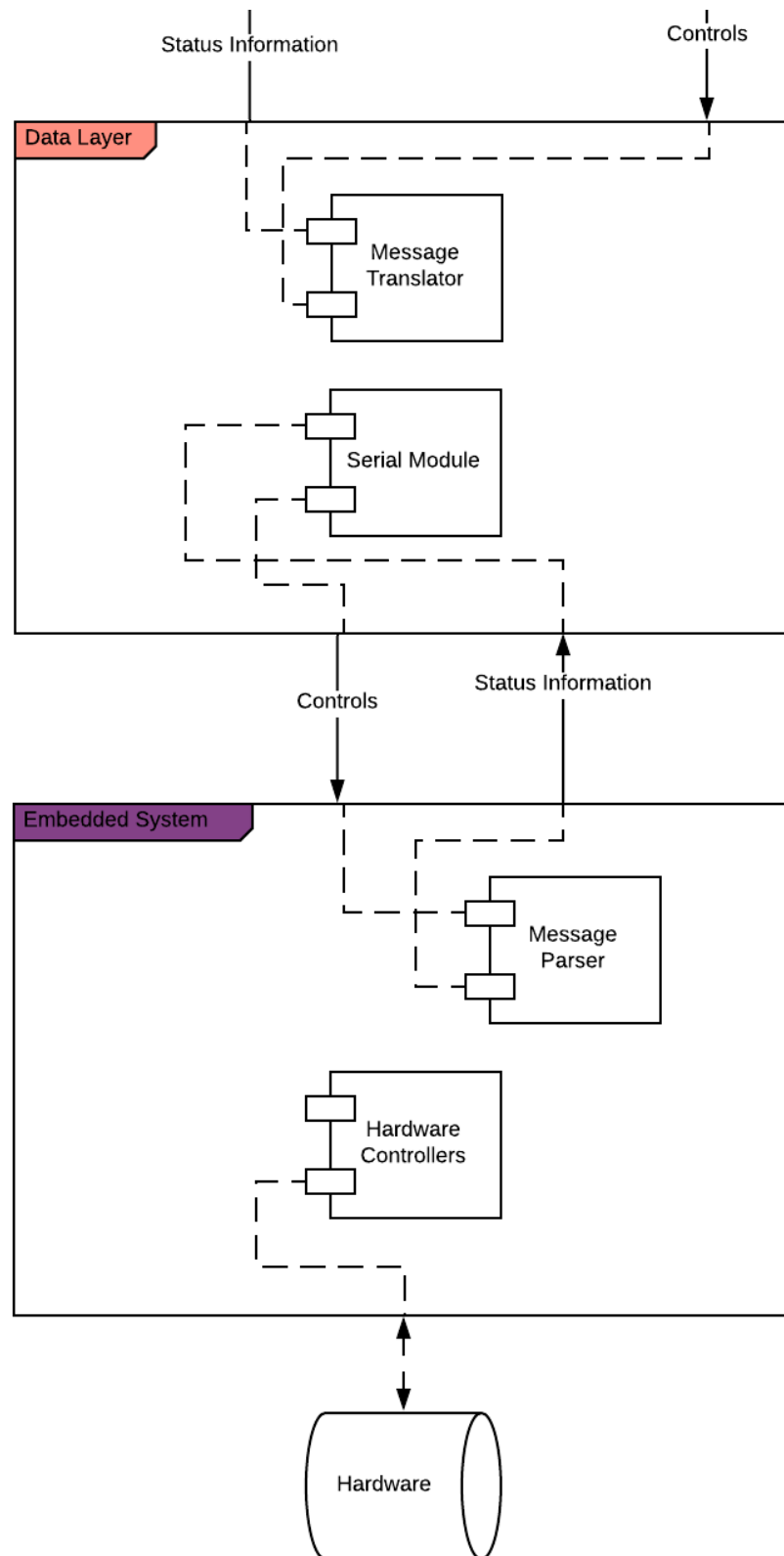
Appendix A – Videos

There are two videos attached below. The first is a three-minute presentation that was given about the project. The second is a video of the prototype operating on a table, demonstrating all the working components. This second video represents the full-stack accomplishments of the team. Both videos are embedded from the web, so if they are not working it is either a problem connecting to the internet or they no longer exist.



Appendix B – Software Architecture





Appendix C – Software Usage

Source Repository

All source code for the MOLE software is available online, hosted in a public GitHub repository. This repository is located at: <https://github.com/TweedChristian/MOLE>, and can be cloned or forked.

Dependencies

The MOLE software relies on several 3rd party software libraries.

Frontend/JavaScript

The following Node packages are required to run the MOLE software. They can be installed by running ``npm install`` in the project directory.

- Node.js
- LowDB
- JSONtoCSV

In addition, there are two other outside imports that don't require installation, but do either require modern browsers or a script tag in the HTML file.

- WebGL
- D3js
- Python

The following python packages are required to run the MOLE software. It can be installed by running ``pip install pyserial``

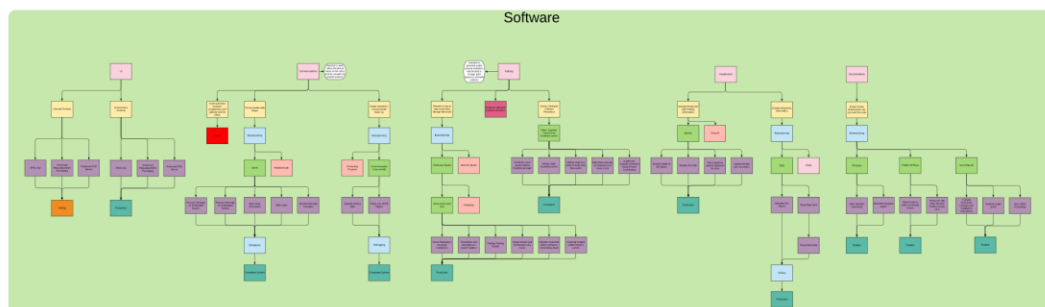
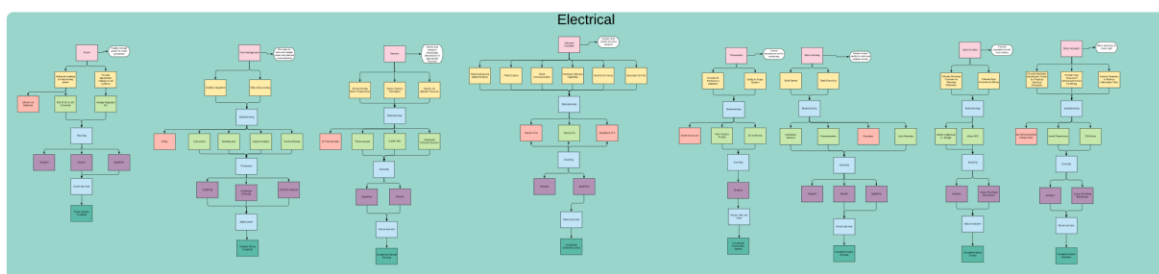
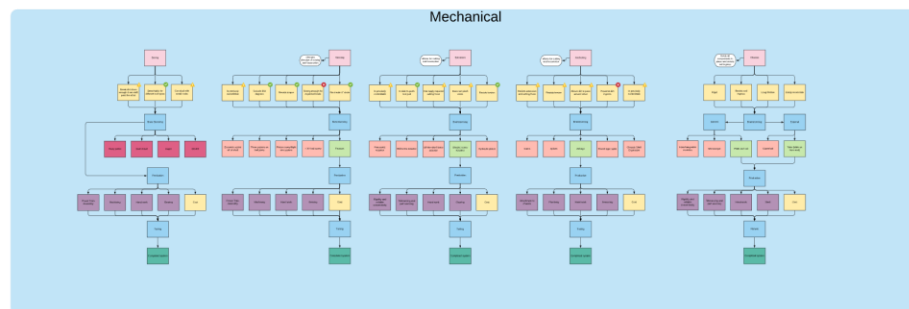
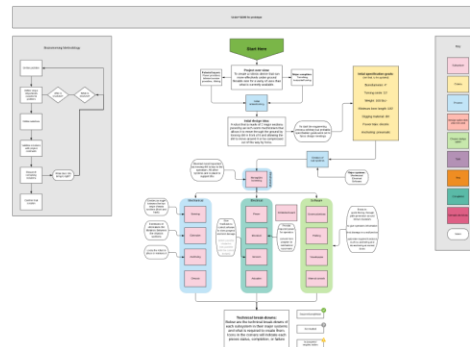
- Pyserial

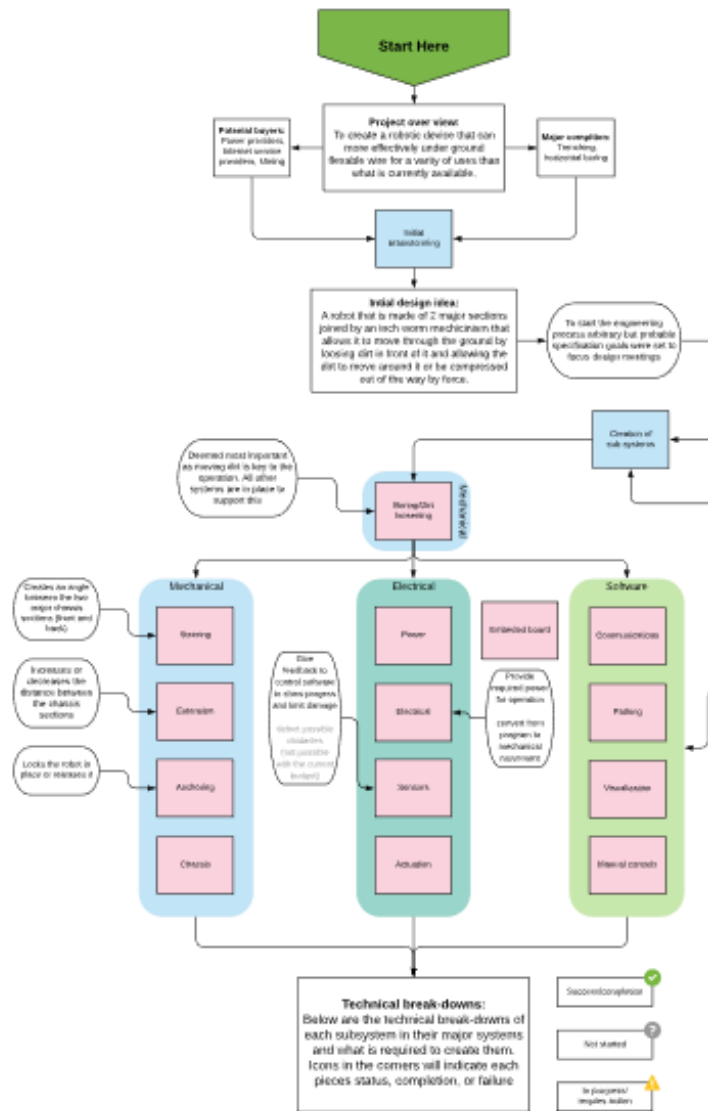
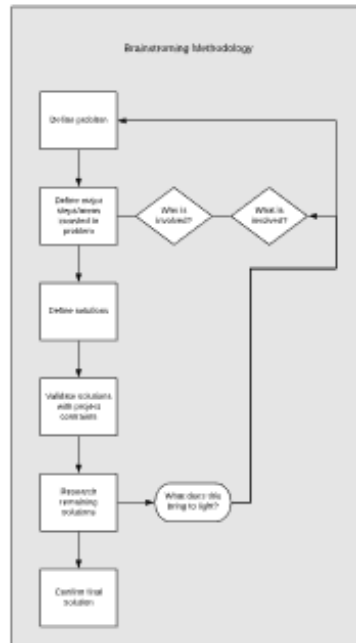
Starting the Software

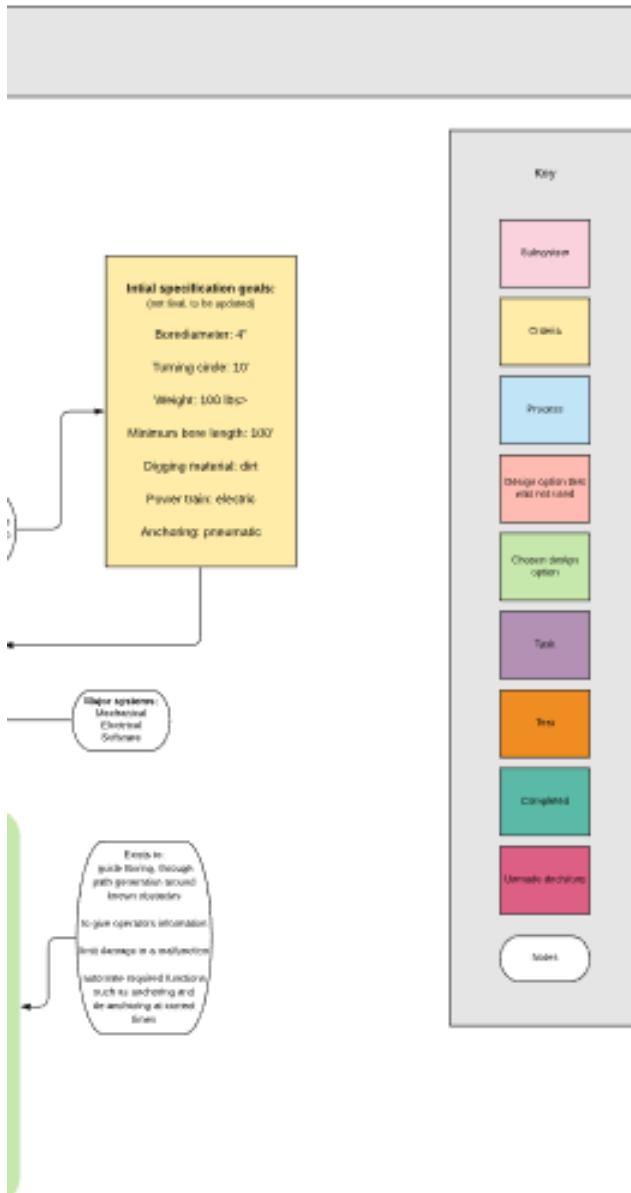
Two separate modules of the MOLE software must be started in sequence. First, run the Node server with the command ``node ./app/server/server.js`` from the project root directory. Next, run the python data layer with the command ``python ./python/client.py <serialPort>`` where `<serialPort>` is replaced by the name of the serial port the Teensy 3.6 is connected to, or `'test'` to run the software in testing mode. The user interface can be accessed by navigating to `localhost:3000` in any web browser. A shell script is included in the software repo to automatically start the system on Unix-based systems, run ``./start.sh <serialPort>`` in the project root directory.

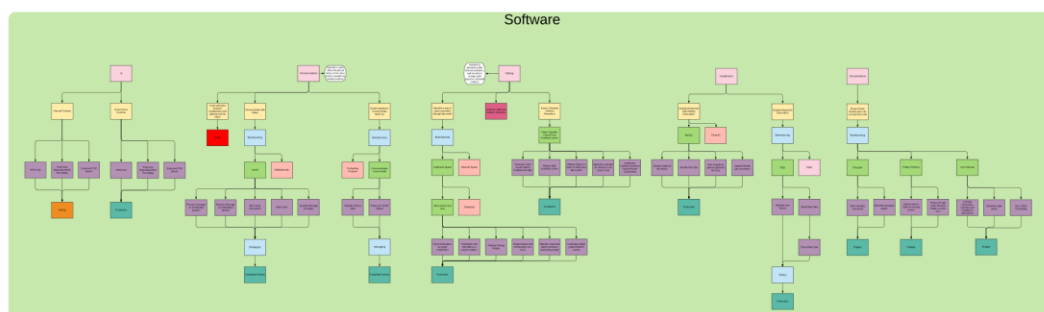
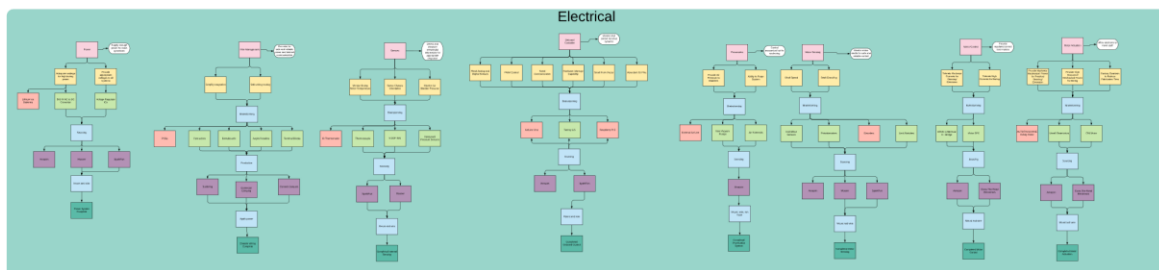
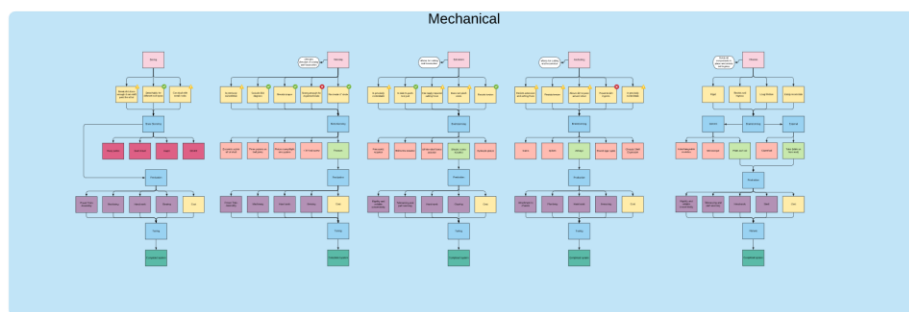
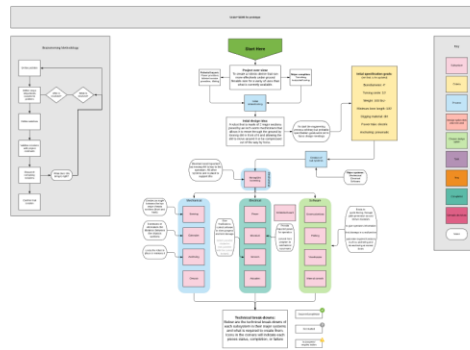
Appendix D – Methodology Chart

Below is the entirety of the methodology chart. Because it is too large to be conveniently integrated in any format, it has been replicated on several different pages. This page can be used as reference.





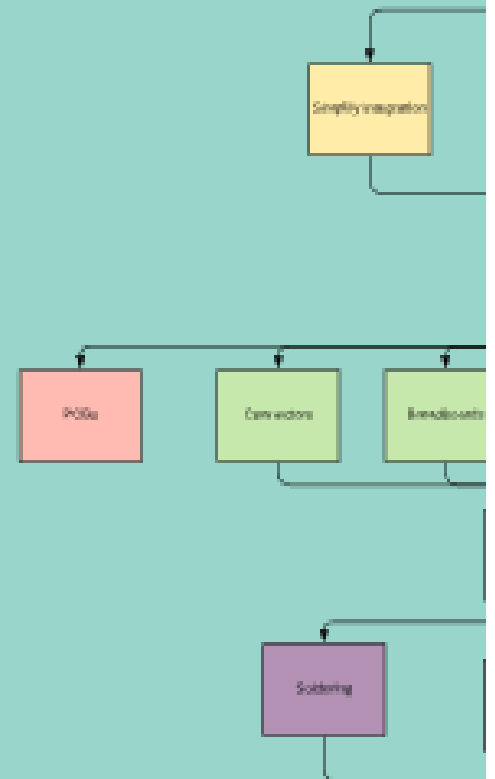
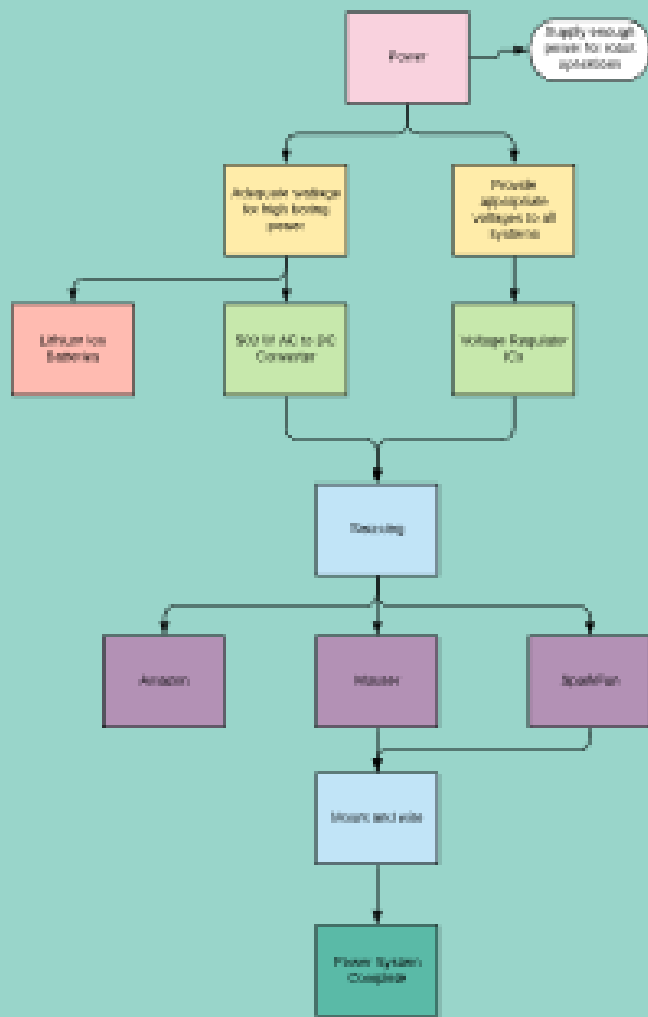


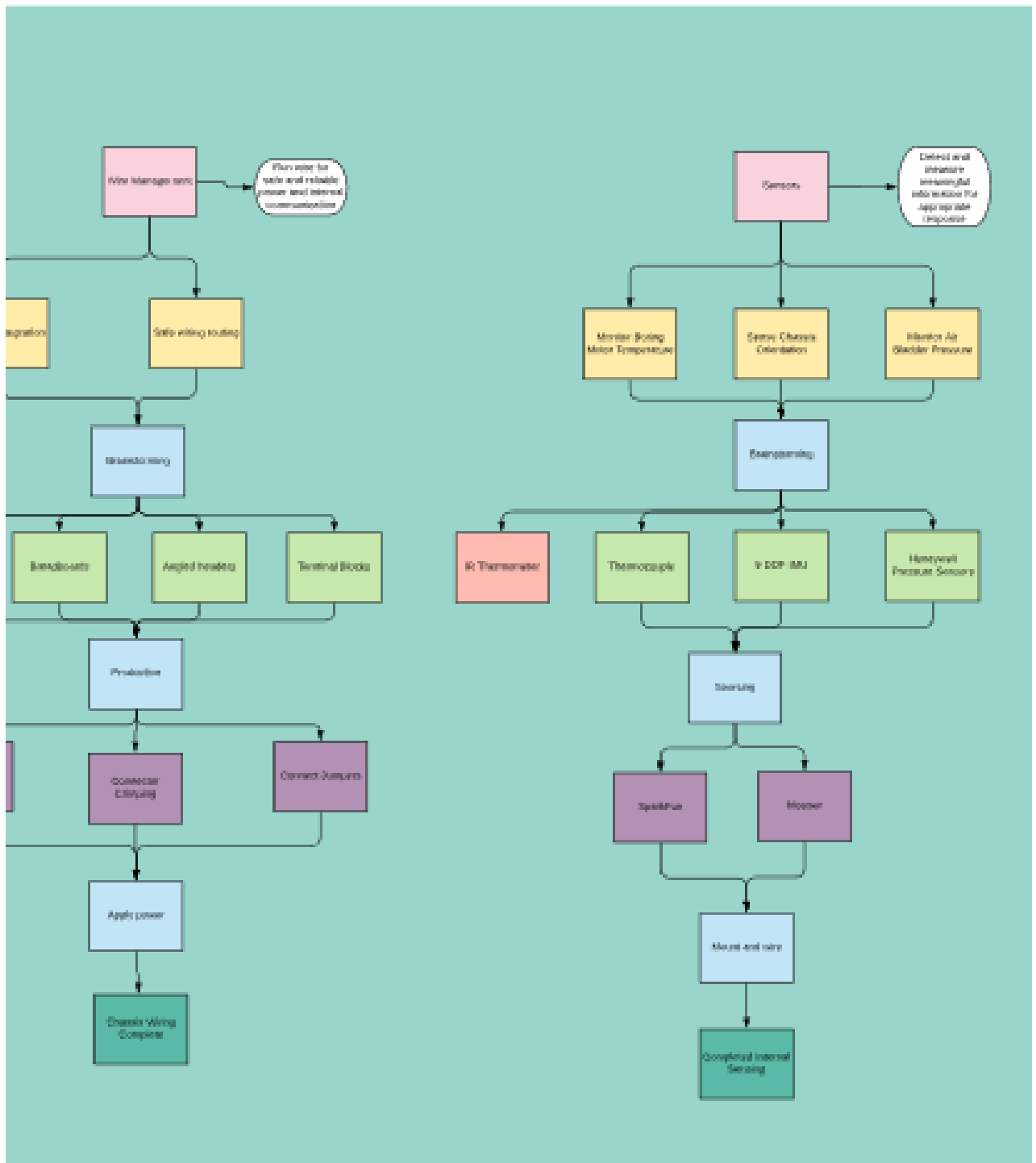


Mechanical

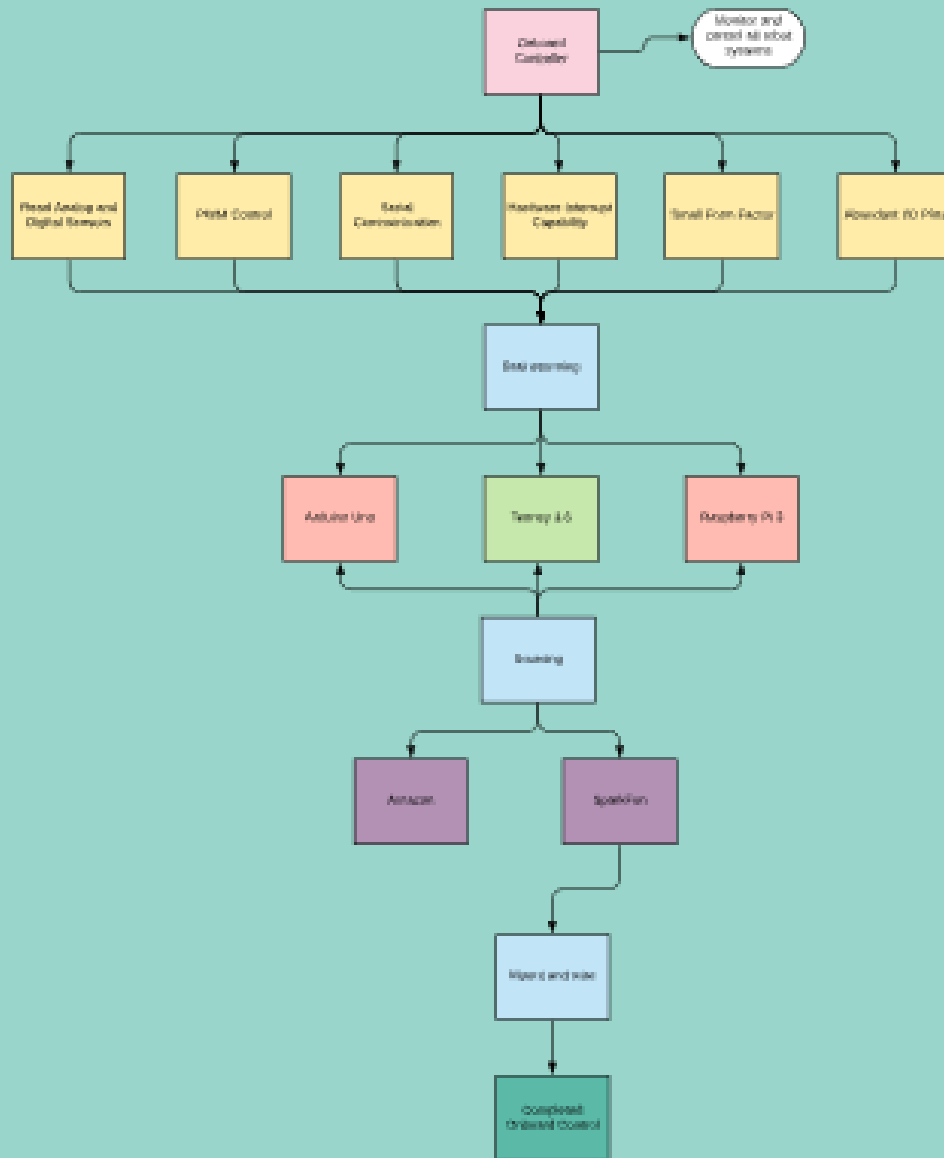


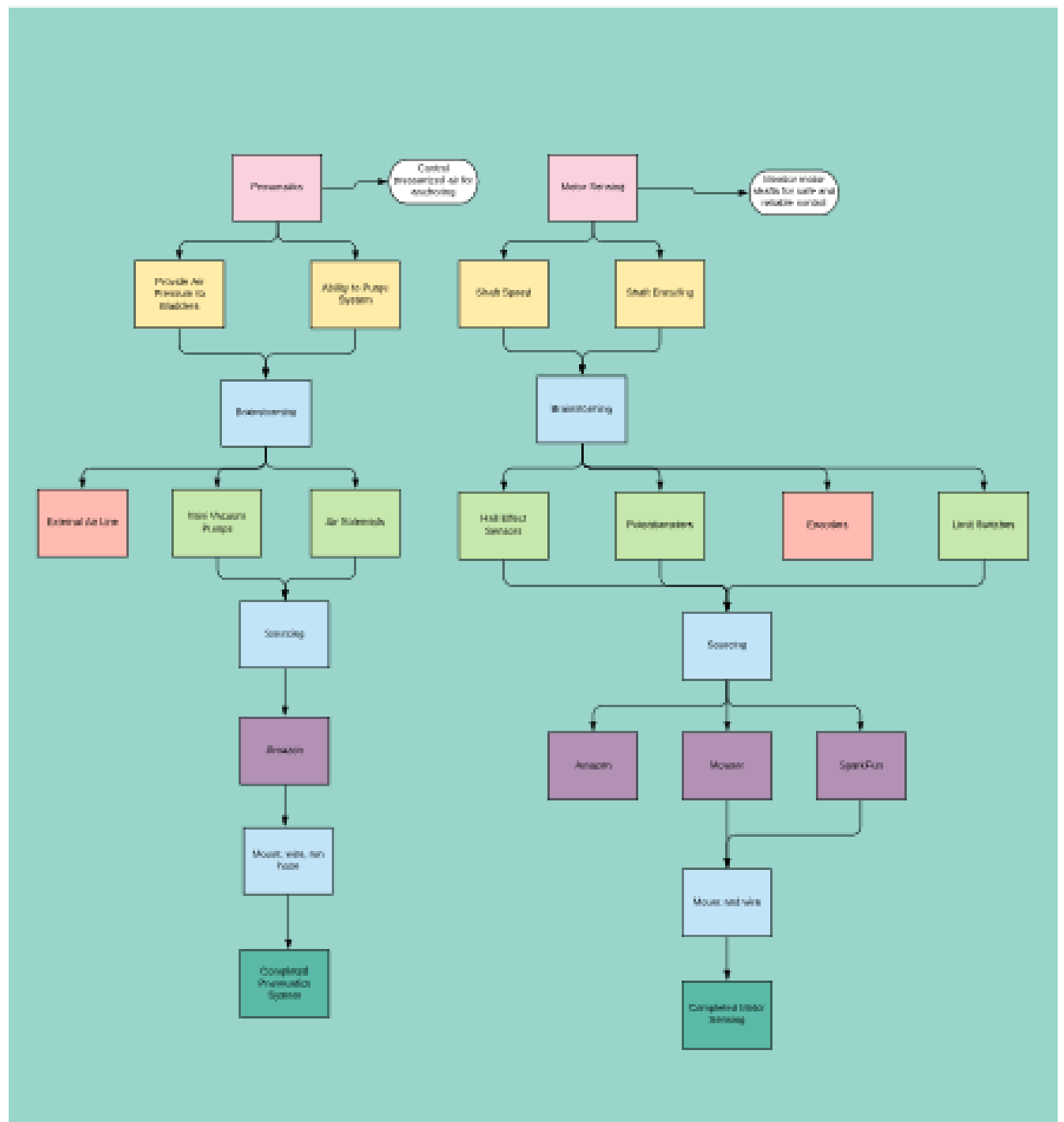


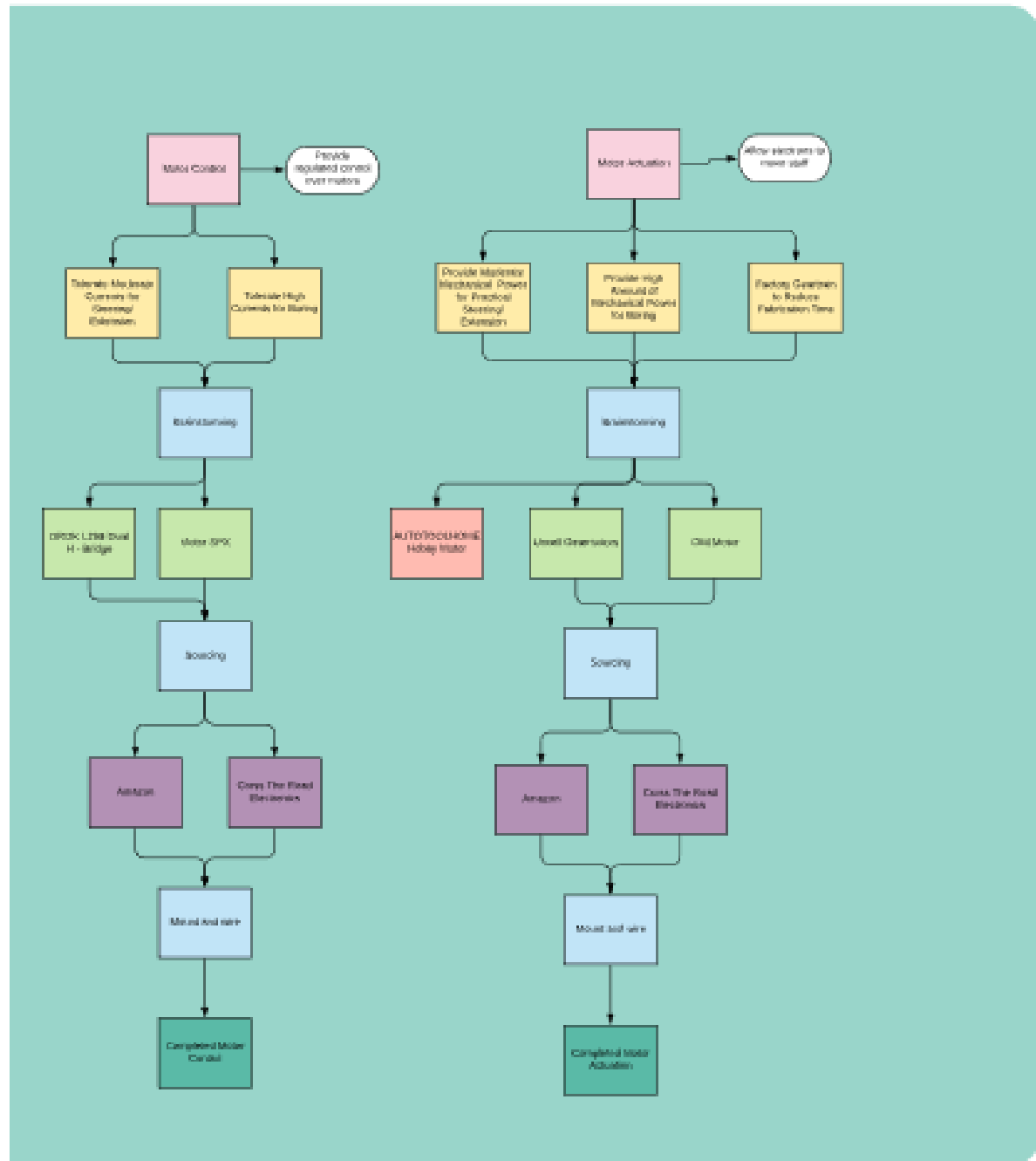


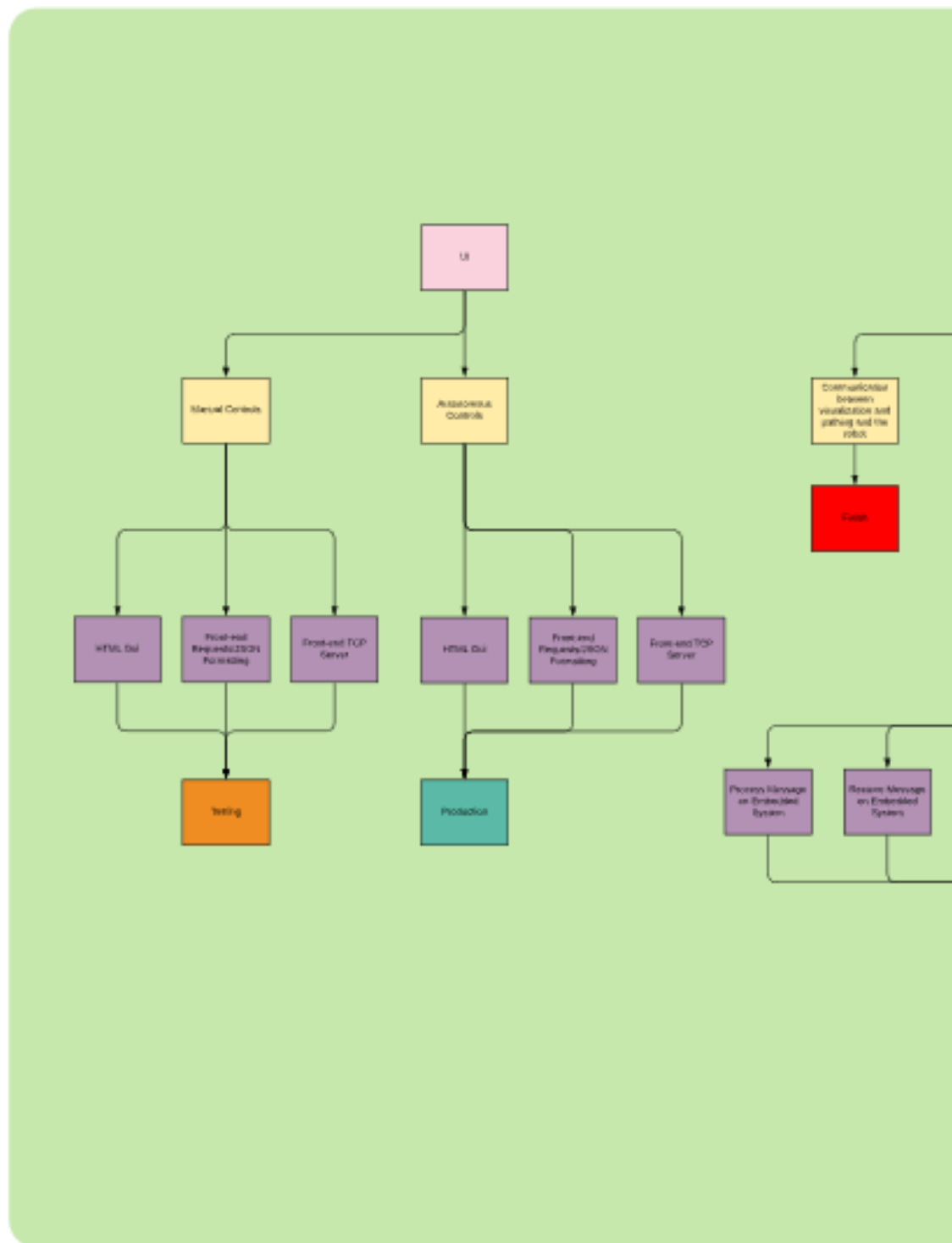


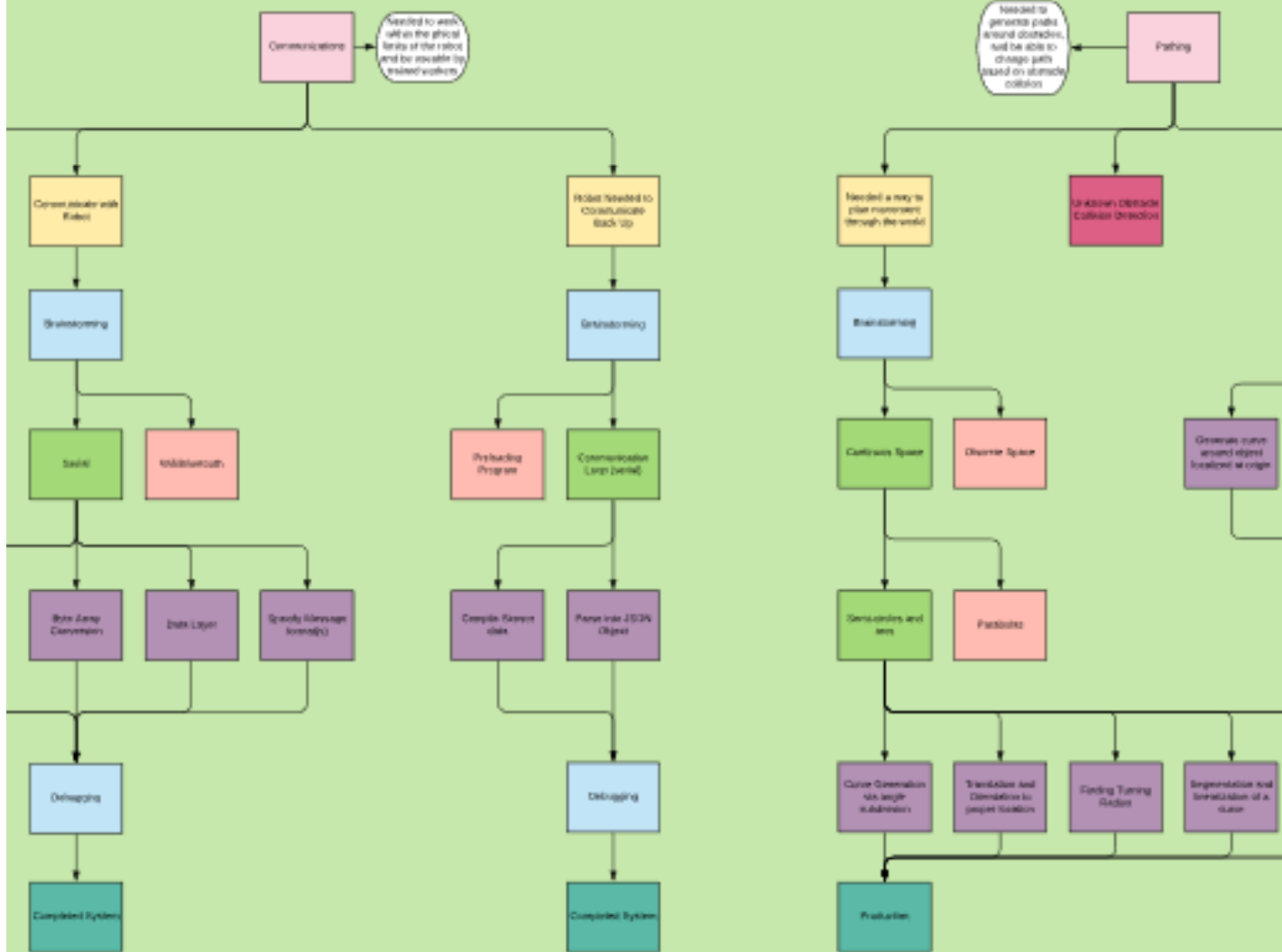
Electrical



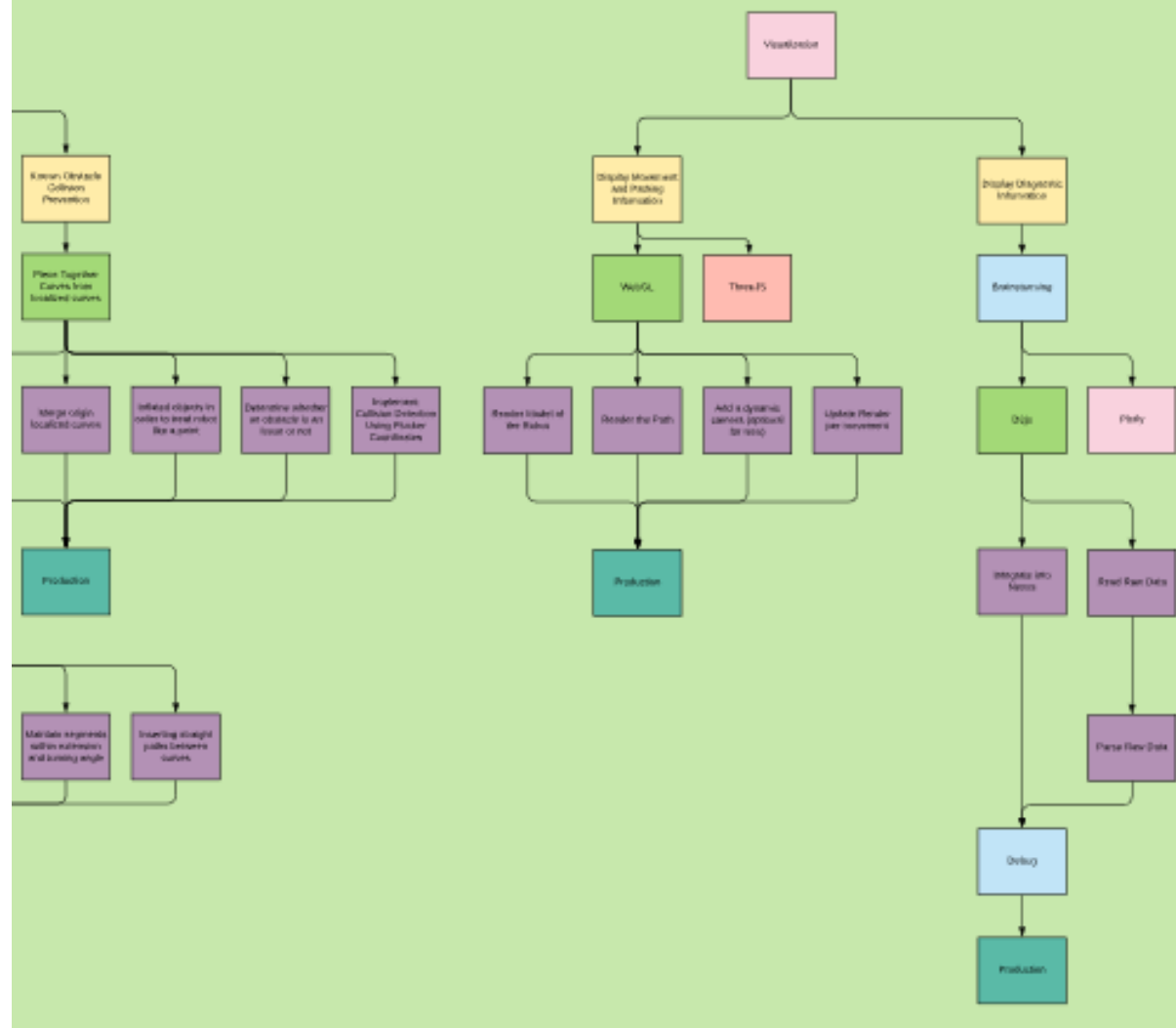


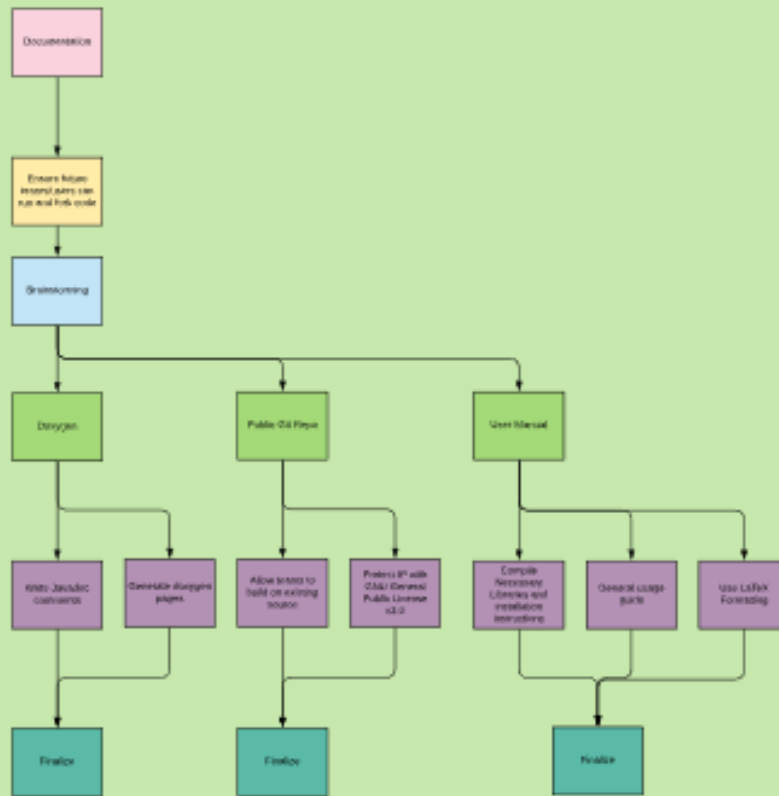






are





Appendix C – How to Use Software

Appendix D – Videos of Operations