Autonomous Subway Inspection Vehicle

Omar Aziz, Noah Bolzoni, Brandon Castro, Anthony Lucero, Sean Maggio, Marcin Wisniowski

Phase 6 Documentation

Final Report

05/12/2020

Table of Contents

Project Overview

Project Rationale

Systems like the MTA see over 2 billion riders a year. This high system usage, in combination with aging infrastructure, introduces a lot of wear and tear on the subway tracks. Constant wear and tear not only results in high maintenance costs, but also creates a dangerous environment for operating trains if not timely detected or repaired. To add, the New York City subway is the busiest metro service in the United States and has strong needs to keep services running for as long as possible without delays. The main focus for the MTA is to ensure that the track gauge - the width between the inside faces of train tracks - is within tolerable limits. A call to an operator for the MTA confirmed this fact by one of our members. As a result, this metric, along with various positional and alignment measurements is the main focus of the ASIV.

The ASIV stands for Autonomous Subway Inspection Vehicle and is expected to run on active tracks 24/7 to collect basic track geometry data and indicate incorrect geometries to the operators of the metro. With it's in house developed retraction system, the vehicle is able to stay operational on the tracks while active trains are running. Current inspection practices range from multi-million dollar geometry cars which are only able to run across all the tracks operated by the MTA six times a year or handheld tools for human monitoring by visual inspections. There are currently two major benefits to the ASIV over current practices. The ASIV is able to continuously patrol rail lines and collect data autonomously and furthermore, the ASIV is much cheaper than current full-scale track inspection vehicles that are in current use with expected costs of one vehicle to be $15,000 after profits.

The Autonomous Subway Inspection Vehicle looks to increase the productivity of subway inspections by uncovering and flagging warnings for track defects early, in order to concentrate a subway inspector's efforts into distinct geographic areas of focus. While the current regulations may restrict the creation of a fully autonomous system, the project, regardless, adds value by creating an autonomous and live asset management of the subway track which can hone in for further human inspection on specific sectors of the rail network. When regulations change in the future, the system can be used to provide stronger analysis for proactive operational investments and scheduling of site-technicians and inspectors for daily maintenance.

Phase Recaps

Phase 1

Recognized Needs

The proposed design for the track inspection vehicle aims to make mass transit by train much more efficient and safe. Current subway transit is constantly plagued by delays and interruptions in service due to issues caused by the track. The four large track inspection vehicles that the MTA presently uses, are only able to inspect the full length of the track 6 times a year. This means that there are two months in between inspections. This gap in between inspections allows for potential deformation and breakage to go unchecked for up to two months. During this time the defects could cause derailments and therefore inconvenience the public. The proposed design aims to make more frequent inspections so that more defects can be identified. With more inspections, there will be a reduced number of derailments and inconveniences to the public.

The main intended customer for the track inspection vehicle would be the MTA or Path as they are directly responsible for transportation by subway. The MTA/Path requires an inspection vehicle to take reliable measurements, save space, and to save time. As stated before, the MTA only has four track geometry cars that are able to inspect the full length of the subway system a mere six times a year. These cars are full sized subway cars that are fitted with sensors and computers with people inside to inspect the tracks. For the MTA/Path to increase efficiency, a smaller, more cost effective solution is needed. With the risk of derailments and deformations, the MTA needs a quicker and cheaper way to inspect tracks. By having a machine that has accurate sensors, the inspection vehicle will be able to safely determine if a rail is aligned and that all rails are up to code. A smaller vehicle will also save space as well as reduce the costs of use as having an autonomous vehicle will help measure the rails quicker and require less personnel for operations.

Concept Generation

The most prevalent issues for an autonomous vehicle was the frequent passing of trains. This places constraints on the conceptualization process. The vehicle had to be very low profile. It also needed a way to reliably and safely evacuate the track when a train was detected coming towards it. To perform this function, the wheels must be able to quickly retract to a narrower profile as a train is approaching. It must also have some sort of mechanism that supports the vehicle on the ground as the wheels are retracted from the track. From here, several concepts for these mechanisms were created.

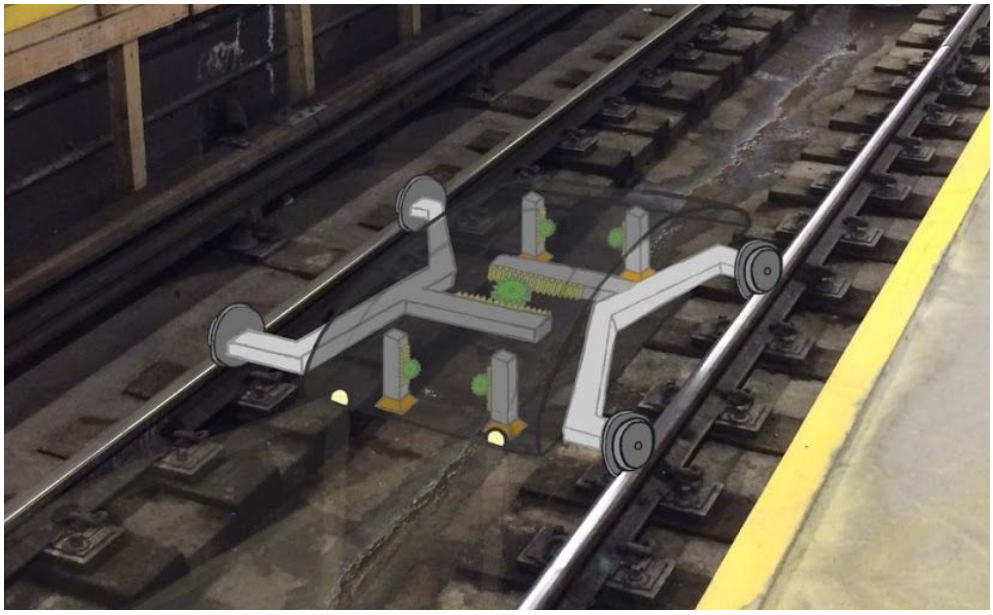


Figure 1: Initial Sketch

The chosen concept followed the basic design of the initial sketch shown in the figure above. However, it used two linear actuators directly connected to the wheel bases in the same line of motion. This was the simplest concept and provided the largest range of motion of the wheel base, at almost 12 inches of retraction total. An isometric view showing the same concept on rails is shown below.

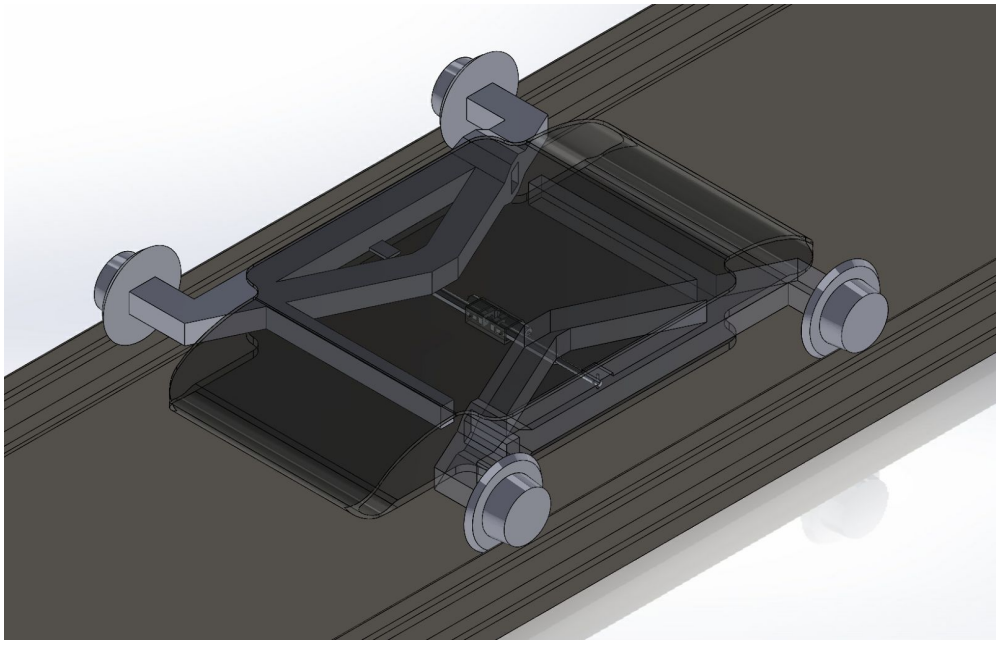


Figure 2: Isometric View of Initial Concept

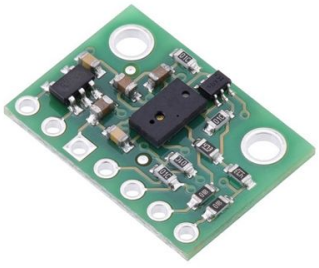
Phase 2

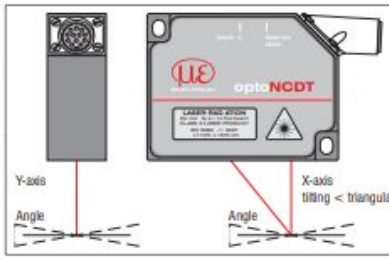
Narrowed Scope

- Focus on measuring track gauge
  - MTA regards as most important
  - FRA: 15-20% of track issues stem from wide gauge
  - Use distance sensors to measure
- Inertial data may also need to be collected
  - Accelerometer and gyroscope

Sensor Choice

- IR Time-of-Flight (TOF) Distance Sensor
  - Inexpensive
  - Sensitive to ambient light
  - IR beam widens over longer ranges
  - Made for lower ranges
    - Accuracy up to 1mm at 10cm range
  - Up to 400kHz communication
  - Easy to interface with microcontrollers, I²C
  - Cheap, accurate alternative to lasers
  - Viable for Alpha Prototype
- Laser Distance Sensors
  - Single point, very accurate
  - Micro-Epsilon sensors
    - Up to 30 deg. tilt with <1% error
    - IP67 Rating





Angle	X-axis %	Y-axis %
±5 °	typ. 0.12	typ. 0.12
±15 °	typ. 0.2	typ. 0.2
±30 °	typ. 0.5	typ. 0.5

Measurement errors through tilting with diffuse reflection

Figure 3: Time of Flight Sensor(left) *MicroEpsilon Sensor*(right)

Alpha Prototype

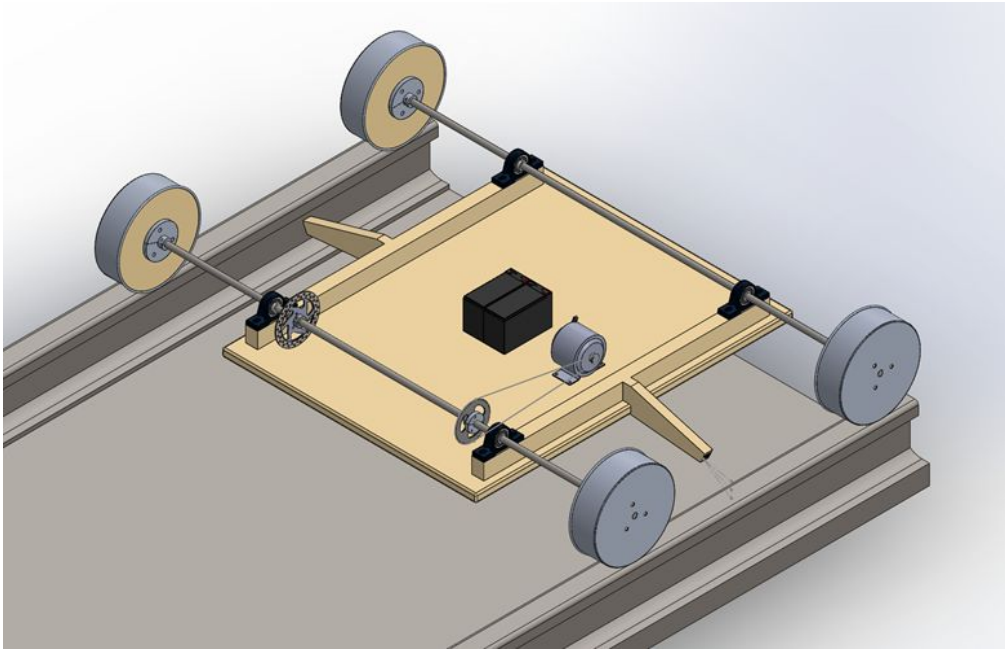


Figure 4: Initial Alpha Prototype

A simple, proof-of-concept was created that would demonstrate the measurement capability of the desired beta design. This would be built as a simple, fixed-axle platform using a repurposed scooter motor and parts that could be reused for the beta prototype. Since at this phase the team did not understand measurement and vehicle dynamics yet, this alpha prototype would be a cheap way of gathering data on a critical function of the vehicle without having to build a large portion of the beta prototype to get it to work. The alpha prototype will continue to be elaborated on in the next phases from this initial design. It serves as an analogue to the beta when it is running on a straight section of track.

Phase 3

Measurement Requirements

Table 1: Rail Gauge compliance values

Class of Track	The gage must be at least -	But not more than -
Excepted track	N/A	4'10¼"
Class 1 track	4' 8"	4' 10"
Class 2 and 3 track	4' 8"	4' 9¾"
Class 4 and 5 track	4' 8"	4' 9½"

Gauge was defined as a measure between the heads of the rails at right angles to the rails in a plane five-eighths of an inch below the top of the rail head. With this definition, sensors could be correctly positioned to measure at the relevant locations of the rail and valid measurements can be submitted into the system. The range of allowable gauge measurements are shown in Table 1 based on track class.

Sensors

Micro Epsilon shipped a donation of two optoNCDT 1402 laser sensors that will be used both in the alpha prototype and the final design. Using laser triangulation, the laser beam can be effectively aligned to give very accurate readings of the track gauge and give back analog responses that can be later parsed into distance measurements. A setting of the sensors allows for accurate output scaling to the desired ranges that can be honed in for the necessary range of values. Accurate alignment and synchronization of the sensors is critical in conveying valuable readings with accurate locations on the rail. The resolution of these sensors is 13-100 micrometers. The maximum deviation from a linear distance output is less than or equal to 0.36mm, which again demonstrates adequate precision. These will be used in the beta prototype.

An inertial measurement unit (IMU) was also purchased in order to allow for more data to be captured about the current rotation, acceleration, and magnetic measurements from an absolute orientation. In order to keep an accurate reading for gauge the ASIV needs to keep a relative inclination of our sensors to the ground plane and by combining the accelerometer and gyroscope readings found from the IMU that would be possible.

Alpha Prototype

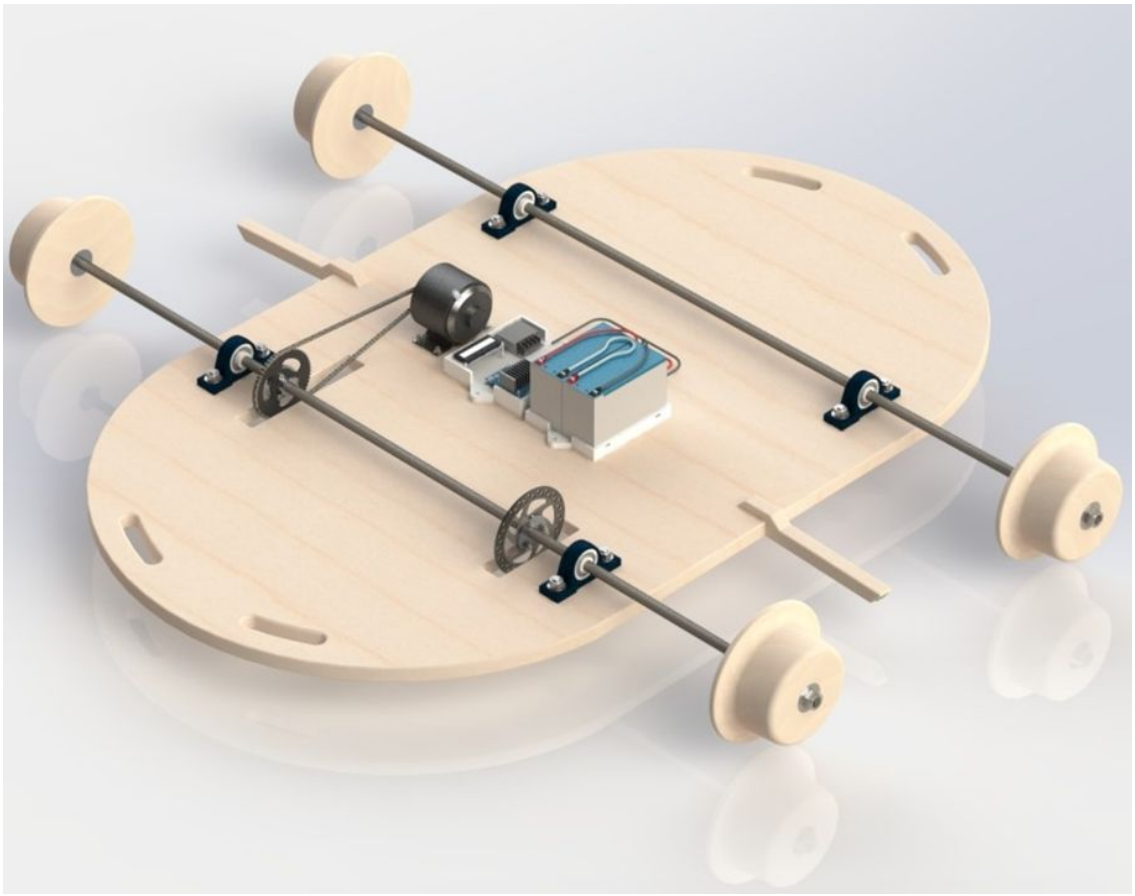


Figure 5: Phase 3 Alpha Prototype Rendering

Shown above is the updated alpha prototype. The prototype is built around a heavy and narrow wooden base to emulate the weight and dimensions of the proposed beta vehicle and chassis shape. The prototype consists of two fixed axles, with one being driven using a chain-drive system connected to a single motor. The driven/live axle has two key differences from the dead/beam axle, which are two attached components - a sprocket and disk brake. The sprocket is used to connect to the chain to transmit power from the motor. The disk brake is added in case testing reveals that motor braking is not enough to stop the vehicle in adequate time and distance.

Phase 4

Electrical Design

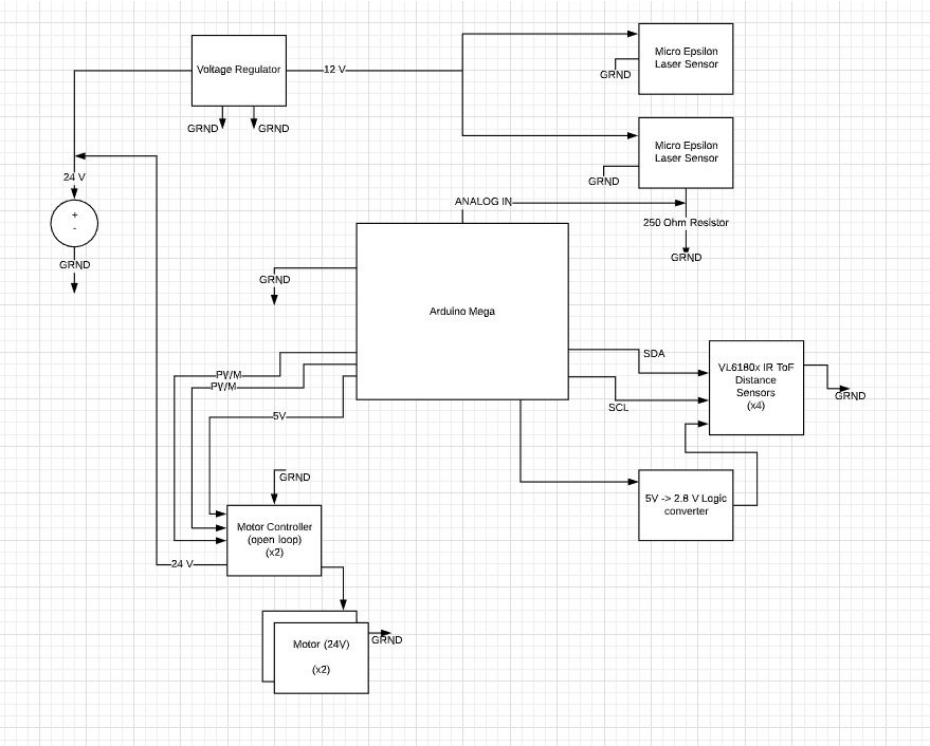


Figure 6: Alpha Prototype Electrical Circuit

The above electrical diagram specifies the necessary connections in order to have the distance sensors and the motors to work as desired. A voltage regulator helps step down the voltage to desired values and PWM control will be implemented on the motors in order to control movement.

Alpha Prototype

Many days over several weeks were spent in the machine shop creating the custom parts for the alpha prototype. Despite the delays, it was necessary to build a testing platform in order to best understand the dynamics of the vehicle. The alpha prototype is designed to represent the final prototype in its “extended” mode, which is just traveling on a straight track. This means that a wheel retraction mechanism was not needed at this stage of the prototyping, but the both vehicles will act the same if tested on a straight track.

Figure 7: Fabrication of Parts

Several different steps, processes, and tools were used in the fabrication of the alpha prototype parts, highlighted below:

- Lathe
- Manually created features
    - Bore holes, turning diameter, cut parts from each other
- Set references to be used in programs
- Wrote programs to create more complicated features multiple times
- Used broaching tool and press to make keyways
- Milling Machine
- Manually used machine to set references
- Wrote a program to recreate the exact wheel part four times

The alpha prototype was constructed and tested in the ABS design space. Long wooden beams were used to simulate a track for the vehicle to run on. Shown in the figure below is the fully constructed alpha prototype connected to a laptop for live readings.

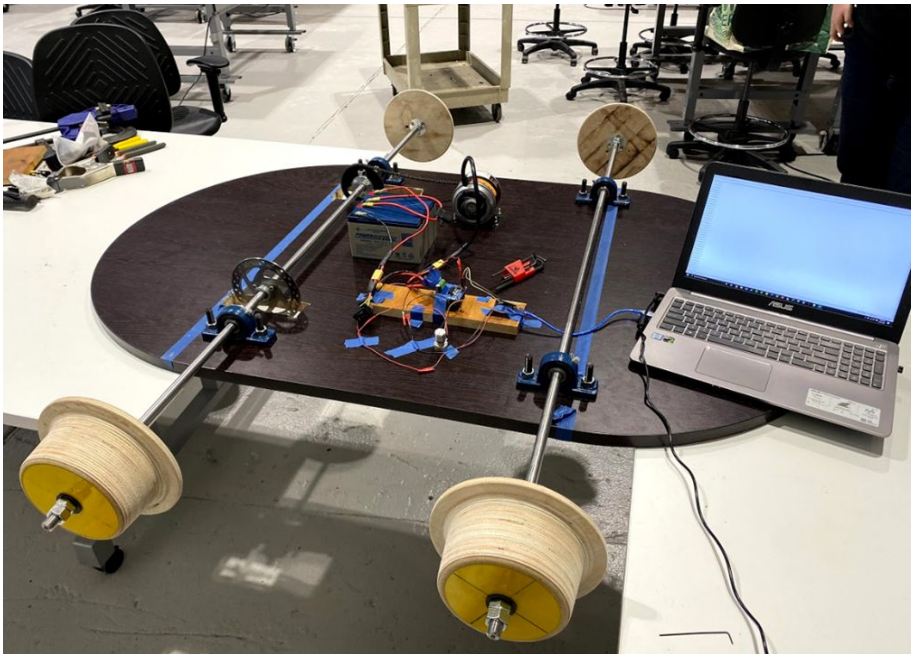


Figure 8: Alpha Prototype

The prototype was placed on the wooden track for short testing runs. Several configurations of inclines were tested to simulate real-world conditions. The vehicle performed nominally under all conditions, with a large margin of power to spare.

Figure 9: Initial Test Run of Alpha Prototype

Beta Prototype

Significant revisions were made to the beta prototype at this phase in preparation for its fabrication during Phase 5. The beta prototype was planned to have two identical aluminum truss structures faced with steel sheet gusset plates, with each structure holding one front and one back wheel and one motor. Two weight bearing sections of aluminum tubing would protrude from the truss structure towards the middle of the vehicle, and these would be placed in slots where they could slide freely while still supporting the weight of the vehicle. One linear actuator per structure would be attached to the middle of the structure, and the actuators would push or pull on this point to extend and retract the wheel base structures.

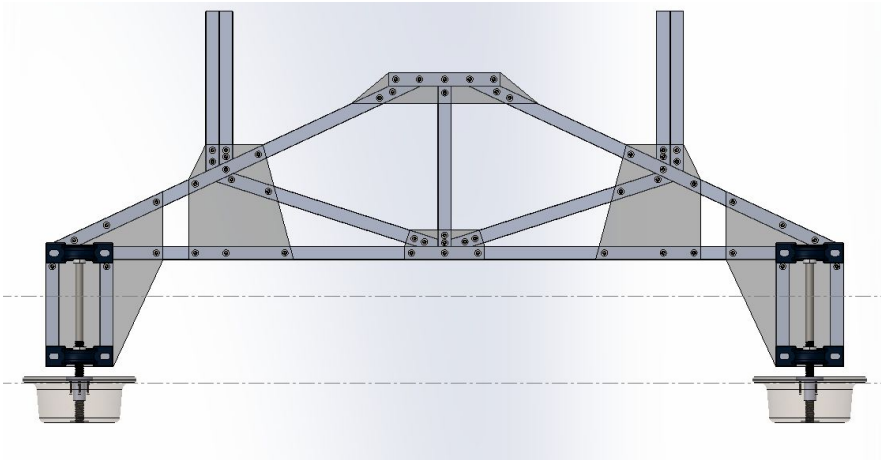


Figure 10: Original Beta Prototype Retraction/Extension CAD Model

There were many issues with this design that led to a recent pivot in the design of the vehicle. First of all, this direct method of retraction and extension limits the range of motion to the stroke of the linear actuators. This means the vehicle would get only six inches of travel on each side. Larger actuators are not only more expensive, but have much slower travel rates, and do not fit in the vehicle if placed back to back. Secondly, the weight bearing beams cannot retract the required range of motion. Since the vehicle is symmetrical, the beams collide into each other after traveling a distance of less than six inches. Shorter beams were considered, however, this still did not provide enough travel, and also now concentrated the weight of the vehicle on a much smaller section of tubing. Additionally, the amount of aluminum and hardware needed such as rollers for the slots would be very costly relative to the remaining budget.

With these factors considered, a new design was created that solves many of the problems. The front and back halves of the wheel base were split and the truss design was largely simplified. The wheel retraction mechanism now functions by rotating four wheel bases about a large pin in a brass sleeve bearing. Shown in the figure below is the new wheel base design in its extended and retracted position.

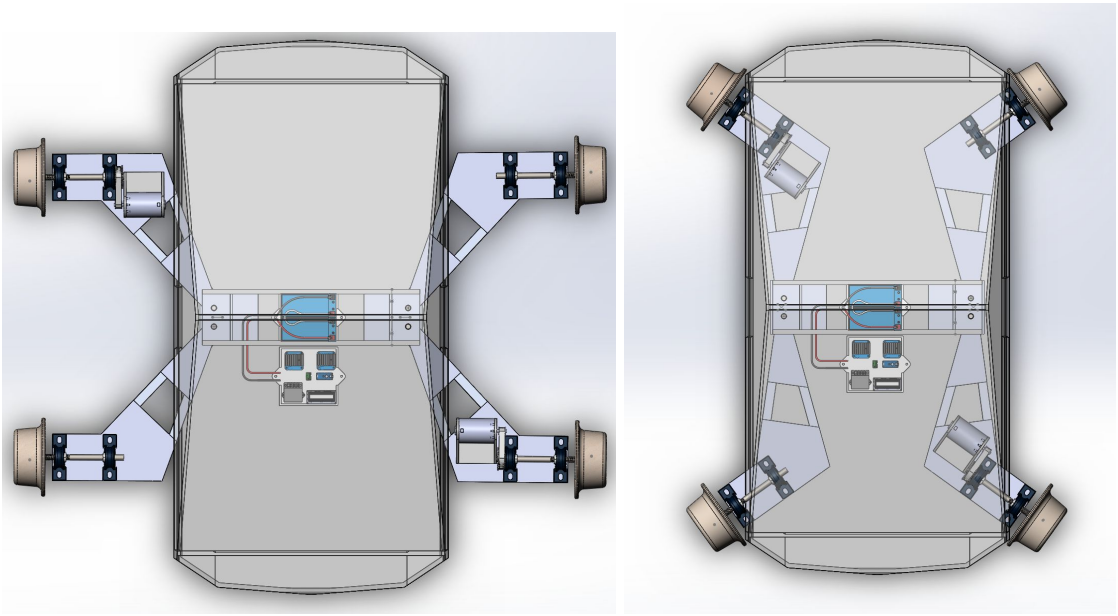


Figure 11: Revised Beta Prototype CAD Design

Beta Functionality

The final prototype was expected to have the following features:

1. Master on/off
2. Speed and direction control via potentiometer knob
3. Data collection on/off switch
4. Emergency off button
5. Gauge and IMU angle collections and write to SD
6. Retraction mechanism triggered upon button press

Phase 5

Pivoting Deliverables

Soon after Phase 4 was complete, the campus, as with the rest of the country, was shut down and all progress on the physical prototype was unfortunately halted due to the threat of COVID-19. Senior Design teams were forced to shift their projects to alternatives like modelling and simulation, which severely limited the progress that could be made. However, the heavy use of CAD in this project allowed the team to have a fully-fledged model that could be simulated in some ways. Electrical baseline design could also progress.

Beta Prototype

Motion analysis was used to simulate all parts of the retraction mechanism with realistic specifications and conditions. The retraction mechanism uses many linear actuators to accomplish four main movements. The first movement is to rotate the four legs from their retracted position around the hinge to their extended, vertical position where they will support the weight of the vehicle. The second movement is to extend the telescoping portion of the legs until each leg contacts the ground, and then slightly lift the vehicle off of the tracks. The third movement is to retract the wheels inward using their respective actuators. Finally, the fourth movement involves retracting the telescoping portion of the leg to lower the vehicle down closer to the ground. The animation can be viewed via the following YouTube link: https://youtu.be/NIcN_vaSdo8 . A before and after snapshot of the sequence is shown below.

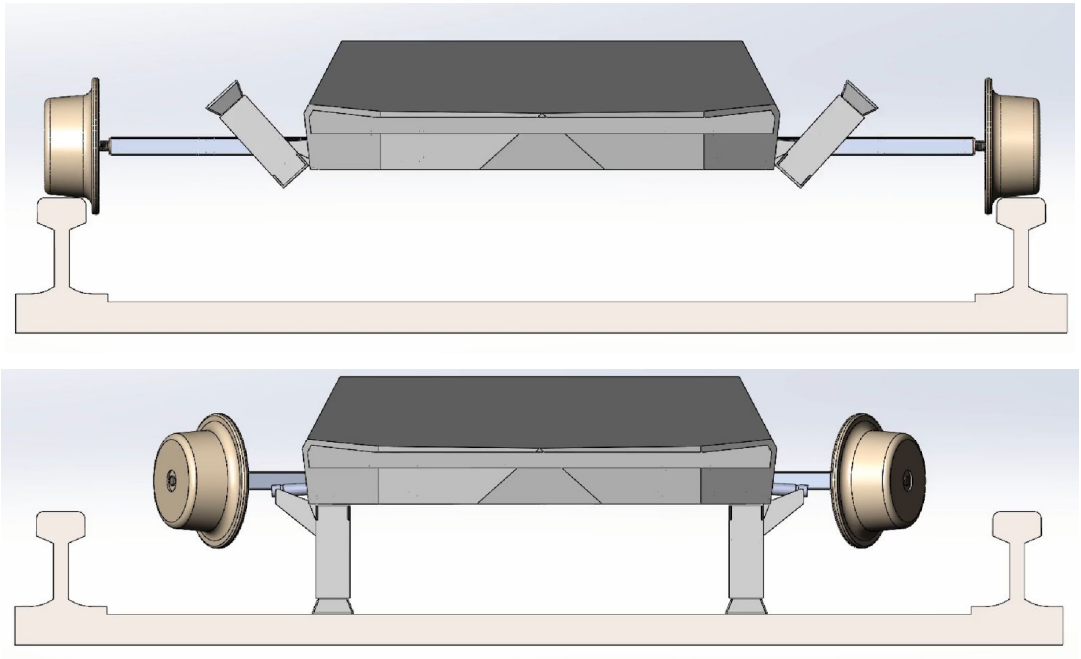


Figure 12 - Before (top) retraction and after (bottom) retraction

Validating Functionality

Doubts have been cast on the functionality and effectiveness of the design. Concerns about the stability of the vehicle, especially through curves, were constantly raised and directed at the team, noting that the sensors would not always be measuring the side of the track as the vehicle shifts and tilts. Through the new objective of fully modeling the vehicle, these concerns and their potential consequences were found to be insignificant. The wheel flange and its distance from the track greatly limit the amount it can tilt, shift, or twist on the track. This ensures that the sensors reading the measurements do not measure very different portions of the track. Shown below are two worst-case scenarios where the vehicle is shifted to its physical limits in two different ways.

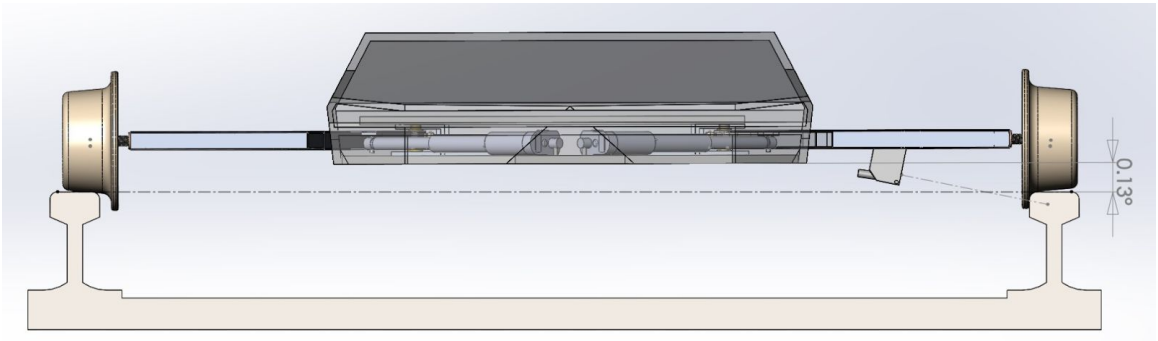


Figure 13 - Maximum tilt from horizontal plane (~0.13 degrees)

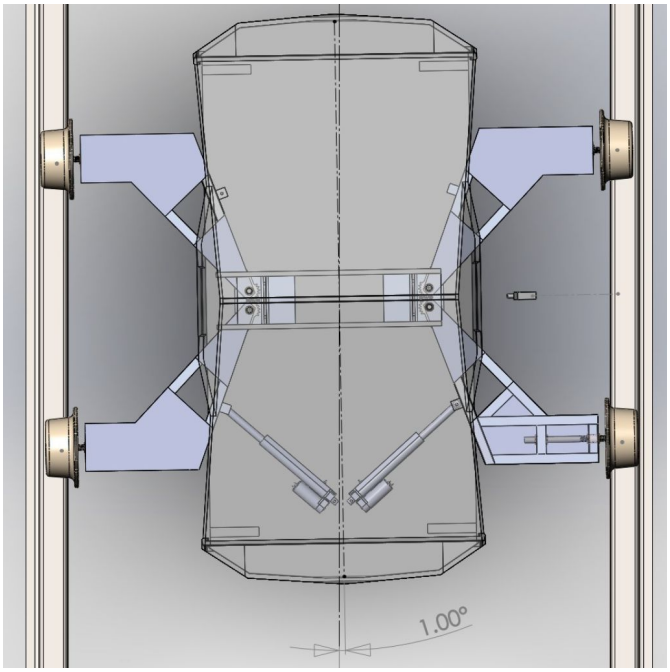


Figure 14 - Maximum skew/twist from the vertical midplane (~1.0 degrees)

Engineering Design

The current prototype design will incorporate several features to demonstrate ASIV's primary purpose. It will utilize two high accuracy laser distance sensors attached to the chassis which will measure the distance to each of the two rails and from there, the values can be converted to gauge values (the distance between the two rails) given the fixed distance between each sensor. Gauge was selected as the primary characteristic to measure since variations in this value can at best cause undesirable ride characteristics, and at worst cause a derailment. Additionally, the vehicle will be self propelled using one electric motor for each of its four wheels to demonstrate its ability to autonomously travel across and measure for defects at any part of the subway system with onboard power for propulsion and electronics provided by a battery. In accordance with this goal of autonomous and self-propelled travel, control systems will be incorporated to allow the ASIV to travel at a constant monitoring speed via Arduino microcontrollers. A unique wheel-retraction and collision avoidance mechanism would also be implemented to allow the ASIV to fold itself into the rail bed, out of the path of any oncoming revenue trains. These features are further discussed in the following sections.

Mechanical Design

Finalized Motor Design

ASIV's propulsion system will consist of 4 DC electric motors that drive each of the vehicle's 4 wheels via belts attached from the shaft to each axle. The need for multiple motors was primarily due to spacing considerations imposed by the retraction system as well as the overall compact sizing requirements of the vehicle. Thus, unlike the alpha prototype, this final design will not have its wheel sets connected by long axles. There are several added benefits to this design selection:

- The omission of the long axles reduces stresses that could bend or deform said axles. Testing of the alpha prototype demonstrated that when the powered axle was subjected to high speeds, it tended to vibrate due to the slight deformation caused by the vehicle's weight. The short axle axles in the final design will help eliminate said vibrations.
- Since each wheel is powered by a motor, ASIV will have better traction characteristics on level and inclined tracks than configurations where only a fraction of the wheels/axles are powered. Thus, slipping is reduced.

As discussed in the previous two phases, the proposed motors were analyzed with real-world production models in mind. This would allow for standardization of parts and ease of replacement when necessary. Thus, the selected model was chosen to be one rated at 250 W with a normal operating voltage of 24 V. Exact specifications were taken directly from the manufacturer's website for the Type MY1016 DC Motor.

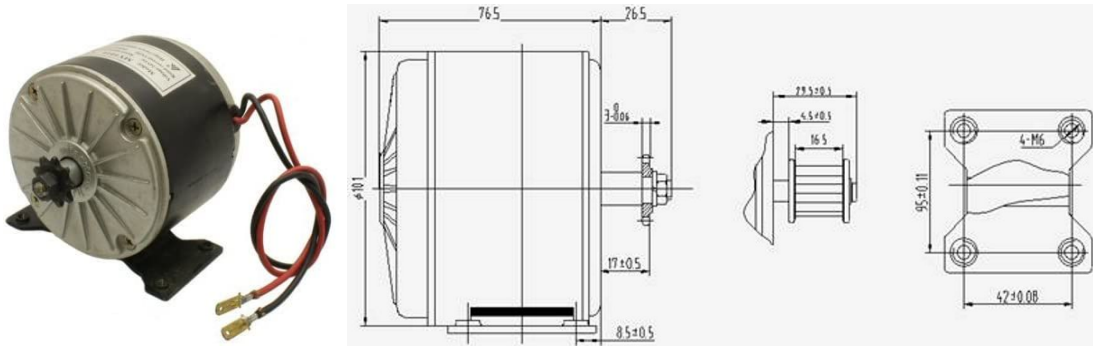


Figure 15: Photo of MY1016 Motor and Dimensions from Manufacturer

Table 2: MY1016 Motor Specifications

Motor Specifications	
Rated Power	250 W
Rated Voltage	24 V
No Load Speed	3350 rpm
Rated Torque	0.9 N-m
No Load Current	1.6 A
Max Rated Current	13.4 A
Quantity	4

The previous reports focused on the theoretical analysis of the chosen motor, while considering approximated vehicle characteristics such as weight, drag profile, wheel size, etc. To summarize this process:

1. Target values for velocity, acceleration, and max incline were selected. In this case, the target velocity was 10 mph (4.47 m/s), acceleration was 0.4 m/s^2 for level track and 0.1 m/s^2 for positive gradients, and the max incline would be 40 mm of positive climb for every meter traveled horizontally. Note that the reduced acceleration value would be used when travelling up severe grades to reduce the chance of the motors burning out.
2. The approximated vehicle specifications were plugged in to basic force equilibrium equations with the target values to yield the required tractive forces at each wheel.
3. Tractive forces were converted to respective values of wheel torque and current draw utilizing the motor's calculated voltage/torque constant.
4. The most crucial outcome of the analysis would be the conclusion as to whether the calculated max current draw would exceed the rated current of the DC motor. If this was the case, the motor selection would need to be revised.

Please note that a detailed account of these calculations can be found in the appendix as well as the previous two phase reports' technical analysis sections.

Given the target values for acceleration and velocity, the following plots were created to determine whether the rated current would be exceeded at any point.

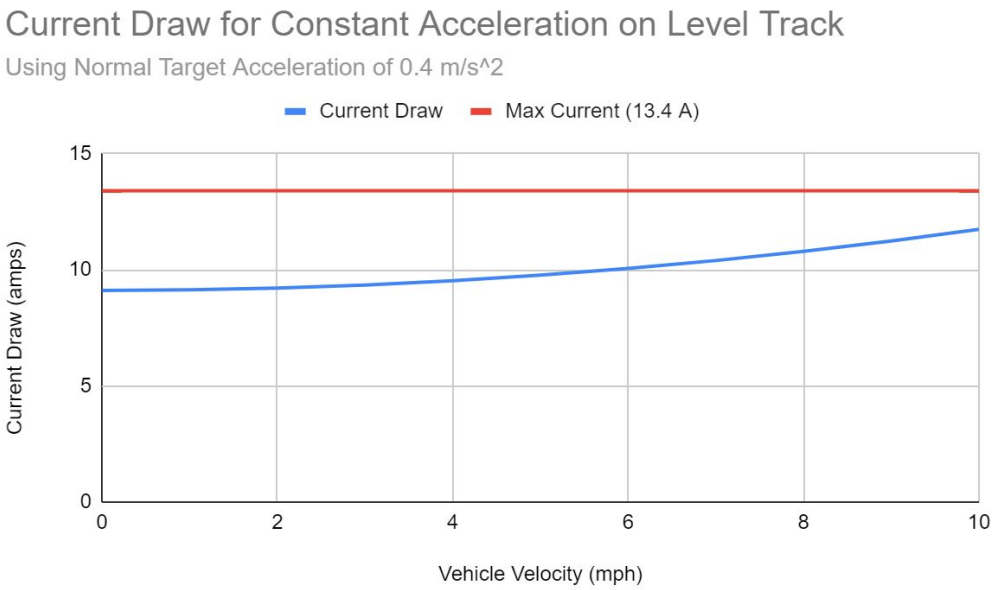


Figure 16: Current Draw vs. Velocity

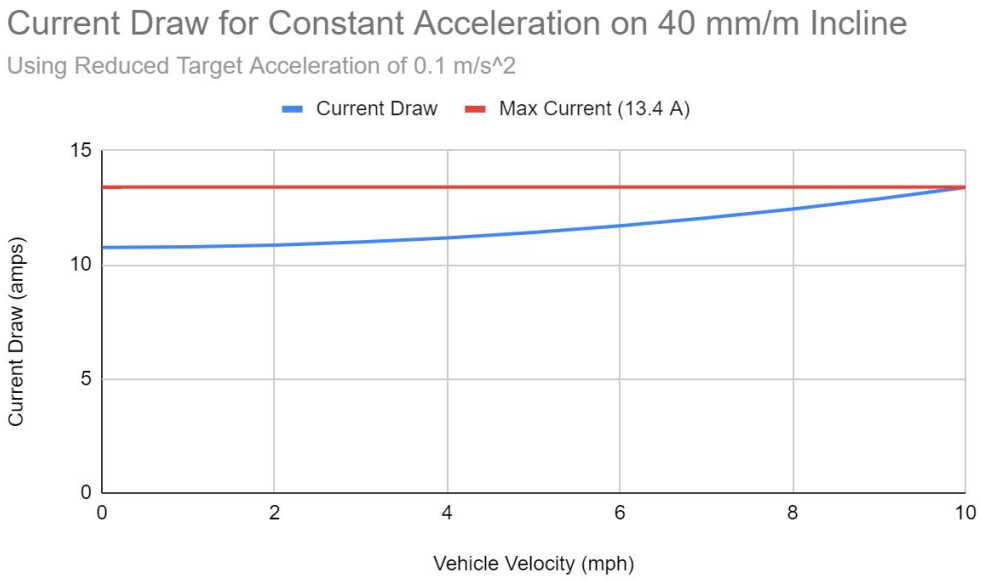


Figure 17: Current Draw vs. Velocity Moving Uphill

In both cases, of level track with the normal acceleration value as well as on ascending track with the reduced acceleration value, the current draw from each motor did not exceed the specified motor's max current of 13.4 amps. Thus, the motors would be suitable for the target values and would not likely burn out due to excessive current. Note that the current draw is calculated conservatively towards the upper range of velocity since in reality, the motor

controller would yield a smooth decrease in acceleration as it approached the target speed rather than a constant value throughout. As such, the current draw would not be as high as what was plotted above.

It was also useful to determine how the overall propulsion system would handle over long distances when paired with the suggested power source. In this case, the power source would be a single onboard 18 Ah battery based on an Okoman 6S5P rechargeable Li-ion model. Approximating constant speed at the target velocity, the following values were calculated.

Figure 18: Photo of Proposed Onboard Battery Solution

Table 3: Calculated Values for Range with Selected Motors and Battery

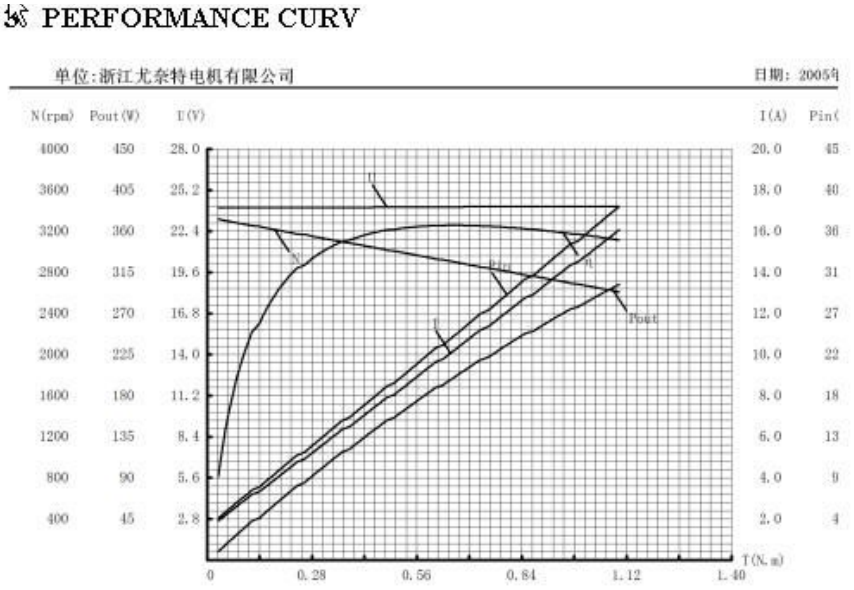
Performance at Target Speed (10 mph)	
Required Tractive Effort	5.10 N
Required Torque	0.2039 Nm
Current Draw	4.580 A
Battery Life	3.930 hours
Equivalent Distance Traveled	39.30 miles

Given that, most of the subway lines on MTA's subway system did not exceed a value in the range of 30-35 miles from end to end, the calculated range was deemed adequate for prototypical purposes.

Suggested Testing Schemes for Propulsion

Originally, it was intended that ASIV's propulsion systems would be tested directly and physically in the form of a built-out beta prototype. However, the COVID-19 outbreak has made such tests difficult, if not impossible to perform before the end of the academic semester. Consequently, a few suggestions are listed below pertaining to possible testing methods for the motors and overall propulsion systems:

1. Run unloaded tests on each DC motor to confirm certain specifications given by the manufacturer match with the actual resulting values. For example, it would be helpful to verify the values used to calculate motor constants. A multimeter could be attached to the motor while it runs while voltage and speed are recorded to find the back-EMF constant. Additional checks for quality and accuracy can be performed by comparing test performances with the curves also provided by the manufacturers. (the figure below is may not exactly reflect performance of the MY1016 model shown previously)



Ref. Point	Torque/N.m	Speed/rpm	P-out/W	Volt/V	Current/A	P-in /W	Efficient/%
No Load	0.03	3320	9.39	24.02	1.91	45.81	20.49
Max. Efficient	0.61	2932	188.86	24.07	9.64	231.97	81.42
Rated Load	0.87	2767	250.82	24.09	12.93	311.50	80.52
Max.Torque	1.10	2610	302.00	24.11	16.08	387.69	77.90

Figure 19: Example of Provided Motor Specs from Manufacturer

2. Loaded tests should be performed using a similar test bed utilized for the alpha prototype. If a full prototype can not be assembled with all components, an analog version with appropriate added weight and 4 motors should be prepared. The test bed should consist of a suitable amount of track to allow ASIV to accelerate to its target speed of 10 mph. This likely indicates that it will need to be tested on industrial,

vehicle-carrying steel rails since a large amount of material may be unavailable for acquisition and assembly as was the case for the wooden test rails used previously (this would likely come in the form of access to a lightly-used industrial siding rather than a revenue mainline track). If the test track is not inclined, it may still be possible to simulate the added resistance of gravity on positive gradients by simply adding more weight to the test vehicle. In all cases, current draw should be monitored to each motor and acceleration should be recorded or calculated to verify that the analysis performed previously was accurate. An example of a suitable test bed might be a 50 meter section of track since kinematic equations indicate that it will take almost half that distance to reach the 10 mph target speed at 0.4 m/s^2 of acceleration.

3. The battery should be tested to verify the range calculations. Ideally, this would involve running the appropriately weighted test vehicle on a subway line used for revenue service as this would be the best way to observe the typical amount of power is drawn as the vehicle traverses the track till the battery is depleted. However, given the difficulty associated with getting access to revenue service trackage, another alternative may be to use a commercial battery load tester. While this will not yield a value for range, the load tester can help in verification of the battery's health. In this case, a battery will have either passed or failed the load test based on the amount of decrease in voltage over the battery terminals when subject to the applied load at a range of temperatures.

Beta Prototype

The beta prototype has been revised and finalized since Phase 5. Previous goals of compacting the landing legs were achieved and implemented in the design. Additionally, actuators were added to control the retraction and extension of the hinged sensor mount. Shown in the figure below is the final beta prototype CAD model.

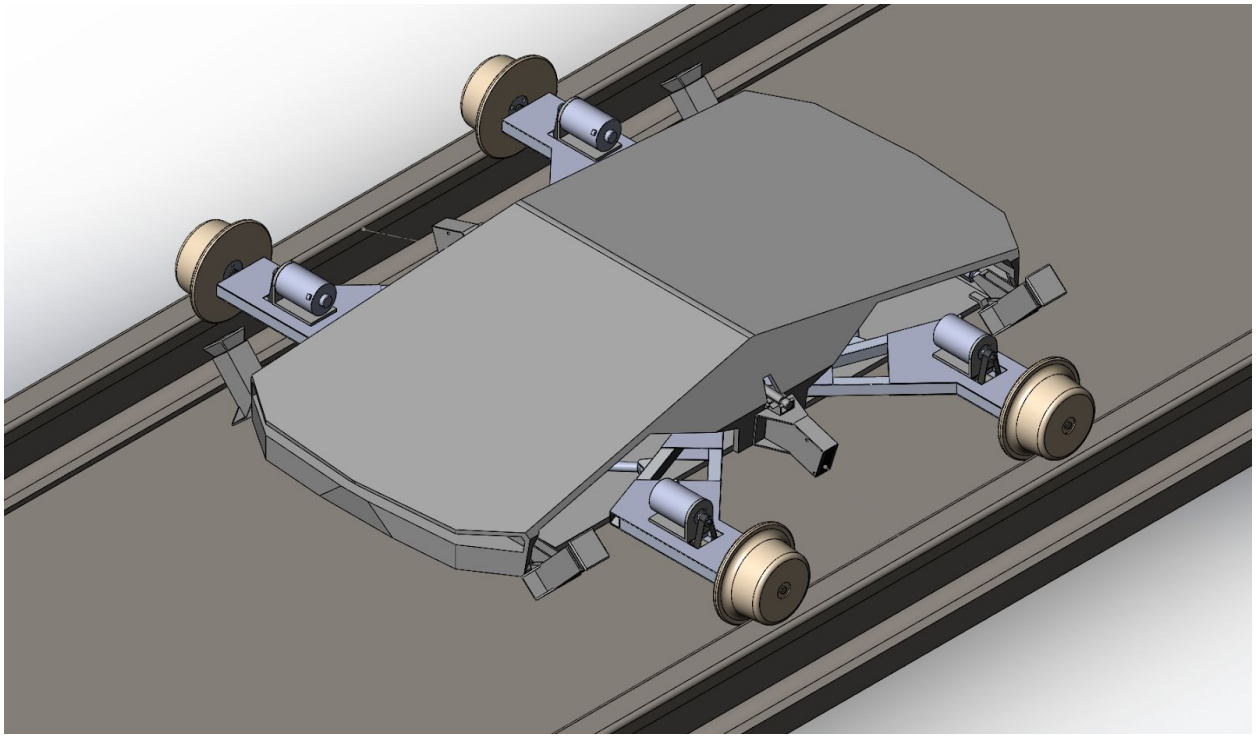


Figure 20: Beta Prototype Final Design

With this CAD model, plans are laid out for developing a fully functional beta prototype. Some features were excluded, such as fasteners and fastener holes, due to the already high part count and file size. It was learned that these features will inevitably change when a manufacturing plan is laid out per the machinists recommendations. The beta prototype offers a flexible and highly modular design. If fabrication could have occured, the prototype would have been built in several stages, and many of the parts are in independently mounted subsystems. A full list of parts (bill of materials) for the entire beta prototype assembly is located in the appendix. Shown below is a see-through isometric view of the beta prototype with all of the components mounted.

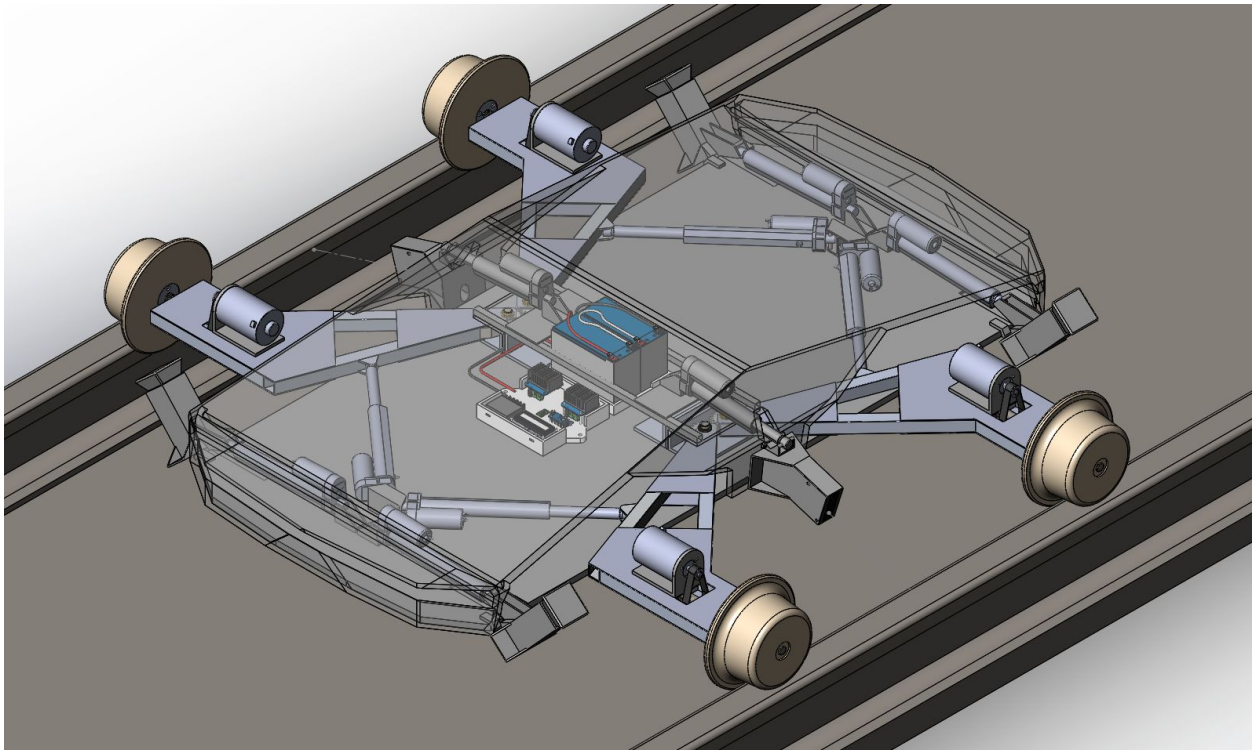


Figure 21: Beta Prototype Interior View

As shown in the figure above, the prototype employs the extensive use of linear actuators to control many of its components and key functions. Four actuators are used to control the wheel-base subsystem's retraction and extension. Shown in the next figure is the retracted configuration of the wheel bases. The four linear actuators retract to pull on each of the four wheel bases. Each linear actuator has an encoder to ensure that all four actuators act synchronously and do not misalign the wheels while moving. Limit switches can be used at the maximum and minimum positions to provide actuator homing for added redundancy, and as a fail safe to protect the actuators from moving too far in either direction.

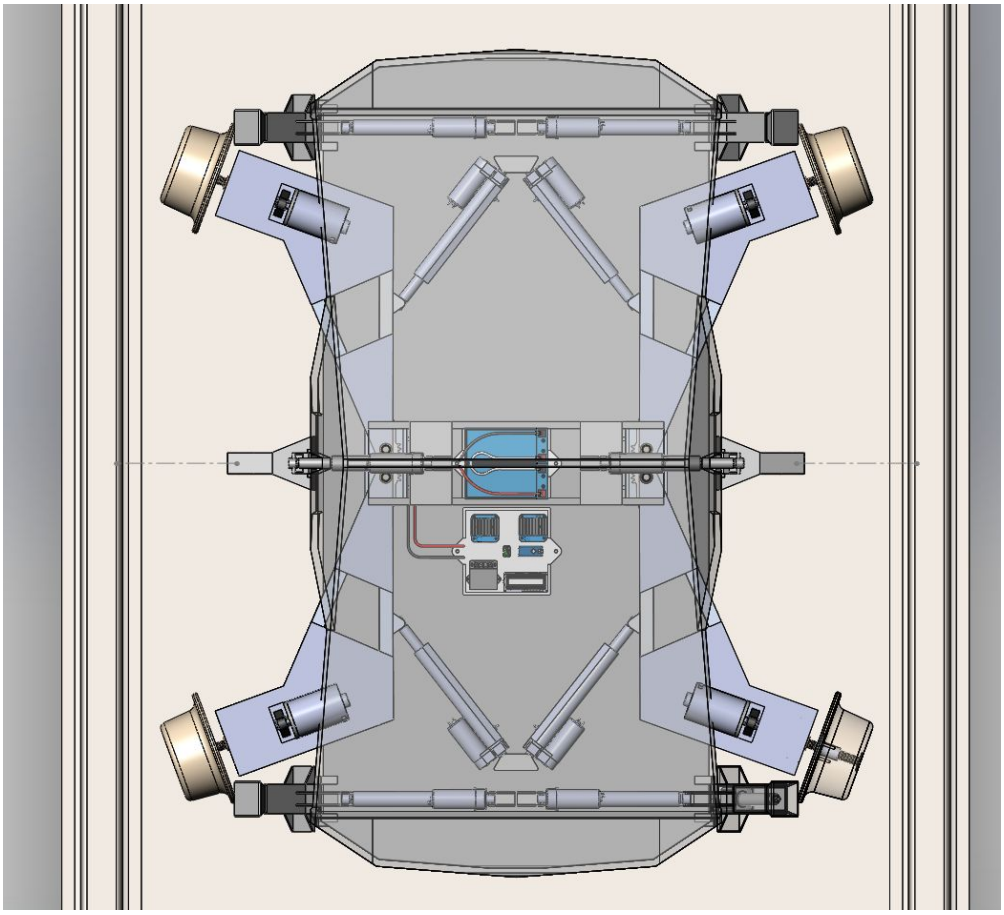


Figure 22: Retracted configuration of wheel bases

Four actuators are used to control the landing leg susbystem's retraction and extension. Four compact stepper actuators, placed inside each leg, control the telescoping portion of the leg when the landing legs are extended.

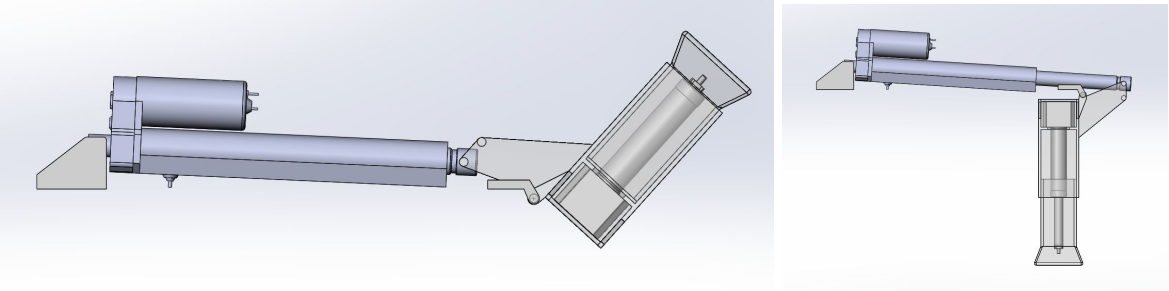


Figure 23: Retracted (left) and extended (right) configuration of landing legs

Finally, two smaller actuators control the retraction and extension of the hinged sensor mounts. The sensor mount has a large surface area that sits flush on the side of the vehicle to reduce any potential vibrations and increase measurement accuracy and stability. Shown below is the retracted and extended configuration of the sensor mount on the side of the vehicle. Magnets on the side of the vehicle and on the sensor mount will ensure the exact alignment of the mount after an extension movement and further reduce vibrations in the mount.

The wiring for the sensor is routed through the mount and into the chassis, with some slack and a stress relief point in the wire to accomodate for the movement and protect it from damage. Similar methods are employed for the wiring of motors and the stepper actuator inside the landing legs, as these are all moving components that also need to be powered.

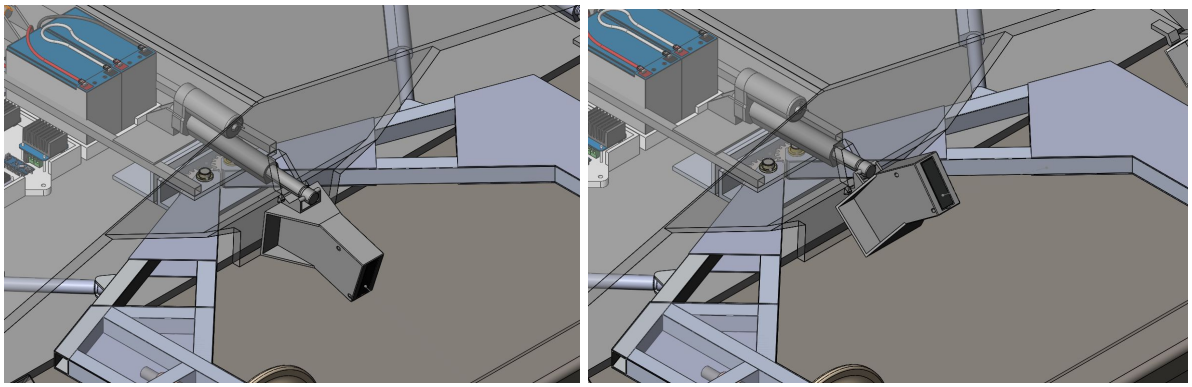


Figure 24: Retracted (left) and extended (right) configuration of sensor mount

In Phase 5, it was mentioned that a simulation of the vehicle moving on the track and collecting data was desired to be completed for this phase. The simulation was set up similar to the video presented in Phase 5, but calculating the SolidWorks Motion Analysis resulted in many issues and errors. The goal proved to be too ambitious for the software and the limitations of the Motion Analysis feature were revealed. There were issues with contact detection and parts going through each other because the software could not accurately compute the motion of each part. Limiting the number of parts in the simulation to only the wheel-bases and creating rigid groups to limit movement within the vehicle still did not work and no results could be produced. After many days of trial and error, SolidWorks Motion Analysis was deemed unsuitable for this project, but at this point, it was unfortunately too late to try to explore and learn other software.

Finally, a stress analysis of the updated wheel-base design was redone to confirm previous results. The wheel-base was fixed at the axle mounting holes, since this part will be rigid and supported by the stronger stainless steel axle. Loading of 100N was applied to a small contact surface at the hinge point on the top of the wheel base. This approximates a 400N (~41kg) load of the entire vehicle. It is important to note that the loading area will be larger, so this is a worst-case scenario. The results of the stress analysis are shown below.

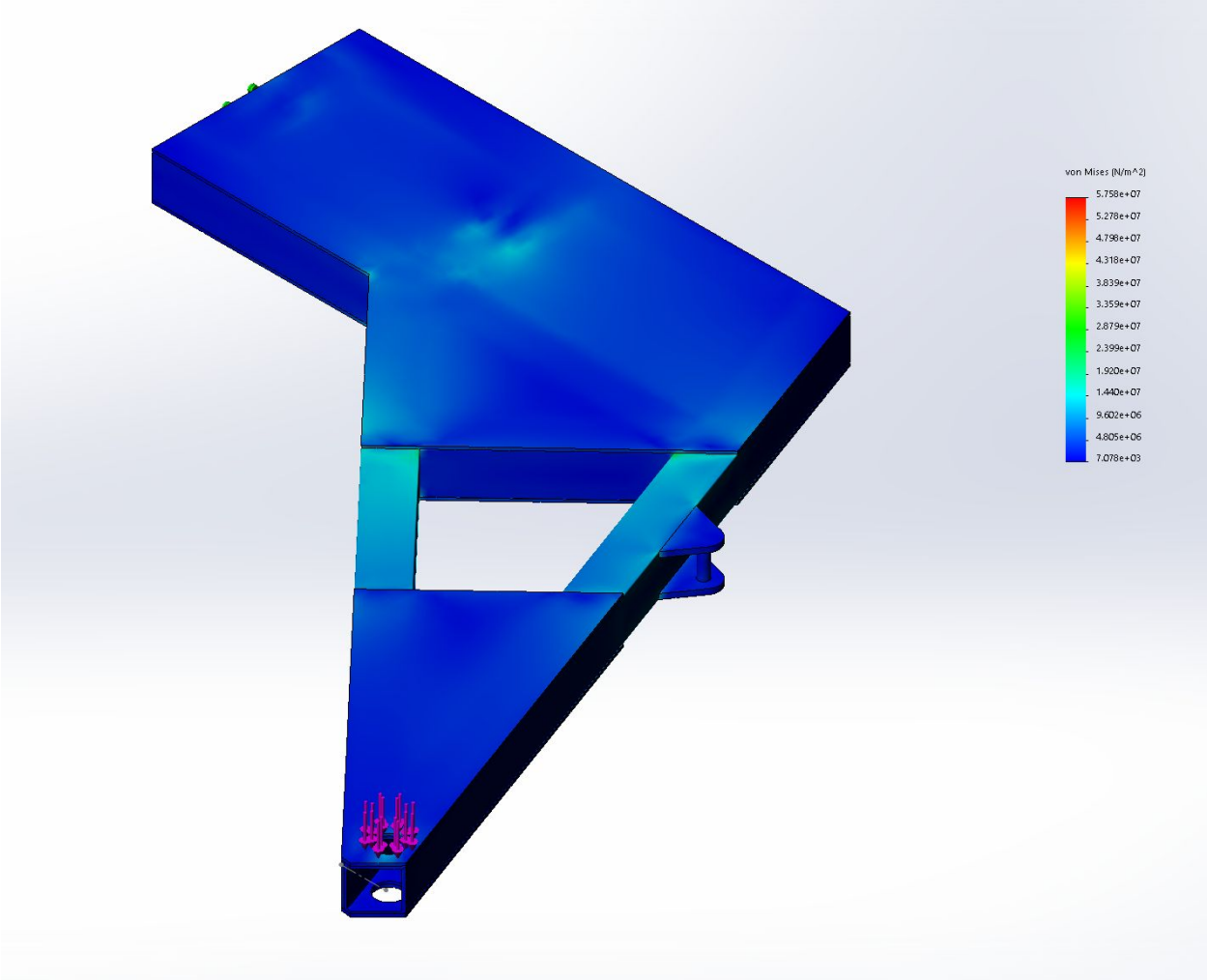


Figure 25: Wheel-Base Stress Analysis

The results above show a maximum stress (von Mises) of 57.6 MPa. This is well under the stress that the aluminum tubing used for the structure can withstand, and far from that of the steel gusset plates on each side. This results in a factor of safety of 4, which is high for this application, but does not bring any negatives such as high cost or high machining time. The steel plates provide a large amount of extra rigidity. They can be minimized, but they provide a useful and versatile mounting surface for components such as the motor and additional sensors that could be added in the future to create a 6-sensor array for measuring curvature of the track.

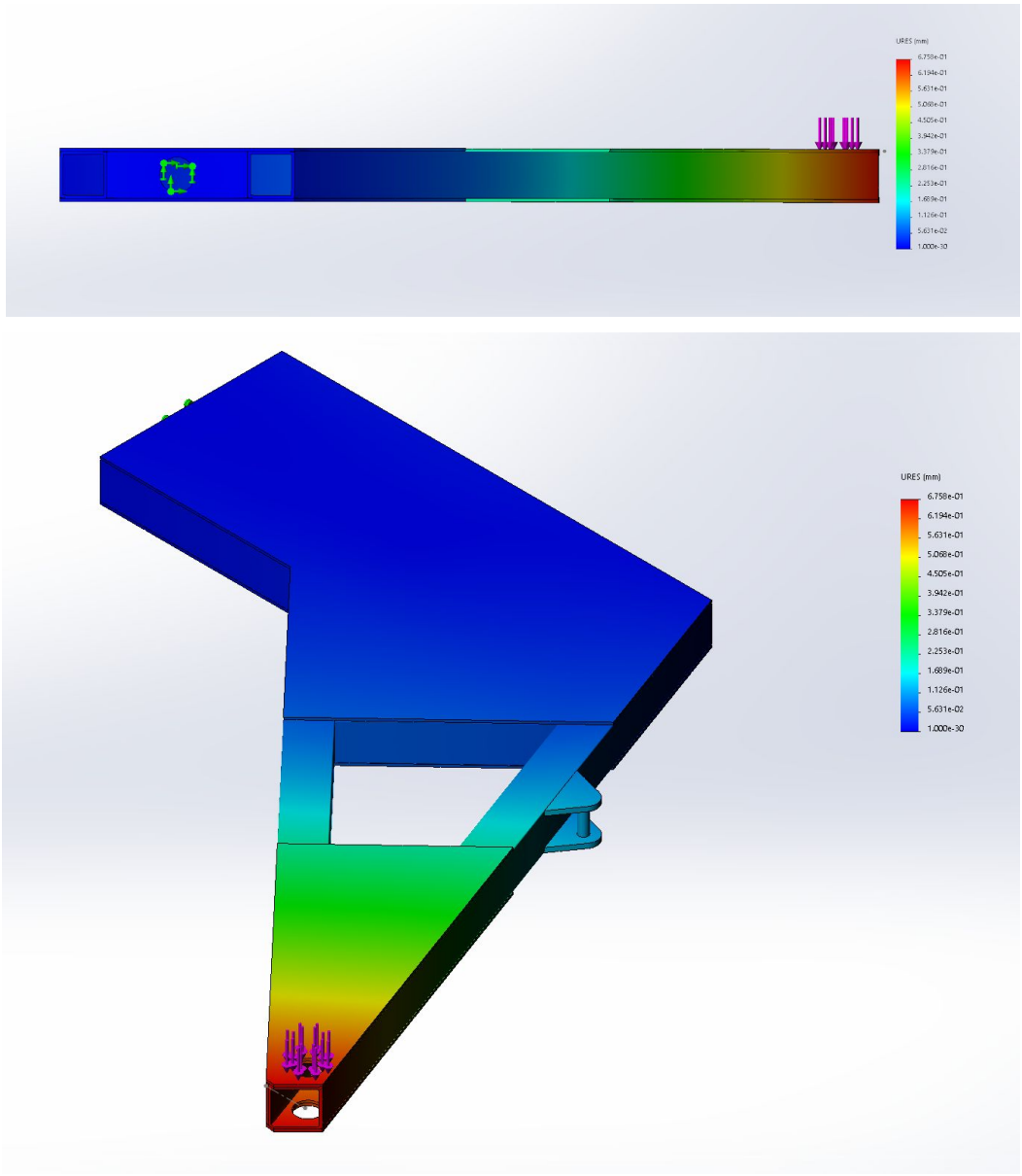


Figure 26: Wheel-Base Deflection under Stress

Additionally, the results above show the deflection of the wheel-base under load. The maximum deflection is around 7mm in this analysis. In practice, the wheel-base will not be able to deflect this much due to the surrounding mounting structure at the hinge point, which will only allow a couple of millimeters of movement vertically due to small clearance. In both cases, the deflection will be very low and will not impact the performance of the vehicle or its measurement systems, which can both compensate in various ways.

Electrical Design

The goal of the electrical design is to add the functionalities of measurements and add the autonomous features to the vehicle. If brought to the market, ASIV is expected to be fully functional without human intervention, however, this is beyond the capabilities of our team and no longer plausible due to the setbacks of the coronavirus pandemic. Instead, the team moved their deliverables to create a documented guide on how each of the components purchased worked and how all of them would fit together, as it was no longer possible to combine electrical and mechanical sections in our beta prototype. With such a guide in place, any new teams or any interested parties would be able to set up their own circuits and get a strong understanding behind the wiring and workflow developed by the ASIV team. In the end, multiple electrical circuits were tested and confirmed to work independently of each other, and are expected to work when combined together, but more physical testing would have been ideal.

The group began by simplifying the problem that the robot would face into separate tasks and tackling each of those individually. Similarly, the group brainstormed the best sensors and measurement techniques to gather the information necessary to have the robot make decisions on its set path. By the end, the following objectives were laid out by the team and tackled in separate Arduino programs. A complete flowchart of operations is found on the next page.

1. Collect Measurements
    a. Measure
        i. Gauge
        ii. Crosslevel
        iii. Distance from start
    b. Collect the data to be analyzed later
        i. Save to SD card
2. Movement
    a. Varying speed and direction
    b. Trigger Evasion and Return Once Safe
        i. Basic Obstacle Detection
    c. Wireless Communication Commands

With these objectives laid out, the group worked through multiple iterations of Arduino code to have each of them successfully implemented and created guides for circuit diagrams and baseline code for future work to be picked up on and completed.

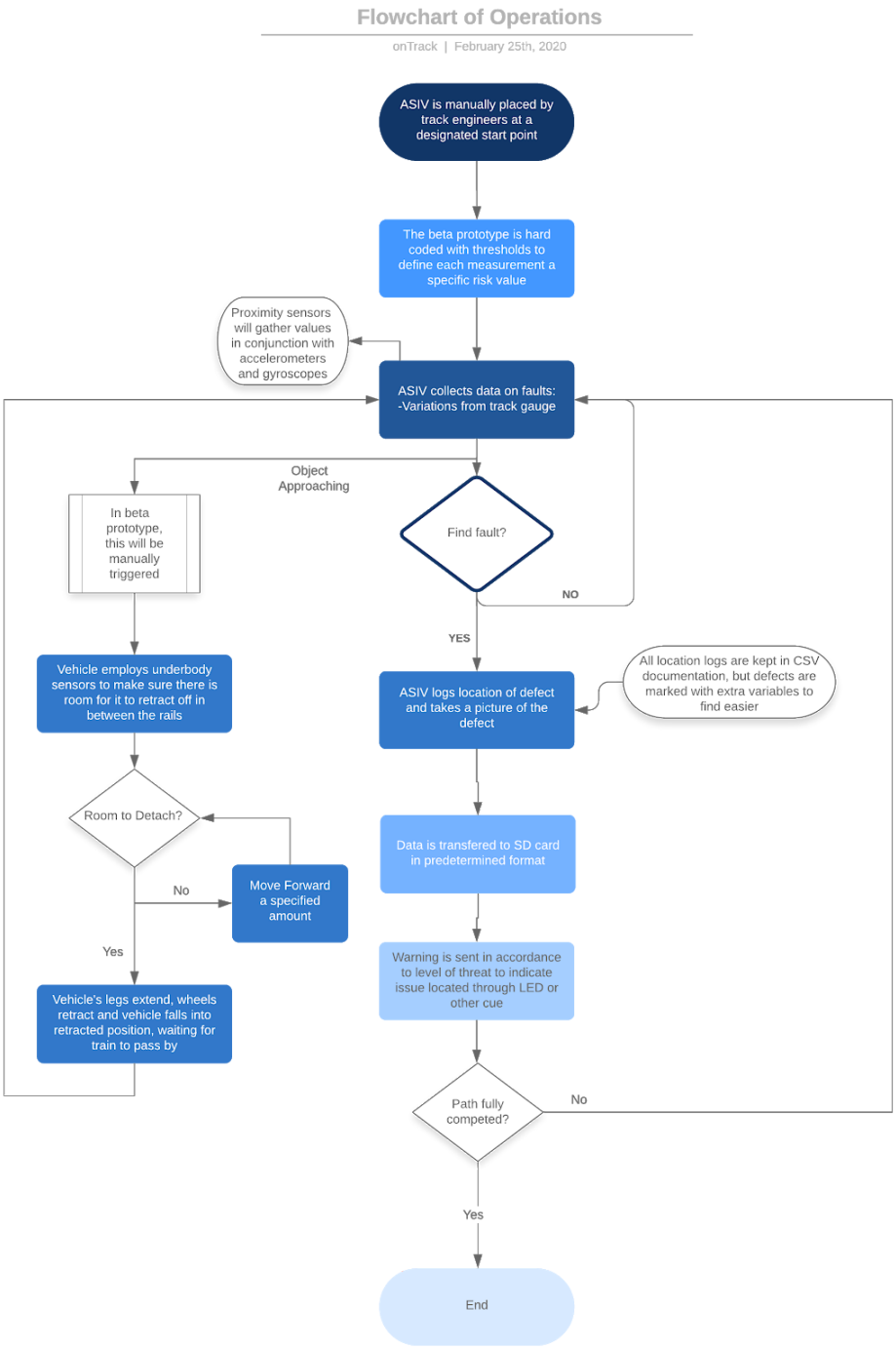


Figure 27: Initial Flowchart of Operations

While the robot was determined to be autonomous, the team decided that it would be best to give the operator deploying the robot a control panel for troubleshooting as well as for manual input. Given the state of legislature in the rail industry, some networks would not feel safe allowing an autonomous robot onto their track without manual control of some kind. While

we included the ability to alter speeds and deploy the retraction mechanism manually, on/off switches were also included to choose which measurements would be made on the track.

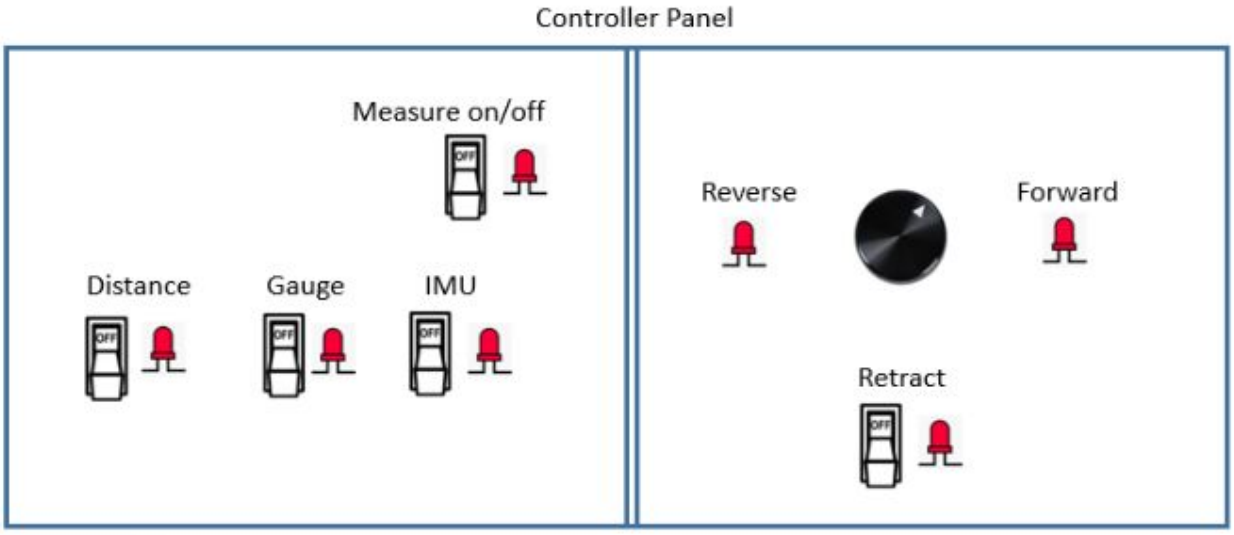


Figure 28: Control Panel

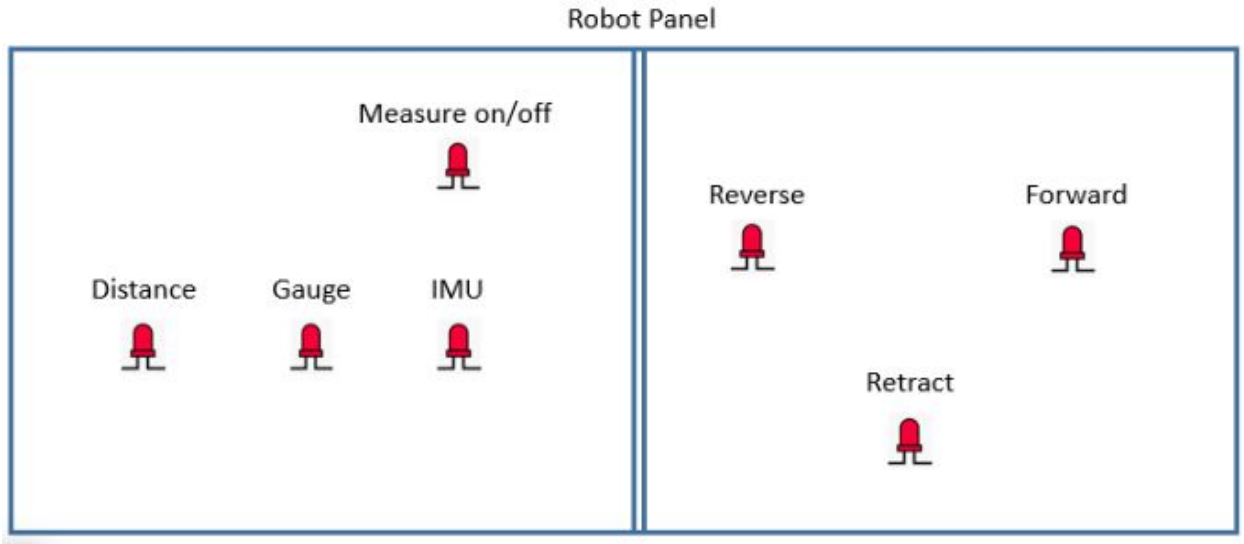


Figure 29: Robot Panel

Given the team was unable to do much physical testing with circuits combined together and with them installed onto the chassis itself, the physical testing presented was developed in a “cookbook” style allowing future work to be done on the project and complete the initial vision of the project. The group created a folder of baseline codes in order to allow anyone to pick up the project and complete the coding to make it run effectively as well as have the flexibility to change parameters and thresholds to suit their needs. Furthermore, a document was written up on each of the sensors that was used and circuits that were built around them to allow for efficient wiring and understanding of each of the connections and those have also been placed

in the same folder. The following section goes through detailed descriptions of the electrical circuits created and run in arduino as well as what is expected of the final circuitry and code when all parts are combined together.

List of Parts Coded for

- Arduinos
- MicroEpsilon Laser sensors
- Input buttons and LED indicators
- Wireless transmitters
- IMU
- Magnetic quadrature encoder
- Motors
- Linear actuators
- Stepper motor controlled linear actuators
- Stepper motor drivers
- PWM controlled motor controllers (also used for some linear actuators)
- Long distance IR sensors
- Adafruit Data Logger with SD card
- 12 and 24 V supplies
- 9V battery

Completed Code and Circuit Diagrams

The completed code exhibits the following functionality:
User control:

- Control of measurement ON/OFF
    - Select gauge
    - Select crosslevel
    - Select odometry reading
- Control of Motion
    - Knob potentiometer for forward and backward. The more you turn the faster you go
    - Trigger retraction / extension

Autonomous behavior:

- Obstacle Detection
    - IR Laser sensors detect object around 1 m away and cease motor motion

Measuring behavior:

- Measurements are taken as soon as possible (could be easily changed in code by adding loop delay or timing tracker
- Output to CSV:
    - Unique file name

- Timestamp
- Time since startup
- Count of line (each write session increases count by 1)
- Two raw distance measurements for gauge
- Absolute roll relative to plane perpendicular to pull of gravity
- Odometry distance value

Miscellaneous:

Code safety feature:

In order for the user to initiate motion, a flag must be set after startup to indicate the potentiometer is in “neutral” or “stop” position. So if the potentiometer is in forward position upon startup accidentally, the vehicle will not begin motion until the knob passes through the stop command first.

File naming:

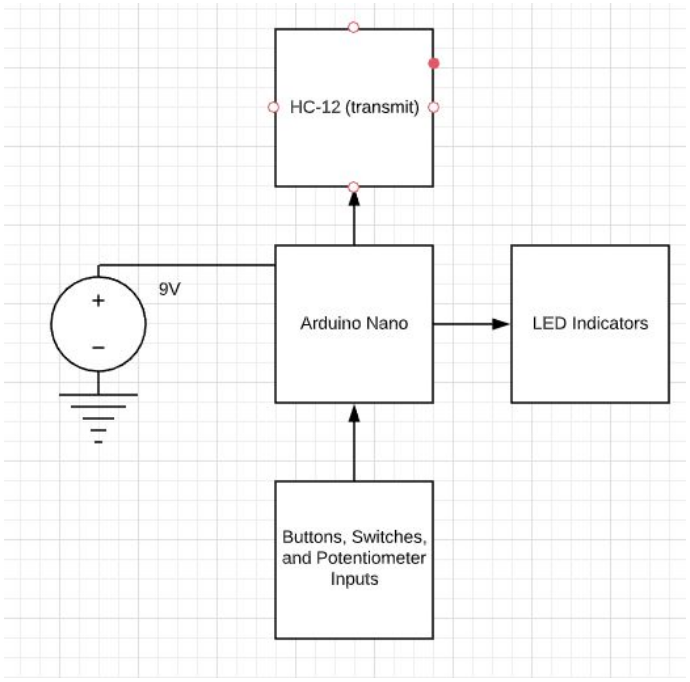
Files are set to be created each time measurement arduino is reset. Name contains a number, and if file number exists on card, new file uses next number up.

Options considered for code:

It was attempted to offer two modes of data collection. The first mode would involve taking a measurement at a set distance (dictated by odometry), and the second is taking a measurement each time the user requests one. This mode of operation would have relied heavily on hardware interrupts and serial communication, which would greatly affect internal timing and possibly the IMU measurements (which rely on timing that might be affected by the use of long interrupts). It was better to eliminate these modes for smoother operation.

While creating the code, many resources were generated. The first is a collection of electrical equipment guides. These can be found at the following link (Equipment Guides) and they contain a brief description of the equipment, necessary circuitry, and information for setting up code (some considered but not used components are included, too.). For each programmable component, a series of “baseline” codes were generated, available at the following link (Baseline Codes). In the end, the robot uses a control circuit, a measurement circuit, and a motion circuit. The control circuit is powered with one microcontroller and takes user instructions and sends them to the motion circuit wirelessly. The motion circuit contains one controller. It controls the motors and linear actuators, as well as which parameters the measurement circuit should collect. The measurement circuit contains one microcontroller and an sd card to take and collect and record data. There is another microcontroller connected to this circuit which gains odometry readings from an encoder. A general outline of the three necessary robot circuits is provided in Figure 30 and 31. The four needed codes which were constructed form the baseline codes (measurement microcontroller, motion microcontroller, control microcontroller, and encoder microcontroller) are found in the appendix and at this link (Overall Code Folder). These provide a structure for building the robot[1].

[1] The VL6180x sensor is not included since it is an optional add on for use with the mathematical twist proofs



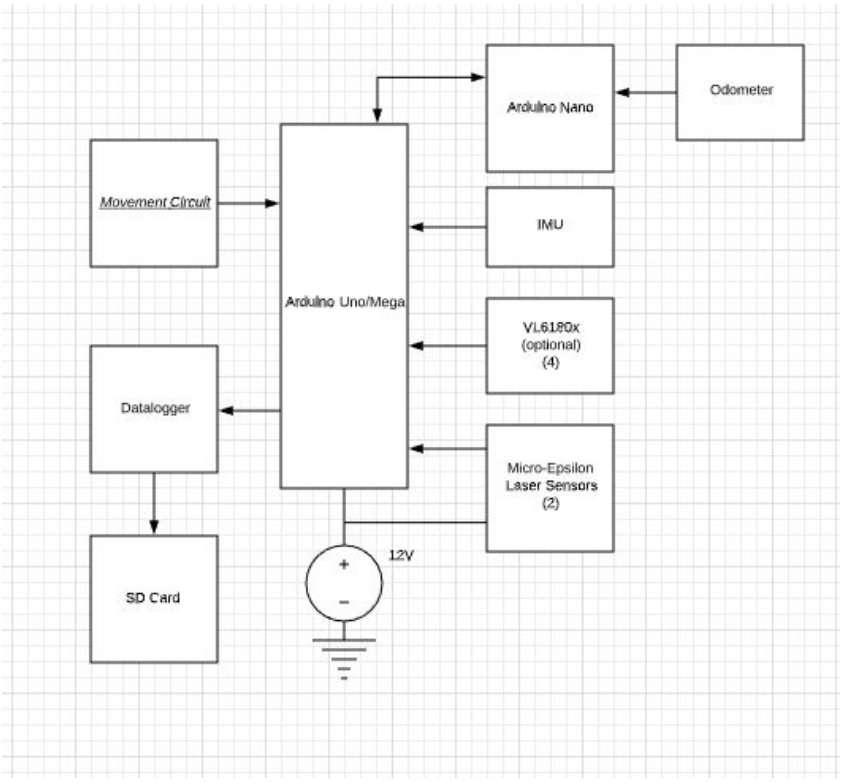


Figure 30: Control and Measurement Circuits

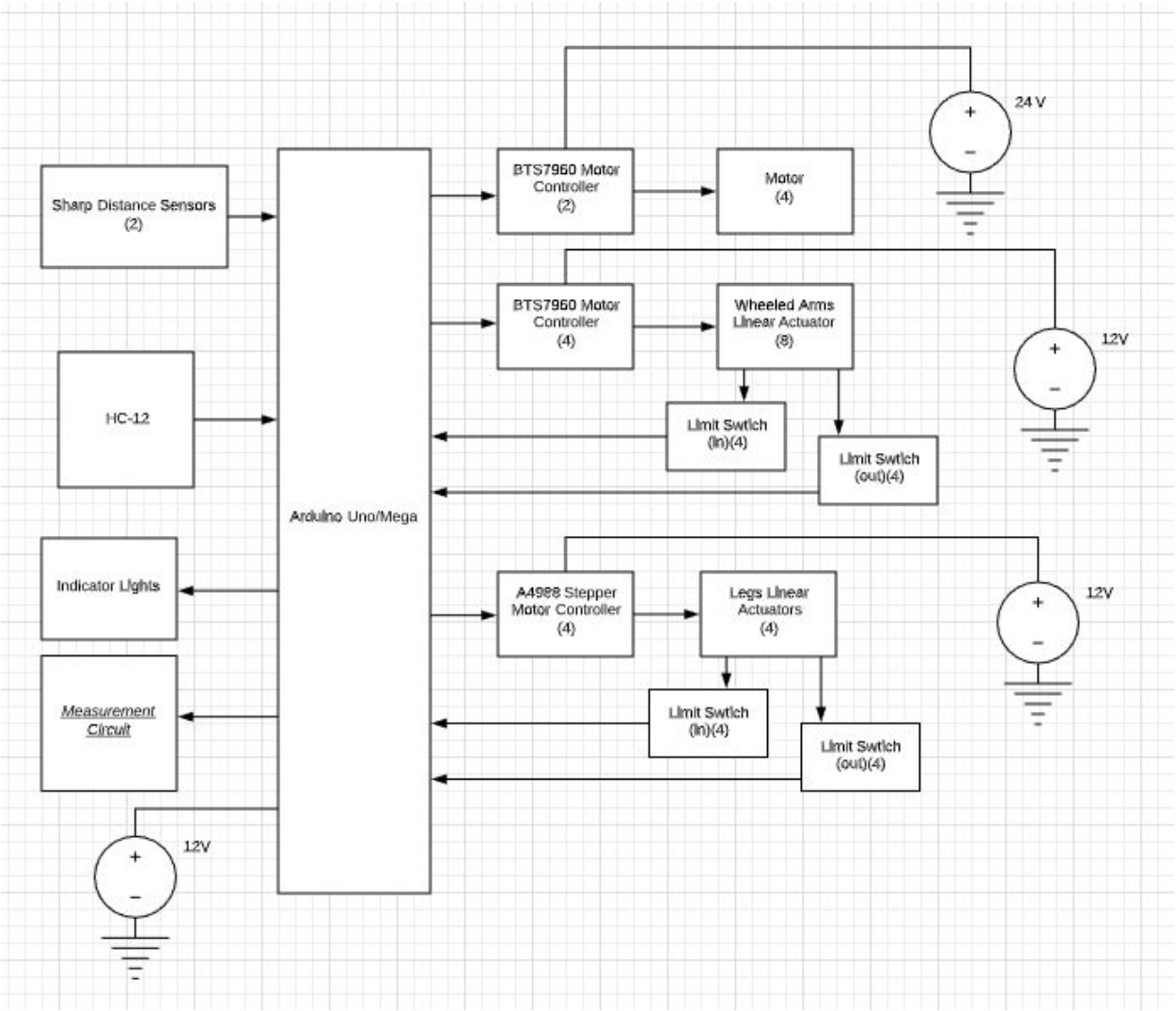


Figure 31: Motion/Central Circuit

Further Explanation Behind Designs

Gauge is the width between the rails and one of the most important measurements defined to the group by an MTA employee. To measure gauge, the group has incorporated two high quality laser distance sensors donated to us from Micro-Epsilon. These are placed on both sides of the robot chassis and the distances are added together (with the distance between the two sensors) to get the full measurement.

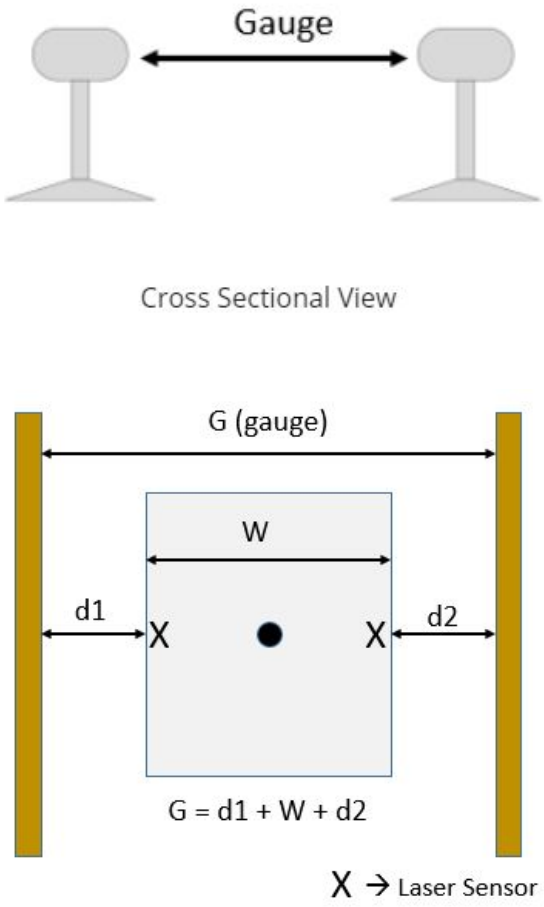


Figure 32: Gauge definition

Crosslevel is defined as the height of one rail above the other (Figure 33). This measurement can be determined by extracting the angle from an IMU, the measured gauge at that location, and trigonometry.

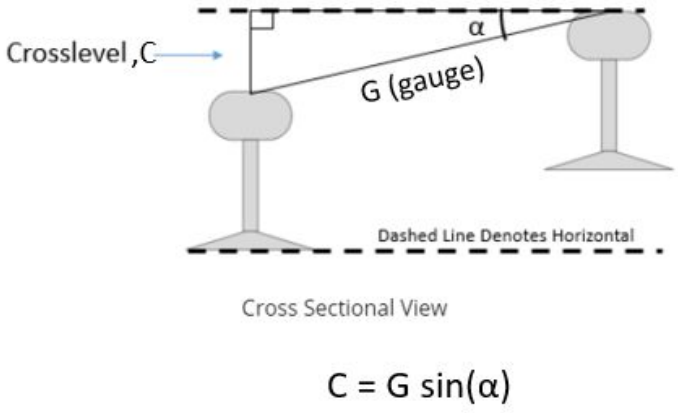
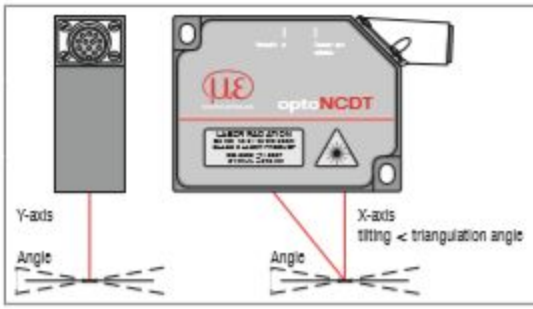


Figure 33: Crosslevel definition

Distance measurements would be done through an encoder that would act as an odometer. Given the circumstances, the group wrote code based on the encoder that was available to them, but it would be simpler in future runs to rely on an odometer unit instead.

The Micro-Epsilon laser sensors are affected adversely by degrees of tilt according to Figure 34 from the equipment manual.

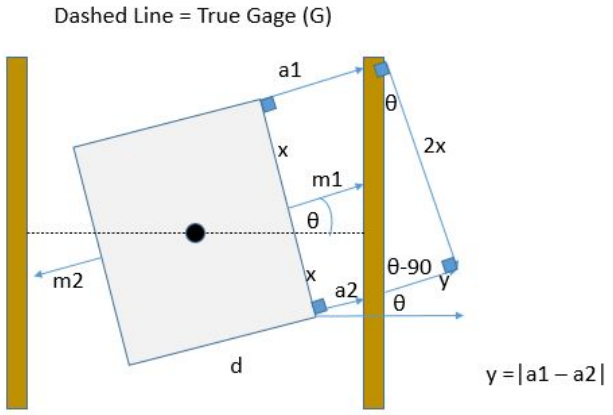


Angle	X-axis %	Y-axis %
±5 °	typ. 0.12	typ. 0.12
±15 °	typ. 0.2	typ. 0.2
±30 °	typ. 0.5	typ. 0.5

Figure 34: Laser Sensor Measurement Error

Two proofs are presented to give basic predictions as to vehicle tilt during motion. The first is for change in gauge due to tilt looking overhead. The second is for tilt around the axis in the direction of motion. Discussion of results follows:

If the vehicle twists along the rails as depicted below (Z-axis), the group would still be able to convert measured readings into accurate ones given the equations and proof below. While the chassis design was built to minimize these tilts to very small angles or sometimes to less than a degree mechanically, two additional proximity sensors of lower quality would be placed on the ends of the vehicle to measure the difference between all three sensors and determine the angle of twist. The measured value of gauge is corrected to match the actual value by multiplying it by the cosine of the angle of twist.



$\text{Tan}(\theta) = y / 2x = |a1 - a2| / 2x$

$G = (m1+m2+d)*\cos\theta$

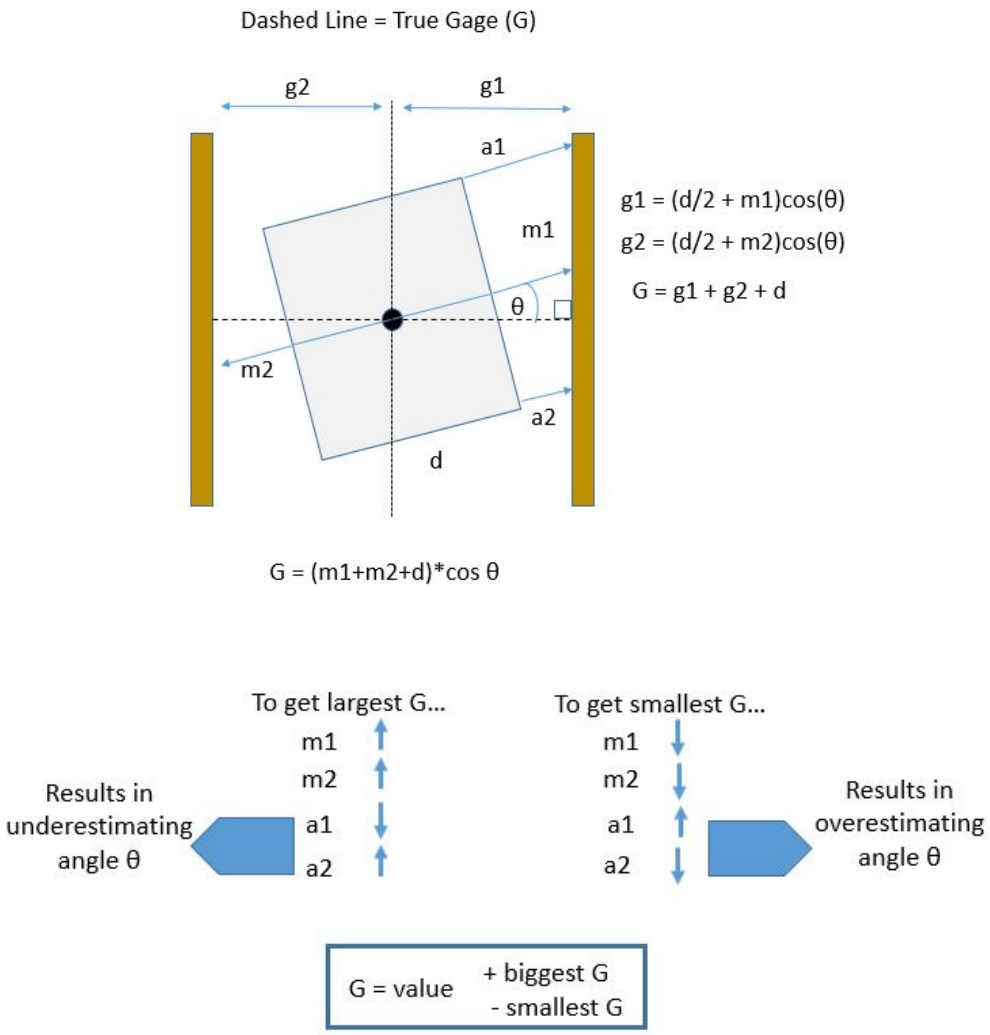


Figure 35: Proof for Rotation on Z-Axis

Similarly, if the vehicle twists along the rails as depicted below (X-axis), the group would still be able to convert measured readings into accurate ones given the equations and proof below.

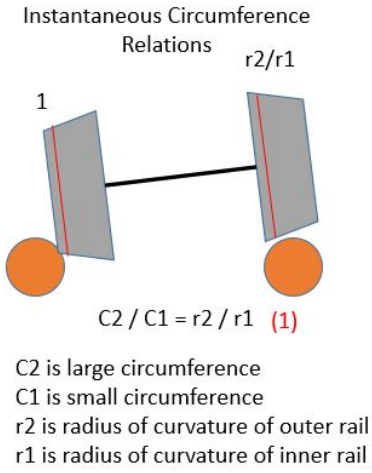


Figure 36: Tilt Calculation

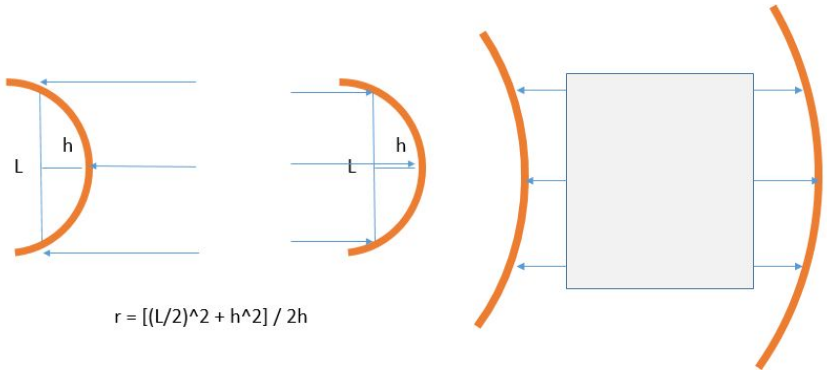
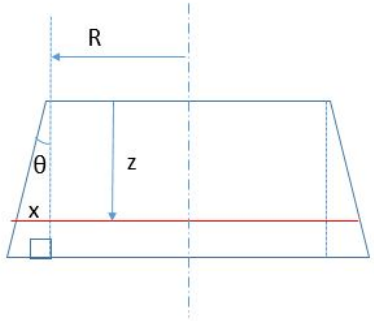


Figure 37: Laser Placement

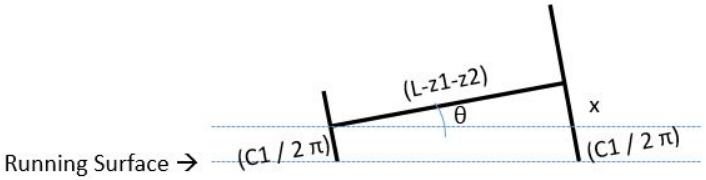
Finally, given the geometry of each wheel, tilt would be a function of curvature and the instantaneous circumference of the wheels. If one of the unknowns in the problem is fixed to its worst case, the error can be found and accounted for. The maximum theoretical tilts for both axes are shown in our CAD model and show that the expected tilts are quite small.



$C(z, \theta) = 2\pi(R + z*\tan\theta)$

Red Line is Instantaneous Circumference, C

Figure 38: Wheel Geometry



L is distance between outer faces of wheels

$x = C2/2\,\pi - C1/2\,\pi$

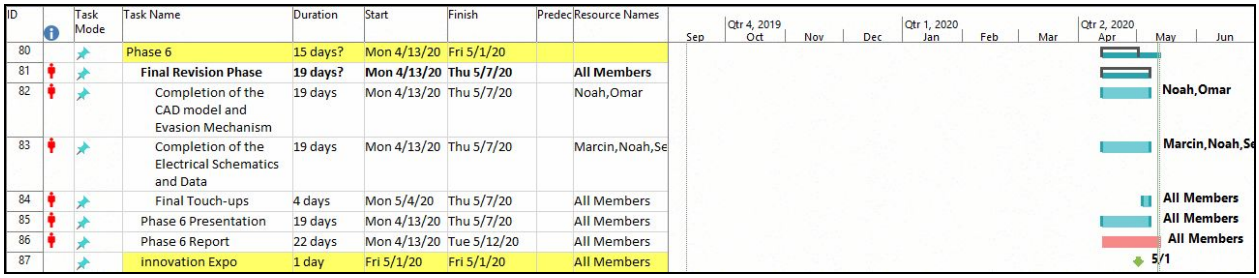
$\theta = \text{invtan}\,(x / (L\text{-}z1\text{-}z2)$ (2)

Figure 39: Tilt Angle Calculation

Project Plan

Completed Deliverables

Deliverable	Description
Transition to Virtual Prototype Development	The transition to virtual development was successfully completed as all components of the project were created and put together to outline the whole vehicle. A flow chart was created to showcase the logic for the total operation of the vehicle.
Mechanical Design Components	The mechanical components were created such as the CAD model, Evasion mechanism, and sensor placements. These are all outlined in the CAD model and provide all the necessary information for physical creation.
Electrical Design Components	All the electrical components have been expanded upon with detailed schematics to outline the functionality of each system. The electrical components were thoroughly explained to allow for exact details for future creation of a physical prototype.
Innovation Expo Preparation	The innovation expo poster was created along with updates to the team website that features videos and all past reports and presentations for the project. This allowed for anyone to see all the hard work put into the project as well as follow along with the development from Phase I.



ID	Task Mode	Task Name	Duration	Start	Finish	Predec	Resource Names
80		Phase 6	15 days?	Mon 4/13/20	Fri 5/1/20		
81		**Final Revision Phase**	**19 days?**	**Mon 4/13/20**	**Thu 5/7/20**		**All Members**
82		Completion of the CAD model and Evasion Mechanism	19 days	Mon 4/13/20	Thu 5/7/20		Noah,Omar
83		Completion of the Electrical Schematics and Data	19 days	Mon 4/13/20	Thu 5/7/20		Marcin,Noah,Se
84		Final Touch-ups	4 days	Mon 5/4/20	Thu 5/7/20		All Members
85		Phase 6 Presentation	19 days	Mon 4/13/20	Thu 5/7/20		All Members
86		Phase 6 Report	22 days	Mon 4/13/20	Tue 5/12/20		All Members
87		innovation Expo	1 day	Fri 5/1/20	Fri 5/1/20		All Members

Figure 40: Phase Six Gantt Chart

Phase 6 of the Gantt Chart, while relatively short, aimed to complete all of the aspects started in the Phase 5. The final revision phase includes all the last minute data testing as well as final touch ups on the CAD model. Most of the prepwork for the expo was done during the week of and featured the compilation of all the main strides taken for the project.

Reflection

Future Work

With the virus greatly inhibiting the ability to create a physical prototype, there is still so much that can be expanded upon for the future. The biggest project would be to create an actual physical model. With all the work and research that has been done over the past two semesters, a concrete virtual prototype was created that showcased how to build and use each of the systems within this project. Using these “cookbook recipes”, a physical prototype could be created as a continuation of this project for next year. Along with the prototype, a user interface could be implemented as it would greatly help with the use of the vehicle. The current systems use buttons and arduino boards to collect and analyze data, but much work could be done to create an easy to use user interface that displays all the data and can be used to easily program the vehicle for the length of track that it needs to measure. Since this vehicle will live on the tracks for extensive periods of time, a good user interface would help to monitor the progress of the vehicle as well as communicate wirelessly if possible. With the physical creation of both of these aspects, the prototype can finally be tested on real railway tracks to receive data and feedback on if the system works or not. This is a big step as the testing would require contact with the MTA or any other railway systems such as the PATH that can assist in testing the vehicle on unused train tracks. Not only will this help in proving that the vehicle actually works, it should also provide enough data for future sponsorship or growth. Lastly, one final way to expand the project is the addition of other key track parameter measurements. With the initial budget for this project, it was difficult to expand our focus on anything besides track gauge in terms of track parameters. The prices for the chassis and drive mechanism made it difficult to purchase other sensors and the Micro-Epsilon laser sensors that were donated would have cost around $2000 each. However, since most of the parts were purchased and the mechanisms have been planned out and created, the next phase would be to create a modular expansion that can be added to the prototype that could focus on other parameters that were initially introduced in the concept development phase of the project. The profile of the rail would be a valuable parameter to measure as it could directly influence the safety of the subway cars. Most of the major derailments come from the profile of the rail becoming deformed to the point where a subway car could slip off of the rail. This implementation to the beta prototype would greatly increase the measurements of safety and would be enough work for future expansion.

Lessons Learned

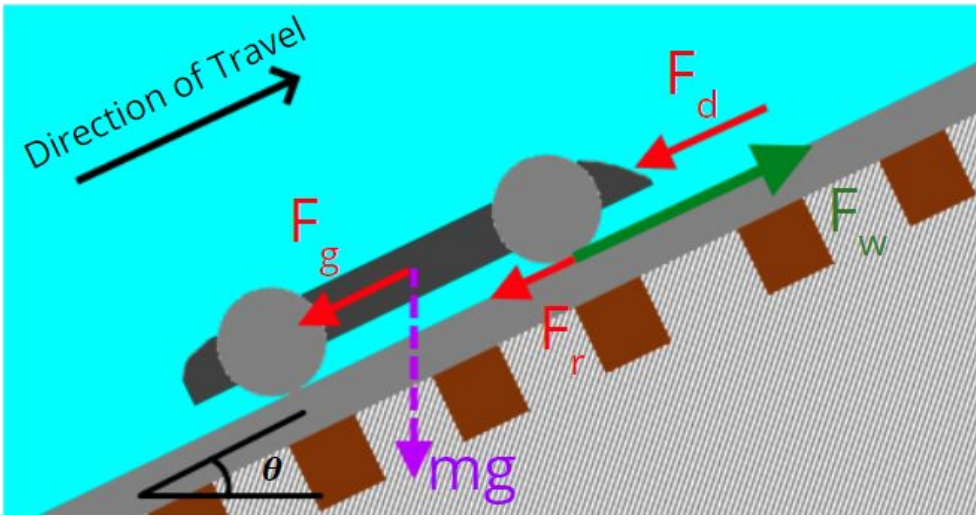
Throughout the course of the entire project, there have been many lessons learned in terms of both hard and soft skills. For the hard skills learned, much knowledge came from the research and development portion of the project. During this research phase, basic understanding of odometry, interrupt vs polling, 3D space geometry, IMU functionality, motor driving and control, organization, and government regulation was learned. During the prototyping phase, other hard skills such as learning how to use a lathe or mill was learned in the machine shop along with using the CNC machine and broaching. In terms of the electrical side, the circuitry software, programming libraries and IDE, and I²C communication were all aspects that were learned in the development phase.

For the soft skills that were learned, the group learned how to collaborate with others on a long term process. This being the biggest project of each of our Steven's career, it took some time to learn how to efficiently work within a group and bounce ideas off together. A big part in our successes was learning how to clearly express our ideas and come to a consensus on what our vision is for the project. A lot of communication with advisors and potential sponsors was also required for this project. The group needed to know how to properly communicate with these different parties in order to receive the proper feedback needed. Lastly, the team learned how to defend the vision for the project and answer questions relating to the functionality of the vehicle. Throughout the course of the year, this project was met with much criticism which hindered the progress of the vehicle. Despite the frustration, the group had to learn how to respond positively to criticism as well as how to help others understand our final vision. Overall, many lessons were learned with project development and prototyping that will continue to prove useful for years to come.

Appendix

A1 - Motor-Battery Analysis

Motor/Battery Analysis and Formulas



The forces associated with travel up an incline for the vehicle can be summarized as follows:

$$\sum F_x = ma = F_w + F_g + F_d + F_r$$

$$F_w = Force\ at\ Wheel = \frac{n\tau_w}{D/2}$$

$$F_g = Component\ of\ Gravitational\ Force\ on\ Slope = mg\sin\theta$$

$$F_d = Drag\ Force = \frac{1}{2}c_d\rho v^2 A$$

$$F_r = Rolling\ Resistance = C_R mg$$

The terms can be rearranged to solve for the input torque at each wheel.

$$\tau_w = \frac{D}{2n}\left(ma - mg\sin\theta - \frac{1}{2}c_d\rho v^2 A - C_R mg\right)$$

When using the SI units, the following relationship is true for an electric motor's torque constant and voltage constant respectively. The rated and no-load values are taken directly from the specified motor's data sheets when provided.

$$k_T = \frac{\tau_{rated}}{I_{rated}} = k_e = \frac{V_{rated}}{\omega_{no\ load}}$$

The motor voltage constant can be substituted into the following relationship to solve for each motor's current draw at a given torque.

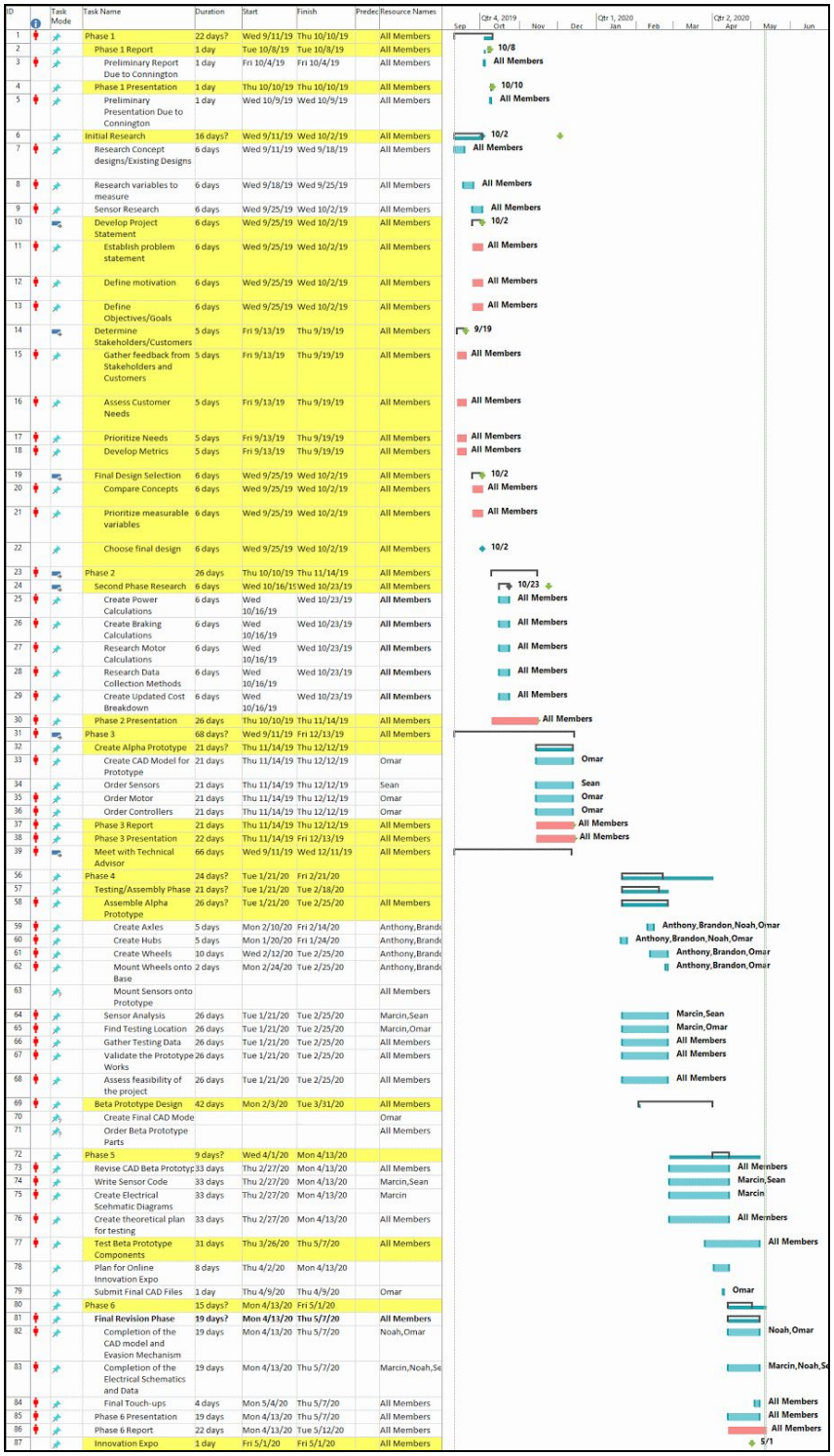
$$I_m = \tau_w \left(\frac{V_{rated}}{\omega_{no\ load}}\right)^{-1} + I_{no\ load}$$

This value of current can then be divided from a DC power source's capacity (in amp-hours) to find the battery life in hours if said current were to be held constant for that duration.

Approximated Vehicle Specifications for Motor Analysis	
Total Mass	30.6 kg
Wheel Diameter	32 cm
Projected Frontal Area	0.375 m
Drag Coefficient	1.0
Rolling Resistance Coefficient	0.002

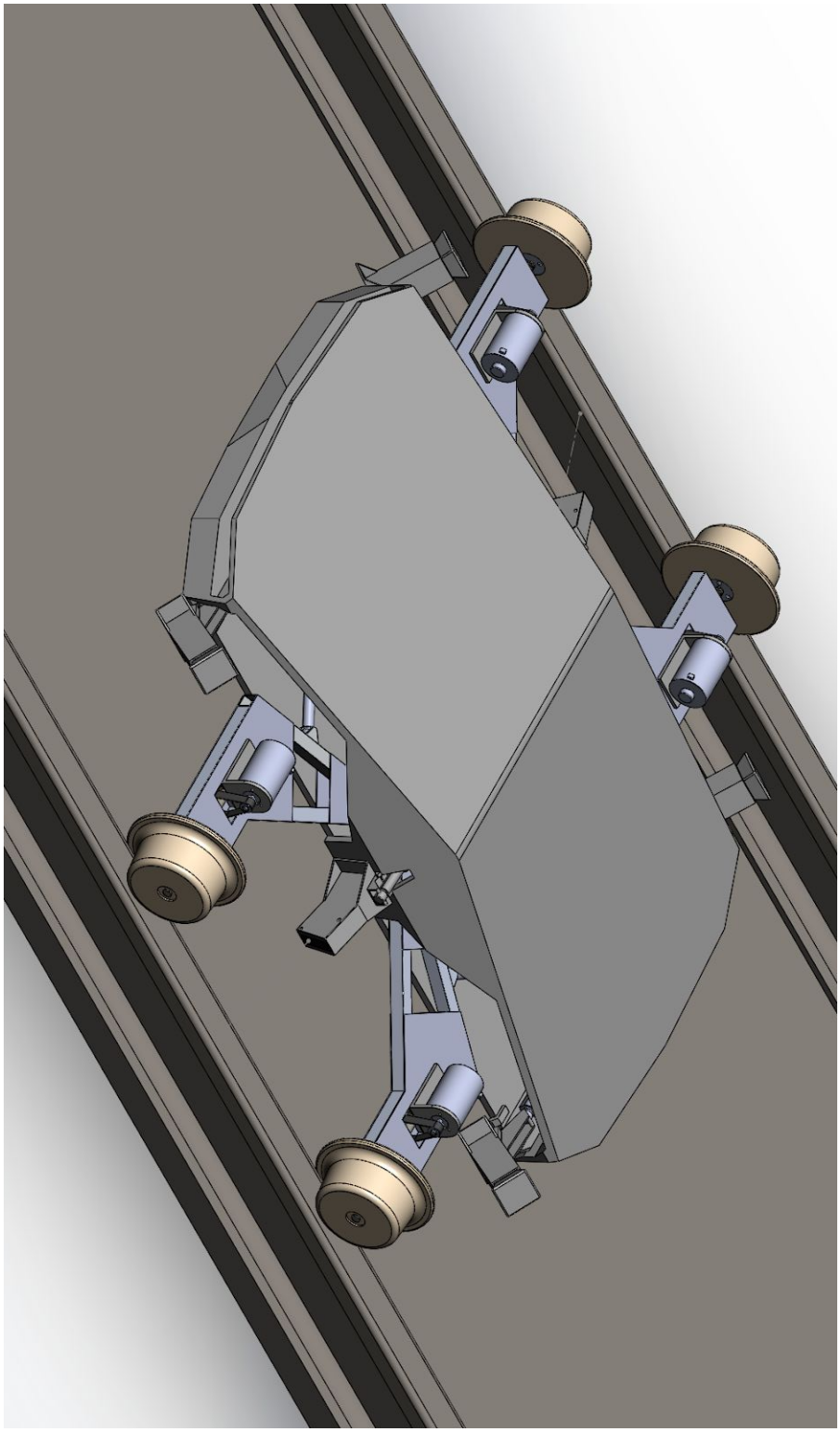
Performance Targets for Motor Analysis	
Normal Operating Velocity	10 mph
Normal Acceleration	0.4 m/s^2
Reduced Acceleration	0.1 m/s^2
Max Gradient	40 mm/m

A2- Full Gantt Chart

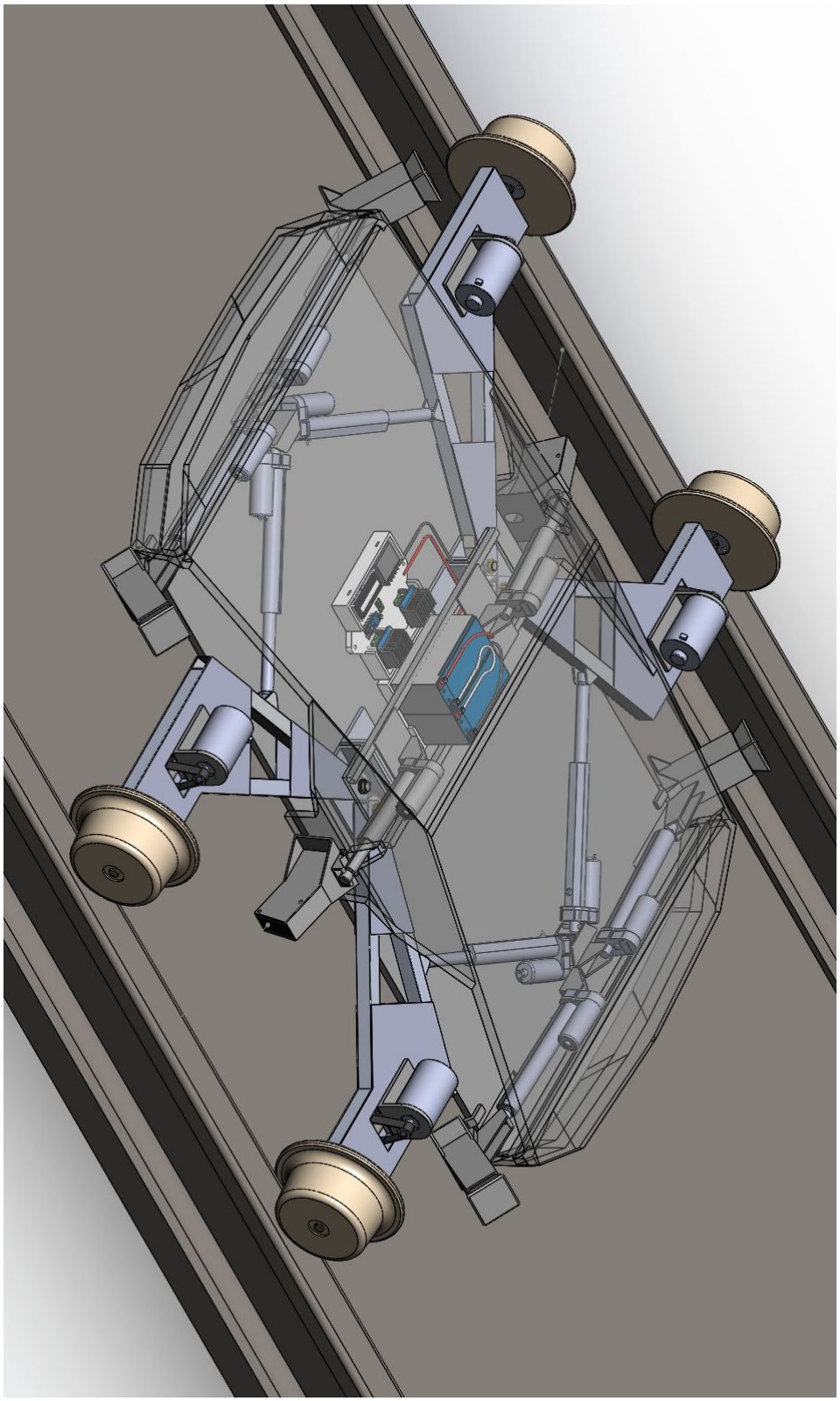


ID	Task Mode	Task Name	Duration	Start	Finish	Predec	Resource Names
1		Phase 1	22 days?	Wed 9/11/19	Thu 10/10/19		All Members
2		Phase 1 Report	1 day	Tue 10/8/19	Tue 10/8/19		All Members
3		Preliminary Report Due to Connington	1 day	Fri 10/4/19	Fri 10/4/19		All Members
4		Phase 1 Presentation	1 day	Thu 10/10/19	Thu 10/10/19		All Members
5		Preliminary Presentation Due to Connington	1 day	Wed 10/9/19	Wed 10/9/19		All Members
6		Initial Research	16 days?	Wed 9/11/19	Wed 10/2/19		All Members
7		Research Concept designs/Existing Designs	6 days	Wed 9/11/19	Wed 9/18/19		All Members
8		Research variables to measure	6 days	Wed 9/18/19	Wed 9/25/19		All Members
9		Sensor Research	6 days	Wed 9/25/19	Wed 10/2/19		All Members
10		Develop Project Statement	6 days	Wed 9/25/19	Wed 10/2/19		All Members
11		Establish problem statement	6 days	Wed 9/25/19	Wed 10/2/19		All Members
12		Define motivation	6 days	Wed 9/25/19	Wed 10/2/19		All Members
13		Define Objectives/Goals	6 days	Wed 9/25/19	Wed 10/2/19		All Members
14		Determine Stakeholders/Customers	5 days	Fri 9/13/19	Thu 9/19/19		All Members
15		Gather feedback from Stakeholders and Customers	5 days	Fri 9/13/19	Thu 9/19/19		All Members
16		Assess Customer Needs	5 days	Fri 9/13/19	Thu 9/19/19		All Members
17		Prioritize Needs	5 days	Fri 9/13/19	Thu 9/19/19		All Members
18		Develop Metrics	5 days	Fri 9/13/19	Thu 9/19/19		All Members
19		Final Design Selection	6 days	Wed 9/25/19	Wed 10/2/19		All Members
20		Compare Concepts	6 days	Wed 9/25/19	Wed 10/2/19		All Members
21		Prioritize measurable variables	6 days	Wed 9/25/19	Wed 10/2/19		All Members
22		Choose final design	6 days	Wed 9/25/19	Wed 10/2/19		All Members
23		Phase 2	26 days	Thu 10/10/19	Thu 11/14/19		All Members
24		Second Phase Research	6 days	Wed 10/16/19	Wed 10/23/19		All Members
25		Create Power Calculations	6 days	Wed 10/16/19	Wed 10/23/19		All Members
26		Create Braking Calculations	6 days	Wed 10/16/19	Wed 10/23/19		All Members
27		Research Motor Calculations	6 days	Wed 10/16/19	Wed 10/23/19		All Members
28		Research Data Collection Methods	6 days	Wed 10/16/19	Wed 10/23/19		All Members
29		Create Updated Cost Breakdown	6 days	Wed 10/16/19	Wed 10/23/19		All Members
30		Phase 2 Presentation	26 days	Thu 10/10/19	Thu 11/14/19		All Members
31		Phase 3	68 days?	Wed 9/11/19	Fri 12/13/19		All Members
32		Create Alpha Prototype	21 days?	Thu 11/14/19	Thu 12/12/19		
33		Create CAD Model for Prototype	21 days	Thu 11/14/19	Thu 12/12/19		Omar
34		Order Sensors	21 days	Thu 11/14/19	Thu 12/12/19		Sean
35		Order Motor	21 days	Thu 11/14/19	Thu 12/12/19		Omar
36		Order Controllers	21 days	Thu 11/14/19	Thu 12/12/19		Omar
37		Phase 3 Report	21 days	Thu 11/14/19	Thu 12/12/19		All Members
38		Phase 3 Presentation	22 days	Thu 11/14/19	Fri 12/13/19		All Members
39		Meet with Technical Advisor	66 days	Wed 9/11/19	Wed 12/11/19		All Members
56		Phase 4	24 days?	Tue 1/21/20	Fri 2/21/20		
57		Testing/Assembly Phase	21 days?	Tue 1/21/20	Tue 2/18/20		
58		Assemble Alpha Prototype	26 days?	Tue 1/21/20	Tue 2/25/20		All Members
59		Create Axles	5 days	Mon 2/10/20	Fri 2/14/20		Anthony,Brand
60		Create Hubs	5 days	Mon 1/20/20	Fri 1/24/20		Anthony,Brand
61		Create Wheels	10 days	Wed 2/12/20	Tue 2/25/20		Anthony,Brand
62		Mount Wheels onto Base	2 days	Mon 2/24/20	Tue 2/25/20		Anthony,Brand
63		Mount Sensors onto Prototype					All Members
64		Sensor Analysis	26 days	Tue 1/21/20	Tue 2/25/20		Marcin,Sean
65		Find Testing Location	26 days	Tue 1/21/20	Tue 2/25/20		Marcin,Omar
66		Gather Testing Data	26 days	Tue 1/21/20	Tue 2/25/20		All Members
67		Validate the Prototype Works	26 days	Tue 1/21/20	Tue 2/25/20		All Members
68		Assess feasibility of the project	26 days	Tue 1/21/20	Tue 2/25/20		All Members
69		Beta Prototype Design	42 days	Mon 2/3/20	Tue 3/31/20		All Members
70		Create Final CAD Mode					Omar
71		Order Beta Prototype Parts					All Members
72		Phase 5	9 days?	Wed 4/1/20	Mon 4/13/20		
73		Revise CAD Beta Prototy	33 days	Thu 2/27/20	Mon 4/13/20		All Members
74		Write Sensor Code	33 days	Thu 2/27/20	Mon 4/13/20		Marcin,Sean
75		Create Electrical Scehmatic Diagrams	33 days	Thu 2/27/20	Mon 4/13/20		Marcin
76		Create theoretical plan for testing	33 days	Thu 2/27/20	Mon 4/13/20		All Members
77		Test Beta Prototype Components	31 days	Thu 3/26/20	Thu 5/7/20		All Members
78		Plan for Online Innovation Expo	8 days	Thu 4/2/20	Mon 4/13/20		
79		Submit Final CAD Files	1 day	Thu 4/9/20	Thu 4/9/20		Omar
80		Phase 6	15 days?	Mon 4/13/20	Fri 5/1/20		
81		**Final Revision Phase**	**19 days?**	**Mon 4/13/20**	**Thu 5/7/20**		**All Members**
82		Completion of the CAD model and Evasion Mechanism	19 days	Mon 4/13/20	Thu 5/7/20		Noah,Omar
83		Completion of the Electrical Schematics and Data	19 days	Mon 4/13/20	Thu 5/7/20		Marcin,Noah,Se
84		Final Touch-ups	4 days	Mon 5/4/20	Thu 5/7/20		All Members
85		Phase 6 Presentation	19 days	Mon 4/13/20	Thu 5/7/20		All Members
86		Phase 6 Report	22 days	Mon 4/13/20	Tue 5/12/20		All Members
87		Innovation Expo	1 day	Fri 5/1/20	Fri 5/1/20		All Members

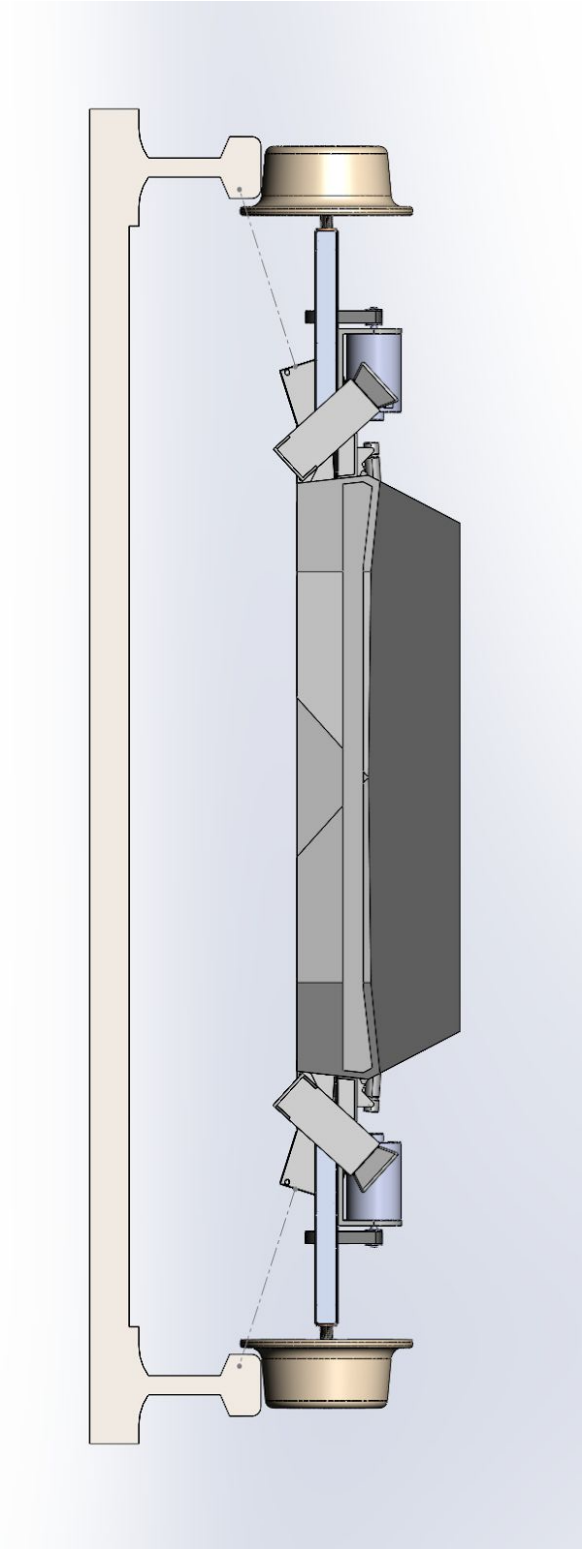
A3- Beta Isometric View



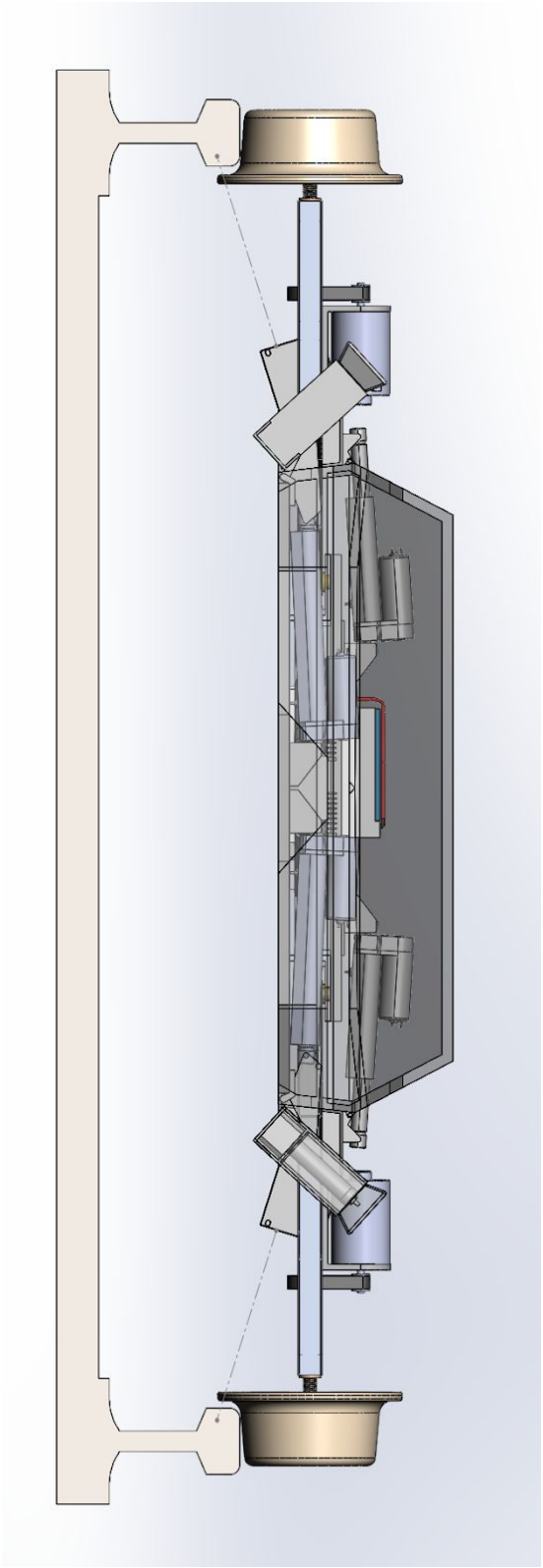
A4- Beta Isometric View, Transparent



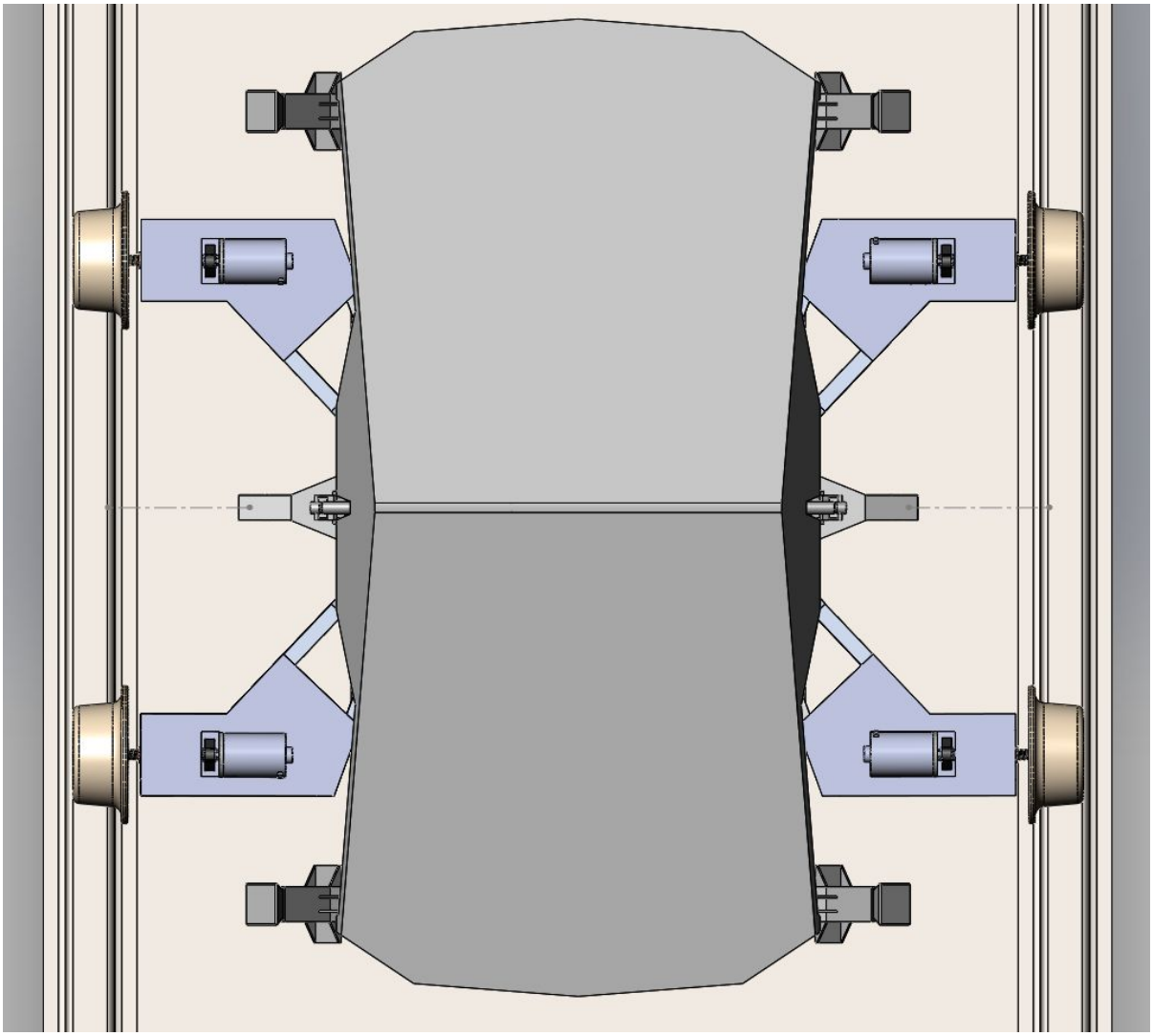
A5- Beta Front View



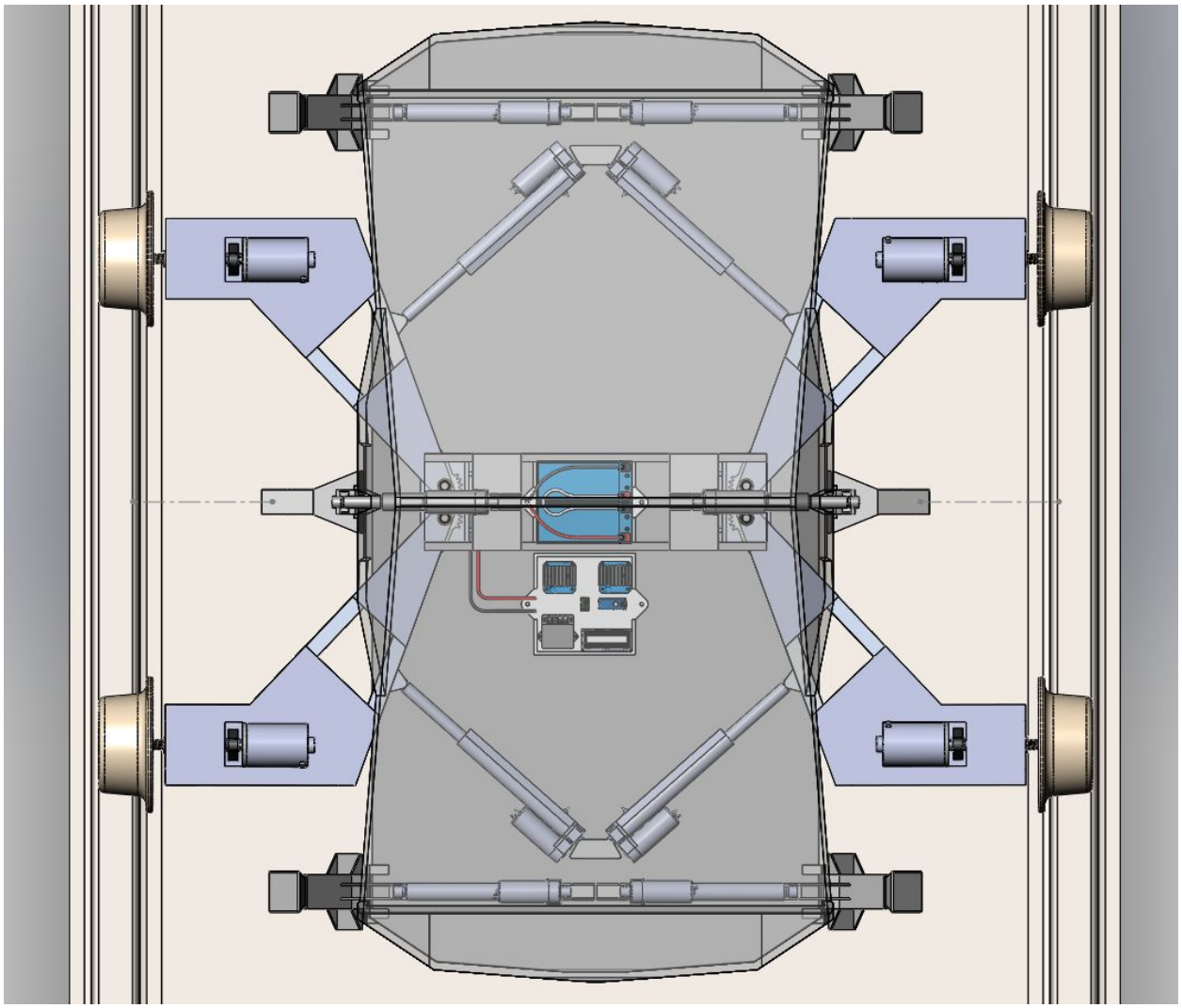
A6- Beta Front View, Transparent



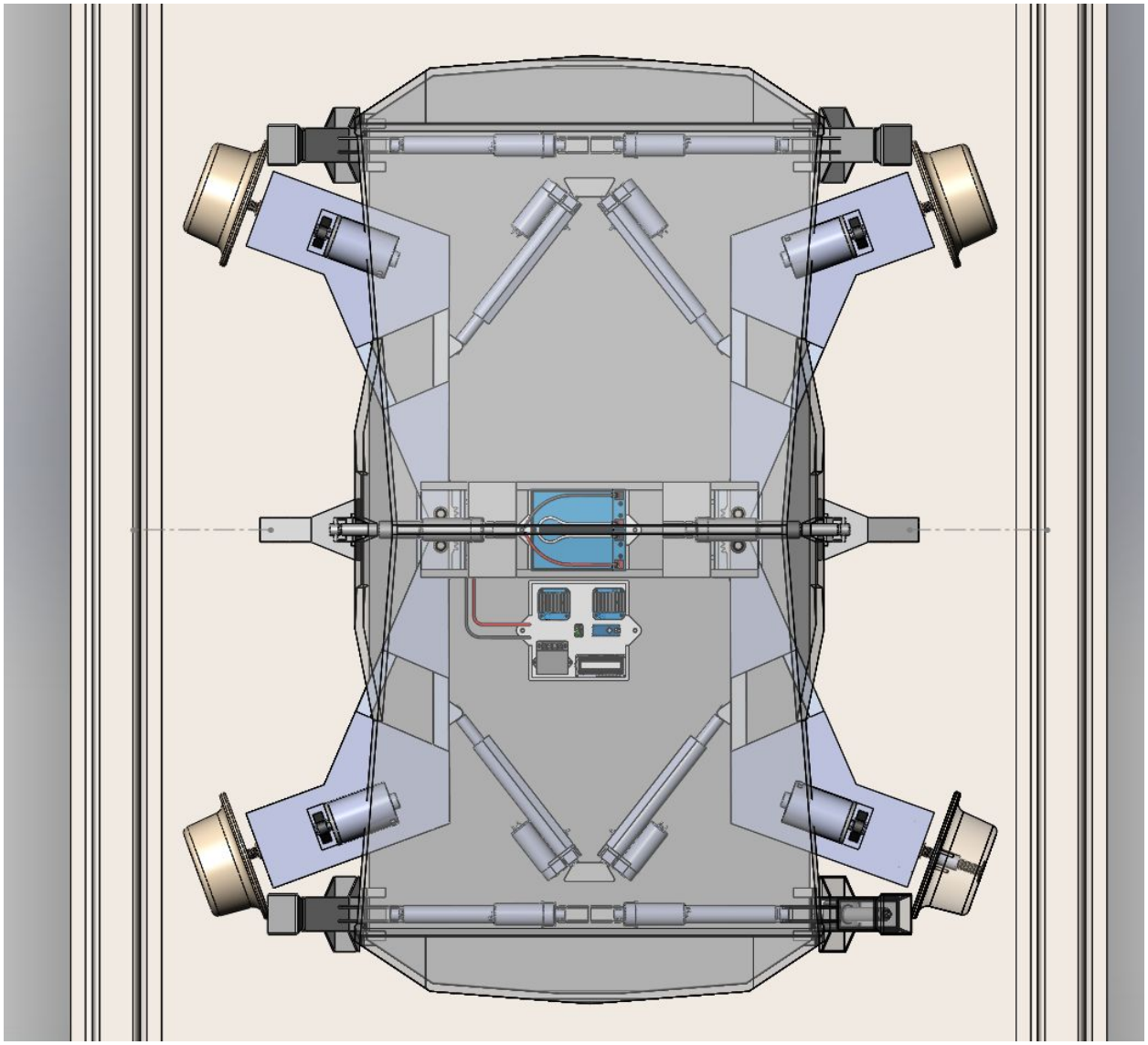
A7- Beta Top View



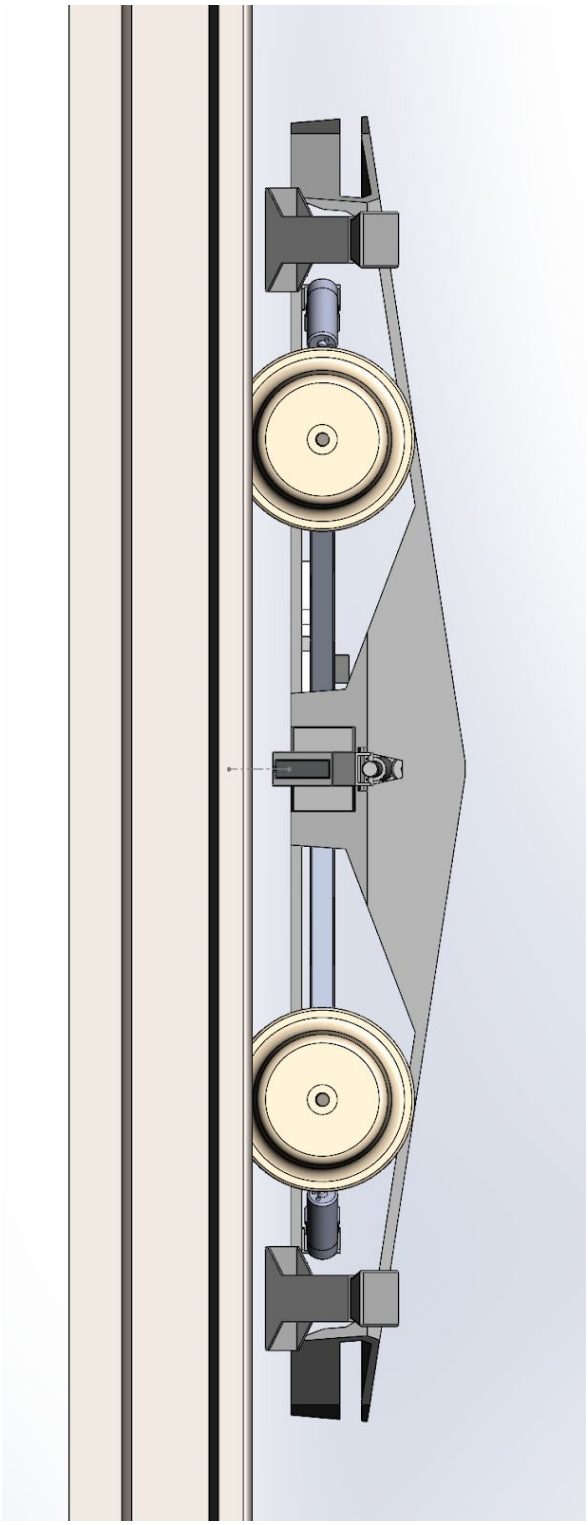
A8- Beta Top View, Transparent



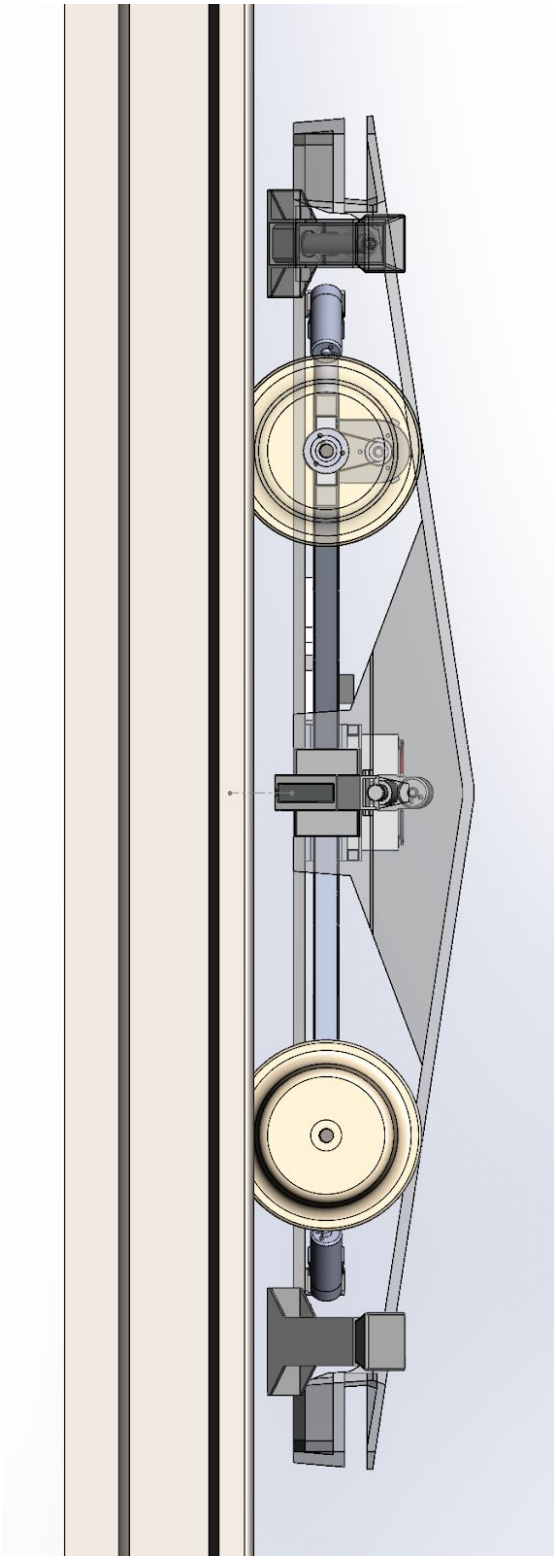
A9- Beta Top View, Transparent, Retracted



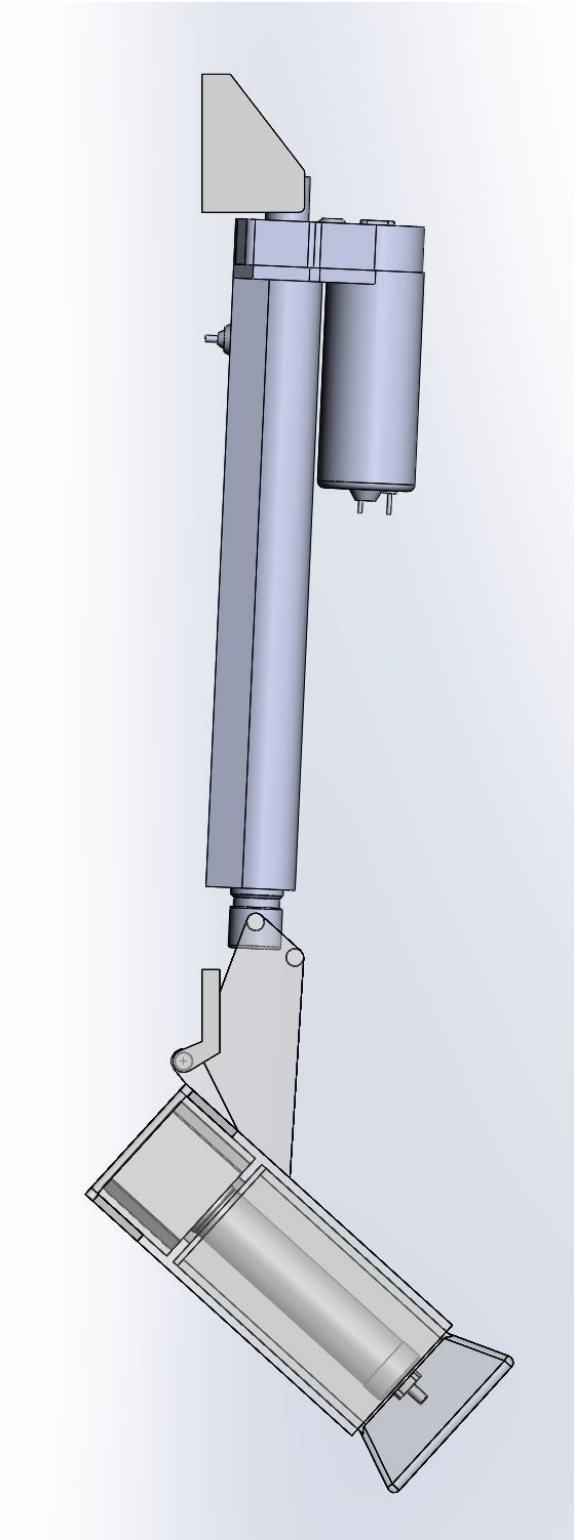
A10- Beta Side View



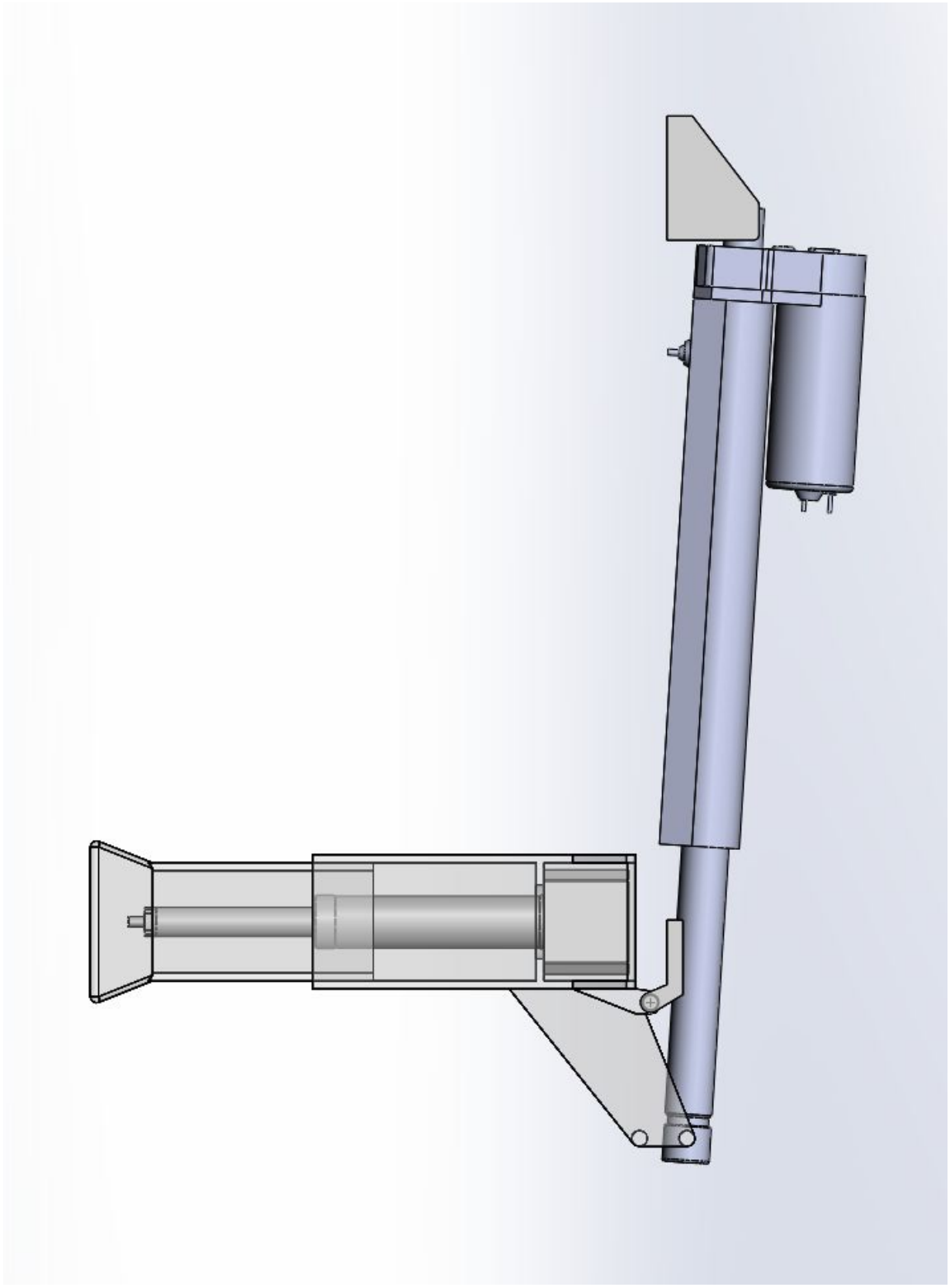
A11- Beta Side View, Transparent



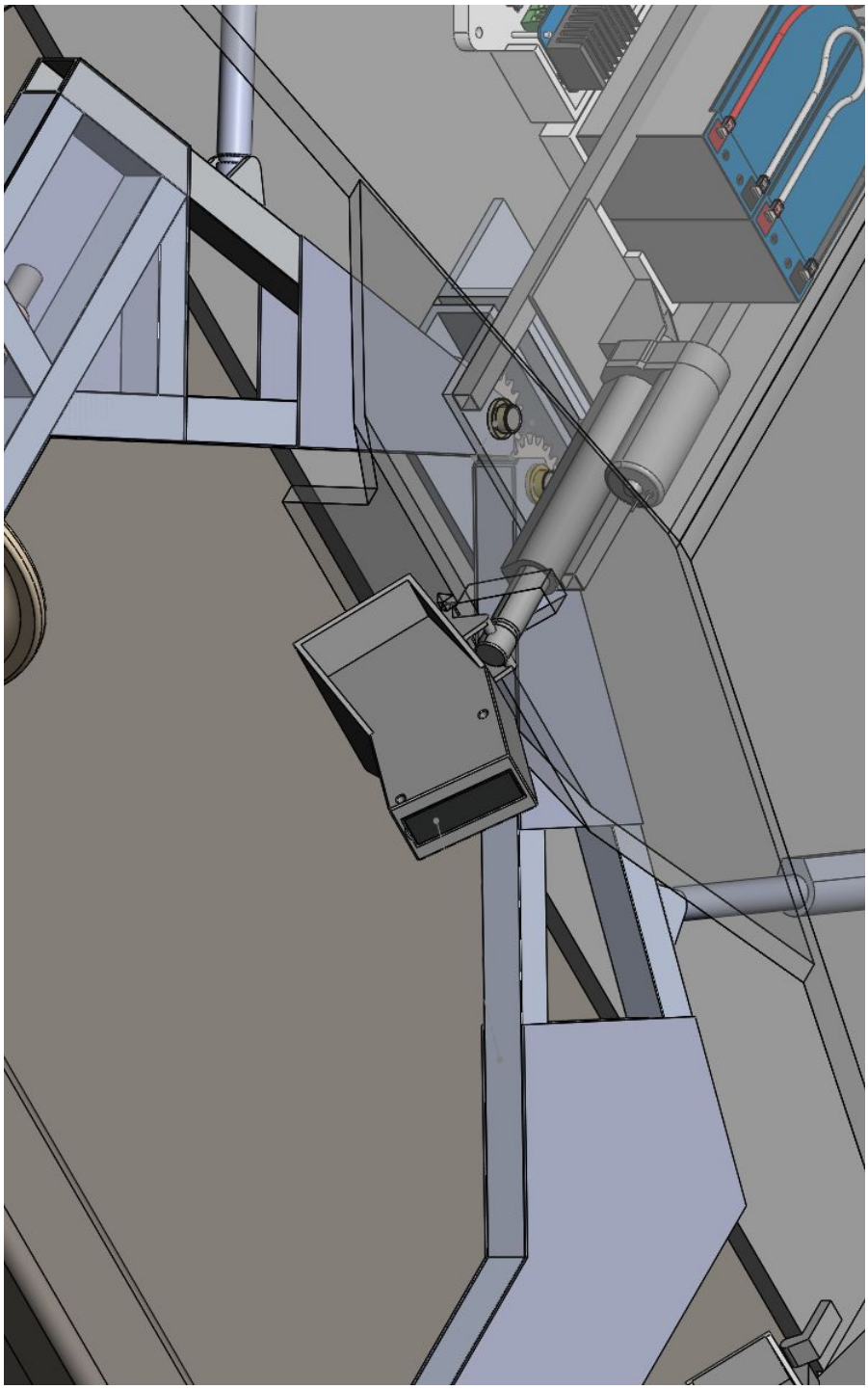
A12- Landing Legs, Retracted



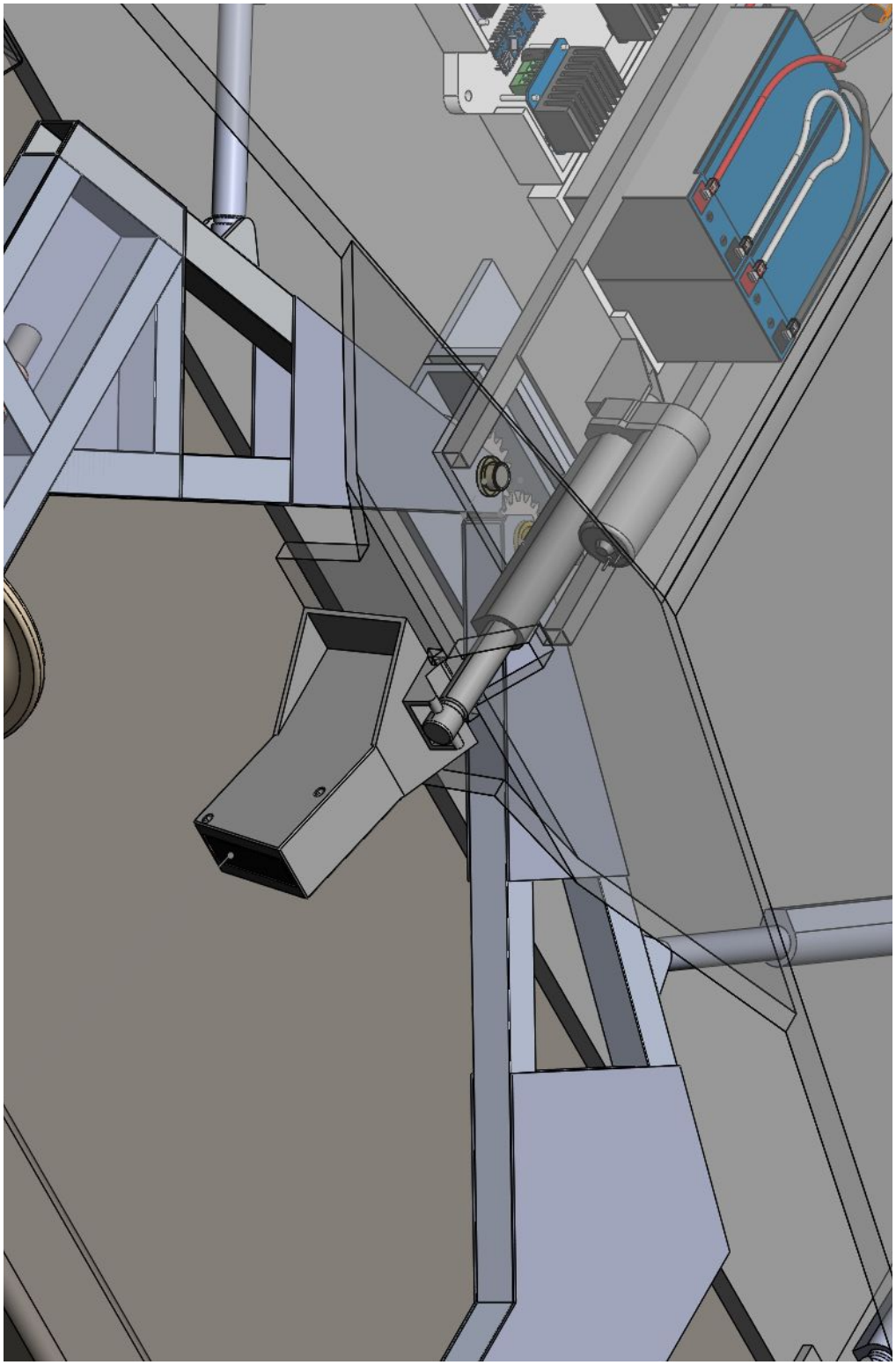
A13- Landing Legs, Extended



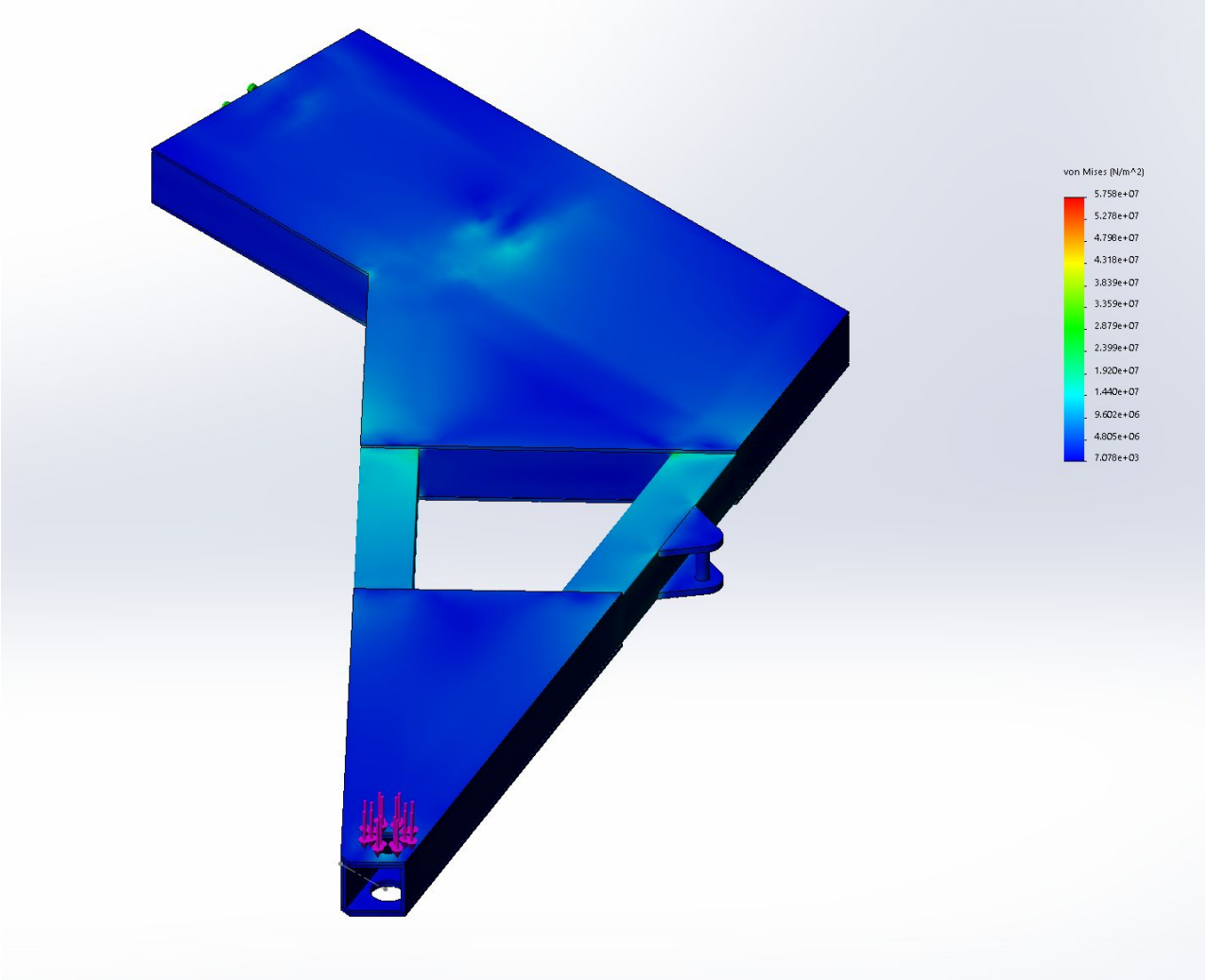
A14- Sensor Mount, Retracted



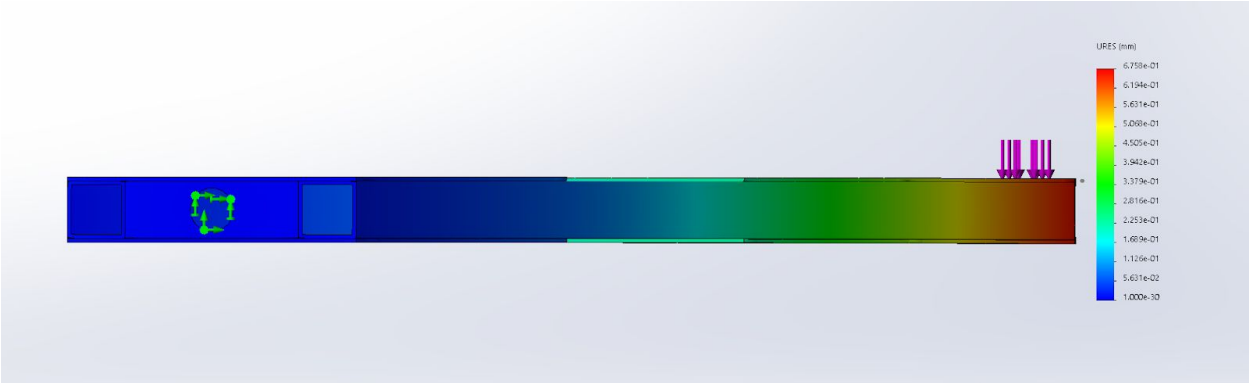
A15- Sensor Mount, Extended

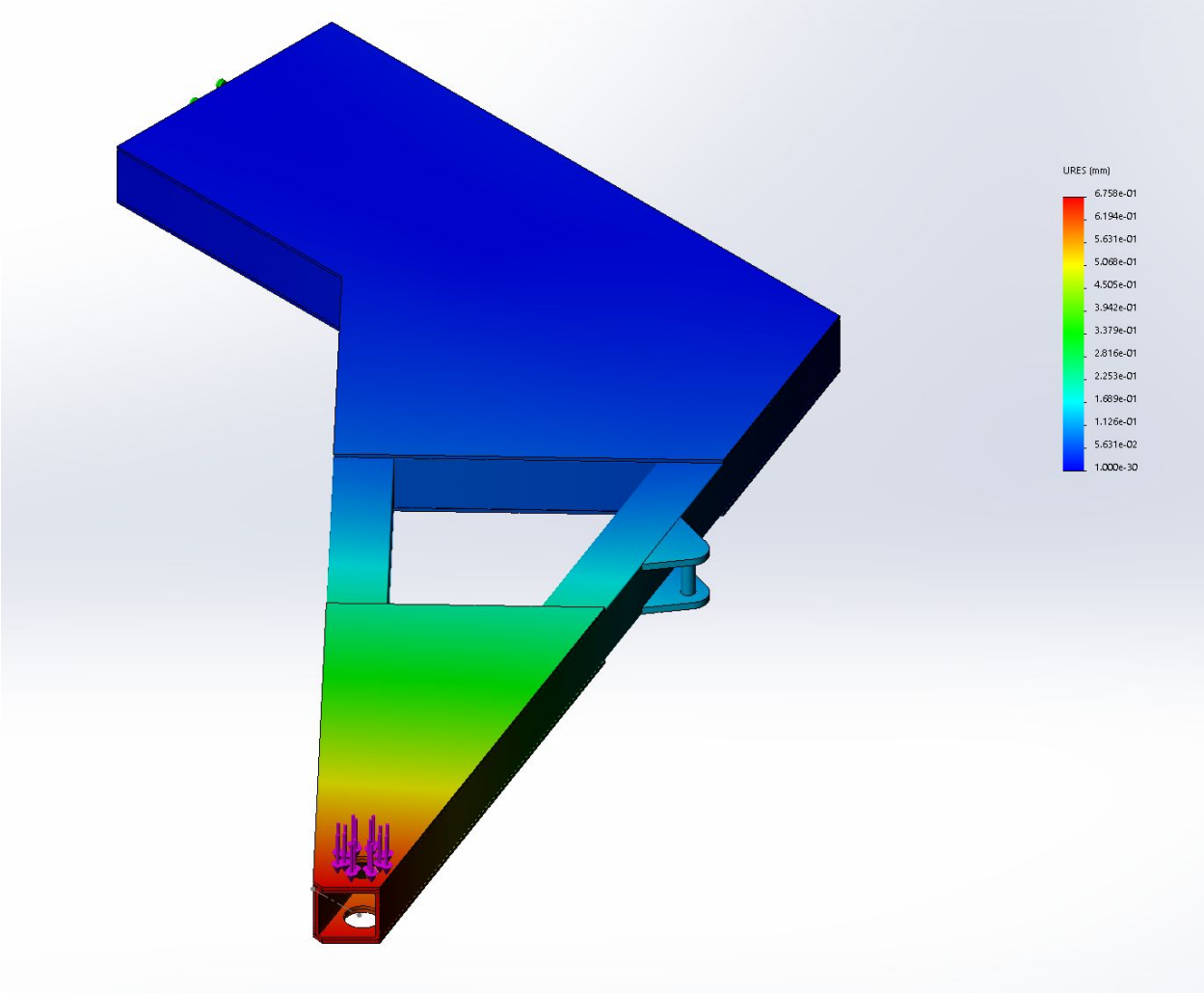


A16- Wheel-Base Stress Analysis



A17- Wheel-Base Displacement Analysis





A18- Bill of Materials

ITEM NO.	PART NUMBER	QTY.
1	cover	1
2	base	1
3	WHEEL BASE ASSEMBLY	1
	Sleeve bearing	4
	WHEEL BASE FRONT	1
	h_1	1
	h_2	1
	v_1	1
	d_1	1
	d_2	1
	d_3	1
	v_1_1	1
	gear_f	1
	p_1_t	1
	p_2_t	1
	p_1_b	1
	p_2_b	1
	WHEEL BASE REAR	1
	h_1-r	1
	h_2-r	1
	v_1-r	1
	d_1-r	1

	d_2-r	1
	d_3-r	1
	v_1_1-r	1
	gear_r	1
	p_1_b	1
	p_1_t	1
	p_2_b	1
	p_2_t	1
	bushing L plate top left	1
	bushing L plate bottom left	1
	pin	2
	bushing L plate mid left	1
	PRT_Axle_v2	2
	ASM_WheelAssembly_v2	2
	PRT_Wheel_v2	1
	PRT_Wheel_Hub_v2	1
	Flat countersunk head tapping screw_ai	3
	0.625x0.75x0.625 shaft sleeve bearing	8
	inner shaft bearing support	4
	Motor assem	2
	Motor body	1
	Pulley and shaft	1
	mount	1
	belt	2

4	WHEEL BASE ASSEMBLY RIGHT	1
	Sleeve bearing	4
	WHEEL BASE FRONT	1
	h_1	1
	h_2	1
	v_1	1
	d_1	1
	d_2	1
	d_3	1
	v_1_1	1
	gear_f	1
	p_1_t	1
	p_2_t	1
	p_1_b	1
	p_2_b	1
	WHEEL BASE REAR	1
	h_1-r	1
	h_2-r	1
	v_1-r	1
	d_1-r	1
	d_2-r	1
	d_3-r	1
	v_1_1-r	1
	gear_r	1

	p_1_b	1
	p_1_t	1
	p_2_b	1
	p_2_t	1
	bushing L plate top left	1
	bushing L plate bottom left	1
	pin	2
	bushing L plate mid left	1
	PRT_Axle_v2	2
	ASM_WheelAssembly_v2	2
	PRT_Wheel_v2	1
	PRT_WheelHub_v2	1
	flatcountersunk head tapping screw_ai	3
	Motor assem right	1
	Motor body	1
	Pulley and shaft	1
	mount	1
	0.625x0.75x 0.625 shaft sleeve bearing	8
	inner shaft bearing support	4
	belt	2
	Motor assem	1
	Motor body	1
	Pulley and shaft	1
	mount	1

5	half inch tube	2
6	Battery Assembly	1
	12V7AH SLA Battery	2
	Cable Batt Batt	1
	Cable Batt Out L	1
	Cable Batt Out R	1
	Battery Mount	1
7	Control Assembly	1
	Control Box	1
	DCtoDC	1
	lcd	1
	Minimu-9-v5	1
	Arduino_nano	1
	BTS7960	2
8	S12_17A16_06_CAA	4
	_ELECTRAK_1_REAR_S12_A_2	1
	_S12_A_1motor_4	1
	_S12_06A262_1tube_6	1
	_S12_06A_1rod_8	1
9	Straight Rail	1
10	leg assem	4
	leg slot	1
	leg ext	1
	MLA actuator	1

	Nema17a_body	1
	housing	1
	rod	1
	leg mount r	1
	leg mount l	1
	Actuator bracket	1
	S12_17A16_06_CAA	1
	_ELECTRAK_1_REAR_S12_A_2	1
	_S12_A_1motor_4	1
	_S12_06A262_1tube_6	1
	_S12_06A_1rod_8	1
11	Dual bracket	2
12	Sensor mount assem	2
	Sensor arm	1
	optoNCDT-1402	1
13	3_in actuator	2
	_ELECTRAK_1_REAR_S24_A_2	1
	_S24_A_1motor_4	1
	_S24_03A185_1tube_6	1
	_S24_03A_1rod_8	1
14	Actuator bracket	2
15	Mounting plate	2
	TOTAL	193

A19- Arduino Control Circuit Code

```
//send info to robot from wireless controller
//hc12 radio communication
#include<SoftwareSerial.h>
SoftwareSerial hc12( , );
//input command pins
int measurePin = ;
int gaugePin = ;
int crosslevelPin = ;
int distancePin = ;
int movementPin = ;

int evadePin = ;
//input commands (on/off)
int measure = 0;//-1 -2
int gauge = 0;//-3 -4
int crosslevel = 0;//-5 -6
int distance = 0;//-7 -8

int evade = 0;//-11 -12
//movement states and info
int potValue;//converted on robot
int stoppedFlag = 0; //must start with pot in middle to turn flag on
//input indicators
int measureLED = ;
int gaugeLED = ;
int crosslevelLED = ;
int distanceLED = ;
int movementLEDback = ;
int movementLEDforward = ;
int evadeLED = ;

void setup() {
  // put your setup code here, to run once:
  hc12.begin(9600);
  pinMode(measurePin, INPUT) = ;
  pinMode(gaugePin, INPUT) = ;
  pinMode(crosslevelPin, INPUT) = ;
  pinMode(distancePin, INPUT) = ;
  pinMode(movementPin, INPUT) = ;
```

```
  //input indicators
  pinMode(measureLED, OUTPUT) = ;
  pinMode(gaugeLED, OUTPUT) = ;
  pinMode(crosslevelLED, OUTPUT) = ;
  pinMode(distanceLED, OUTPUT) = ;
  pinMode(movementLEDforward, OUTPUT) = ;
  pinMode(movementLEDback, OUTPUT) = ;
  }

void loop() {
  // put your main code here, to run repeatedly:
  if digitalRead(measurePin == HIGH && measure != 1) {
    digitalWrite(measureLED, HIGH)
    measure = 1;
    hc12.write(-1);
  }
  else if (measurePin == LOW && measure != 0) {
    digitalWrite(measureLED, LOW)
    measure = 0;
    hc12.write(-2);
  }

  if digitalRead(gaugePin == HIGH && gauge != 1) {
    digitalWrite(gaugeLED, HIGH)
    gauge = 1;
    hc12.write(-3);
  }
  else if (gaugePin == LOW && gauge != 0) {
    digitalWrite(gaugeLED, LOW)
    gauge = 0;
    hc12.write(-4);
  }

  if digitalRead(crosslevelPin == HIGH && crosslevel != 1) {
    digitalWrite(crosslevelLED, HIGH)
    crosslevel = 1;
    hc12.write(-5);
  }
  else if (crosslevelPin == LOW && crosslevel != 0) {
    digitalWrite(crosslevelLED, LOW)
    crosslevel = 0;
    hc12.write(-6);
```

```
  }

  if digitalRead(distancePin == HIGH && distance != 1) {
    digitalWrite(distanceLED, HIGH)
    distance = 1;
    hc12.write(-7);
  }
  else if (distancePin == LOW && distance != 0) {
    digitalWrite(distanceLED, LOW)
    distance = 0;
    hc12.write(-8);
  }

  if digitalRead(evadePin == HIGH && evade != 1) {
    digitalWrite(evadeLED, HIGH)
    evade = 1;
    hc12.write(-11);
  }
  else if (evadePin == LOW && evade != 0) {
    digitalWrite(evadeLED, LOW)
    evade = 0;
    hc12.write(-12);
  }

  potValue = analogRead(movementPin);
  if analogRead(movementPin >= 409 && movementPin <= 614) {
    digitalWrite(movementLEDback, LOW);
    digitalWrite(movementLEDforward, LOW);
    hc12.write(0);
    stoppedFlag = 1;
  }
  else if (potValue < 409 && stoppedFlag != 0) {
    digitalWrite(movementLEDforward, LOW);
    digitalWrite(movementLEDback, HIGH);
    hc12.write(potValue);
  }
  else if (potValue > 614 && stoppedFlag != 0) {
    digitalWrite(movementLEDforward, HIGH);
    digitalWrite(movementLEDback, LOW);
    hc12.write(potValue);
  }
}
```

A20- Arduino Motion Circuit Code

```
//receives commands from control circuit
//gives commands to measurement circuit
//controls motors, arms, and legs

//receiving commands (on/off)
#include <SoftwareSerial.h>
SoftwareSerial hc12( , );
int measure = 0;//-1 -2
int gauge = 0;//-3 -4
int crosslevel = 0;//-5 -6
int distance = 0;//-7 -8

int evade = 0;//-11 -12
int capture;

//controlling motors
int leftMotorsForward = ;
int leftMotorsBackward = ;
int rightMotorsForward = ;
int rightMotorsBackward = ;

//avoid collision
int frontSensorPin = ;
int backSensorPin = ;

//controlling measurement
int measurePin = ;
int gaugePin = ;
int crosslevelPin = ;
int distancePin = ;


//Retraction_routine() + extension routine()
//Arms with wheels (motor controller PWM inputs)
int arm_1and2_extend = ;
int arm_1and2_retract = ;
int arm_3and4_extend = ;
int arm_3and4_retract = ;
//put legs in position (motor controller PWM inputs)
int legExtend_1and2_extend = ;
```

```
int legExtend_1and2_retract = ;
int legExtend_3and4_extend = ;
int legExtend_3and4_retract = ;
//legs(stepper motor driver inputs)
int step_leg1 = ;
int dir_leg1 = ;
int step_leg2 = ;
int dir_leg2 = ;
int step_leg3 = ;
int dir_leg3 = ;
int step_leg4 = ;
int dir_leg4 = ;
//limit switches
int arm_1and2_in = ;
int arm_1and2_out = ;
int arm_3and4_in = ;
int arm_3and4_out = ;
int legExtend_1and2_in = ;
int legExtend_1and2_out = ;
int legExtend_3and4_in = ;
int legExtend_3and4_out = ;
int leg1_in = ;
int leg1_out = ;
int leg2_in = ;
int leg2_out = ;
int leg3_in = ;
int leg3_out = ;
int leg4_in = ;
int leg4_out = ;
//memory
int counts_to_dip;

void setup() {
  // put your setup code here, to run once:
  hc12.begin(9600);
  //controlling motors
  pinMode(leftMotorsForward, OUTPUT);
  pinMode(rightMotorsBackward, OUTPUT);

  //controlling measurement
  pinMode(measurePin, OUTPUT);
  pinMode(gaugePin, OUTPUT);
```

```
  pinMode(crosslevelPin, OUTPUT);
  pinMode(distancePin, OUTPUT);


  //Arms with wheels (motor controller PWM inputs)
  pinMode(arm_1and2_extend, OUTPUT);
  pinMode(arm_1and2_retract, OUTPUT);
  pinMode(arm_3and4_extend, OUTPUT);
  pinMode(arm_3and4_retract, OUTPUT);
  //put legs in position (motor controller PWM inputs)
  pinMode(legExtend_1and2_extend, OUTPUT);
  pinMode(legExtend_1and2_retract, OUTPUT);
  pinMode(legExtend_3and4_extend, OUTPUT);
  pinMode(legExtend_3and4_retract, OUTPUT);
  //legs(stepper motor driver inputs)
  pinMode(step_leg1, OUTPUT);
  pinMode(dir_leg1, OUTPUT);
  pinMode(step_leg2, OUTPUT);
  pinMode(dir_leg2, OUTPUT);
  pinMode(step_leg3, OUTPUT);
  pinMode(dir_leg3, OUTPUT);
  pinMode(step_leg4, OUTPUT);
  pinMode(dir_leg4, OUTPUT);
  //limit switches (active when switch breaks circuit and pullup makes
signal high)
  pinMode(arm_1and2_in, INPUT_PULLUP);
  pinMode(arm_1and2_out, INPUT_PULLUP);
  pinMode(arm_3and4_in, INPUT_PULLUP);
  pinMode(arm_3and4_out, INPUT_PULLUP);
  pinMode(legExtend_1and2_in, INPUT_PULLUP);
  pinMode(legExtend_1and2_out, INPUT_PULLUP);
  pinMode(legExtend_3and4_in, INPUT_PULLUP);
  pinMode(legExtend_3and4_out, INPUT_PULLUP);
  pinMode(leg1_in, INPUT_PULLUP);
  pinMode(leg1_out, INPUT_PULLUP);
  pinMode(leg2_in, INPUT_PULLUP);
  pinMode(leg2_out, INPUT_PULLUP);
  pinMode(leg3_in, INPUT_PULLUP);
  pinMode(leg3_out, INPUT_PULLUP);
  pinMode(leg4_in, INPUT_PULLUP);
  pinMode(leg4_out, INPUT_PULLUP);
```

```
  //for safety
  analogWrite(leftMotorsForward, 0);
  analogWrite(rightMotorsForward, 0);
  analogWrite(leftMotorsBackward, 0);
  analogWrite(rightMotorsBackward, 0);

  pinMode(frontSensorPin, INPUT);
  pinMode(backSensorPin, INPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  if (hc12.available())(
      capture = hc12.read();
    if (capture == -1) {
    measure = 1;
    digitalWrite(measurePin, HIGH);
    }
    else if (capture == -2) {
    measure = 0;
    digitalWrite(measurePin, LOW);
    }
    else if (capture == -3) {
    gauge = 1;
    digitalWrite(gaugePin, HIGH);
    }
    else if (capture == -4) {
    gauge = 0;
    digitalWrite(gaugePin, LOW);
    }
    else if (capture == -5) {
    crosslevel = 1;
    digitalWrite(crosslevelPin, HIGH);
    }
    else if (capture == -6) {
    crosslevel = 0;
    digitalWrite(crosslevelPin, LOW);
    }
    else if (capture == -7) {
    distance = 1;
    digitalWrite(distancePin, HIGH);
    }
```

```
    else if (capture == -8) {
    distance = 0;
    digitalWrite(distancePin, LOW);
    }
    else if (capture == -11) {
    evade = 1;
    retraction_routine();
      while (evade == 1) {
        int temp = hc12.read();
        if (temp == -12) {
          evade = 0;
        }
      }
      extension_routine();
    }
    else if (capture > 0 && capture < 409 && analogRead(sensorPinfront) >
threshold) {
    analogWrite(leftMotorsForward, 0);
      analogWrite(rightMotorsForward, 0);
      analogWrite(leftMotorsBackward, map(capture, 409, 0, 0, 255);
                  analogWrite(rightMotorsBackward, map(capture, 409, 0, 0,
255);
    }
    else if (capture > 614 && capture <= 1023 && analogRead(sensorPinfront)
> threshold) {
    analogWrite(leftMotorsBackward, 0);
      analogWrite(rightMotorsBackward, 0);
      analogWrite(leftMotorsForward, map(capture, 614, 1023, 0, 255);
                  analogWrite(rightMotorsForward, map(capture, 614, 1023,
0, 255);
    }
    else if (capture == 0) {
    analogWrite(leftMotorsForward, 0);
      analogWrite(rightMotorsForward, 0);
      analogWrite(leftMotorsBackward, 0);
      analogWrite(rightMotorsBackward, 0);
    }
}


void retraction_routine() {
  //gain footing
```

```
digitalWrite(dir_leg1, HIGH);
digitalWrite(dir_leg2, HIGH);
digitalWrite(dir_leg3, HIGH);
digitalWrite(dir_leg4, HIGH);
while (leg1_in == 0 || leg2_in == 0 || leg3_in == 0 || leg4_in == 0) {
  digitalWrite(step_leg1, HIGH);
  digitalWrite(step_leg2, HIGH);
  digitalWrite(step_leg3, HIGH);
  digitalWrite(step_leg4, HIGH);
  delayMicroseconds(500);
  digitalWrite(step_leg1, LOW);
  digitalWrite(step_leg2, LOW);
  digitalWrite(step_leg3, LOW);
  digitalWrite(step_leg4, LOW);
  delayMicroseconds(500);
}
//lift slightly (pick iterations based on motor step size)
for (int x = 0; x = 100; x++) {
  digitalWrite(step_leg1, HIGH);
  digitalWrite(step_leg2, HIGH);
  digitalWrite(step_leg3, HIGH);
  digitalWrite(step_leg4, HIGH);
  delayMicroseconds(500);
  digitalWrite(step_leg1, LOW);
  digitalWrite(step_leg2, LOW);
  digitalWrite(step_leg3, LOW);
  digitalWrite(step_leg4, LOW);
  delayMicroseconds(500);
}
//retract arms
analogWrite(arm_1and2_retract, 255);
analogWrite(arm_3and4_retract, 255);
while (arm_1and2_in == 0 || arm_3and4_in == 0) {
}
analogWrite(arm_1and2_retract, 0);
analogWrite(arm_3and4_retract, 0);
//lower chassis
digitalWrite(dir_leg1, LOW);
digitalWrite(dir_leg2, LOW);
digitalWrite(dir_leg3, LOW);
digitalWrite(dir_leg4, LOW);
while (leg1_in == 0 && leg2_in == 0 && leg3_in == 0 && leg4_in == 0 ) {
```

```
    digitalWrite(step_leg1, HIGH);
    digitalWrite(step_leg2, HIGH);
    digitalWrite(step_leg3, HIGH);
    digitalWrite(step_leg4, HIGH);
    delayMicroseconds(500);
    digitalWrite(step_leg1, LOW);
    digitalWrite(step_leg2, LOW);
    digitalWrite(step_leg3, LOW);
    digitalWrite(step_leg4, LOW);
    delayMicroseconds(500);
    counts_to_dip++;
  }
}

void extension_routine() {
  //rise back up based on lowering memory
  digitalWrite(dir_leg1, HIGH);
  digitalWrite(dir_leg2, HIGH);
  digitalWrite(dir_leg3, HIGH);
  digitalWrite(dir_leg4, HIGH);
  for (int x = 0; x = counts_to_dip; x++) {
    digitalWrite(step_leg1, HIGH);
    digitalWrite(step_leg2, HIGH);
    digitalWrite(step_leg3, HIGH);
    digitalWrite(step_leg4, HIGH);
    delayMicroseconds(500);
    digitalWrite(step_leg1, LOW);
    digitalWrite(step_leg2, LOW);
    digitalWrite(step_leg3, LOW);
    digitalWrite(step_leg4, LOW);
    delayMicroseconds(500);
  }
  //extend arms
  analogWrite(arm_1and2_extend, 255);
  analogWrite(arm_3and4_extend, 255);
  while (arm_1and2_out == 0 || arm_3and4_out == 0) {
  }
  analogWrite(arm_1and2_extend, 0);
  analogWrite(arm_3and4_extend, 0);
  //return to rails
  digitalWrite(dir_leg1, LOW);
  digitalWrite(dir_leg2, LOW);
```

```
digitalWrite(dir_leg3, LOW);
digitalWrite(dir_leg4, LOW);
while (leg1_in == 0 || leg2_in == 0 || leg3_in == 0 || leg4_in == 0 ) {
  digitalWrite(step_leg1, HIGH);
  digitalWrite(step_leg2, HIGH);
  digitalWrite(step_leg3, HIGH);
  digitalWrite(step_leg4, HIGH);
  delayMicroseconds(500);
  digitalWrite(step_leg1, LOW);
  digitalWrite(step_leg2, LOW);
  digitalWrite(step_leg3, LOW);
  digitalWrite(step_leg4, LOW);
  delayMicroseconds(500);
}
```

A21- Arduino Measurement Circuit Code

```
//measurement circuit
//take commands from motion circuit
//interface with encoder microcontroller

//commands
int measure = 0;//-1 -2
int gauge = 0;//-3 -4
int crosslevel = 0;//-5 -6
int distance = 0;//-7 -8

//command origin
int measurePin = ;
int gaugePin = ;
int crosslevelPin = ;
int distancePin = ;

//command received indicators
int measureLED = ;
int gaugeLED = ;
int crosslevelLED = ;
int distanceLED = ;

//setting up and reading from sensors

//distance
int dist1Pin = 0;
int dist1;
int dist2Pin = 0;
int dist2;

//IMU
#include <Wire.h>
#include<LSM6.h>

int roll = 0; //for SD use
//Declaring some global variables
int gyro_x, gyro_y, gyro_z;
long acc_x, acc_y, acc_z, acc_total_vector;
long gyro_x_cal, gyro_y_cal, gyro_z_cal;
long loop_timer;
```

```
float angle_pitch, angle_roll;
boolean set_gyro_angles;
float angle_roll_acc, angle_pitch_acc;
float angle_pitch_output, angle_roll_output;

int time_elapsed;
int new_time = 0;
int prev_time = 0;

//odometer
int distanceValue = 0;


//RTC + SD
//Outside loop setup for RTC
#include "RTClib.h"
RTC_PCF8523 rtc;

//outside loop setup for sd
#include <SD.h>
const int chipSelect = 10; //for comm with new Adafruit data logger
File distDataFile; //create file class to allow for repeated opening
char filename[] = "dist00.CSV"; //first name to try

int count = 0;

void setup() {
  // put your setup code here, to run once:

  //receiving commands
  pinMode(measurePin, OUTPUT);
  pinMode(gaugePin, OUTPUT);
  pinMode(crosslevelPin, OUTPUT);
  pinMode(distancePin, OUTPUT);

  //command received indicators
  pinMode(measureLED, OUTPUT);
  pinMode(gaugeLED, OUTPUT);
  pinMode(crosslevelLED, OUTPUT);
  pinMode(distanceLED, OUTPUT);
```

```
  //setting up sensors
  Serial.begin(9600);//for odometry
  //distance
  pinMode(dist1Pin, INPUT);
  pinMode(dist2Pin, INPUT);

  //IMU
  Wire.begin();
//Start I2C as master
  imu.init();
  imu.enableDefault();

  //find average offsets from 2000 runs
  for (int cal_int = 0; cal_int < 2000 ; cal_int ++) {
//Run this code 2000 times
    imu.read();                                                //Read the raw
acc and gyro data from the IMU
    gyro_x_cal += imu.g.x;
//Add the gyro x-axis offset to the gyro_x_cal variable
    gyro_y_cal += imu.g.y;
//Add the gyro y-axis offset to the gyro_y_cal variable
    gyro_z_cal += imu.g.z;
//Add the gyro z-axis offset to the gyro_z_cal variable
    delay(3);
//Delay 3us to simulate the 250Hz program loop
  }
  gyro_x_cal /= 2000;
//Divide the gyro_x_cal variable by 2000 to get the average offset
  gyro_y_cal /= 2000;
//Divide the gyro_y_cal variable by 2000 to get the average offset
  gyro_z_cal /= 2000;
//Divide the gyro_z_cal variable by 2000 to get the average offset
  prev_time = micros();


  //start RTC
  rtc.begin();
  /*if(!rtc.initialized()){
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));//give it compiler time
(time of computer)
    }*/
```

```
  //initializing the SD
  pinMode(chipSelect, OUTPUT);
  SD.begin(chipSelect);

  //create a new file(and change name if previous exists)
  for (int i = 0; i < 100; i++) { //keep searching until name not taken
    filename[4] = i / 10 + '0'; //Are '0's necessary ?
    filename[5] = i % 10 + '0';
    if (!SD.exists(filename)) {
      distDataFile = SD.open(filename, FILE_WRITE);
      distDataFile.close();
      Serial.print("Current File: ");
      Serial.println(filename);
      break;
    }
  }
//Make titles
distDataFile = SD.open(filename, FILE_WRITE);
//TimeStamp
distDataFile.print("Time");
//RunTime
distDataFile.print(",");
distDataFile.print("Run Time (s)");
//Count
distDataFile.print(',');
distDataFile.println("Count");
//Distances
distDataFile.print(',');
distDataFile.println("dist1");
distDataFile.print(',');
distDataFile.print("dist2");
//Crosslevel roll
distDataFile.print(',');
distDataFile.print("roll");
//Distance odometry
distDataFile.print(',');
distDataFile.println("distanceValue");
distDataFile.close();
}

void loop() {
  // put your main code here, to run repeatedly:
```

```
  //read from and update IMU
  new_time = micros();
  imu.read(); //Read the raw acc and gyro data from the MPU-6050
  gyro_x = imu.g.x;
  gyro_y = imu.g.y;
  gyro_z = imu.g.z;

  gyro_x -= gyro_x_cal;
//Subtract the offset calibration value from the raw gyro_x value
  gyro_y -= gyro_y_cal;
//Subtract the offset calibration value from the raw gyro_y value
  gyro_z -= gyro_z_cal;
//Subtract the offset calibration value from the raw gyro_z value

  //Gyro angle calculations (relative)
  time_elapsed = new_time - prev_time;
  angle_pitch += (gyro_x * .00875) * (time_elapsed / 1000000);
//Calculate the traveled pitch angle and add this to the angle_pitch
variable
  angle_roll += (gyro_y * 0.00875) * (time_elapsed / 1000000);
//Calculate the traveled roll angle and add this to the angle_roll variable
  prev_time = new_time;

  //absolute pitch and roll
  //0.000001066 = 0.0000611 * (3.142(PI) / 180degr) The Arduino sin
function is in radians
  angle_pitch += angle_roll * sin((gyro_z * 0.00875) * (time_elapsed /
1000000));               //If the IMU has yawed transfer the roll angle to
the pitch angle
  angle_roll -= angle_pitch * sin((gyro_z * 0.00875) * (time_elapsed /
1000000));               //If the IMU has yawed transfer the pitch angle to
the roll angle

  //Accelerometer angle calculations
  acc_total_vector = sqrt((imu.a.x * imu.a.x) + (imu.a.y * imu.a.y) +
(imu.a.z * imu.a.z)); //Calculate the total accelerometer vector
  //57.296 = 1 / (3.142 / 180) The Arduino asin function is in radians
  angle_pitch_acc = asin((float)imu.a.y / acc_total_vector) * 57.296;
//Calculate the pitch angle
  angle_roll_acc = asin((float)imu.a.x / acc_total_vector) * -57.296;
//Calculate the roll angle
```

```
  //need to CALIBRATE with custom IMU
  //Place the IMU level and note the values in the following two lines for
calibration
  angle_pitch_acc -= 0.0;
//Accelerometer calibration value for pitch
  angle_roll_acc -= 0.0;
//Accelerometer calibration value for roll

  if (set_gyro_angles) {                                                //If
the IMU is already started
    angle_pitch = angle_pitch * 0.9996 + angle_pitch_acc * 0.0004;
//Correct the drift of the gyro pitch angle with the accelerometer pitch
angle
    angle_roll = angle_roll * 0.9996 + angle_roll_acc * 0.0004;
//Correct the drift of the gyro roll angle with the accelerometer roll
angle
  }
  else {                                                                //At
first start
    angle_pitch = angle_pitch_acc;
//Set the gyro pitch angle equal to the accelerometer pitch angle
    angle_roll = angle_roll_acc;
//Set the gyro roll angle equal to the accelerometer roll angle
    set_gyro_angles = true;
//Set the IMU started flag
  }

  //To dampen the pitch and roll angles a complementary filter is used
  angle_pitch_output = angle_pitch_output * 0.9 + angle_pitch * 0.1;
//Take 90% of the output pitch value and add 10% of the raw pitch value
  angle_roll_output = angle_roll_output * 0.9 + angle_roll * 0.1;
//Take 90% of the output roll value and add 10% of the raw roll value


  //check status of commands
  if (measurePin == HIGH) {
    measure == 1;
  }
  else {
    measure == 0;
  }
```

```
if (gaugePin == HIGH) {
  gauge == 1;
}
else {
  gauge == 0;
}

if (crosslevelPin == HIGH) {
  crosslevel == 1;
}
else {
  crosslevel == 0;
}

if (distancePin == HIGH) {
  distance == 1;
}
else {
  distance == 0;
}

//do asked operations
if (measure == 1) {
  if (gauge == 1) {
    dist1 = analogRead(dist1Pin);
    dist2 = analogRead(dist2Pin);
  }
  if (crosslevel == 1) {
    roll = angle_roll_output;
  }
  if (distance == 1) {
    Serial.write('d');
    while (!Serial.available()) {
      //wait
    }
    distanceValue = Serial.read();
  }
  DateTime now = rtc.now(); //process for capturing readable timestamp

  distDataFile = SD.open(filename, FILE_WRITE);
  //TimeStamp
  distDataFile.print(now.year());
```

```
    distDataFile.print('/');
    distDataFile.print(now.month());
    distDataFile.print('/');
    distDataFile.print(now.day());
    distDataFile.print(' ');
    distDataFile.print(now.hour());
    distDataFile.print(':');
    distDataFile.print(now.minute());
    distDataFile.print(':');
    distDataFile.print(now.second());
    //RunTime
    distDataFile.print(",");
    distDataFile.print(millis() / 1000);
    //Count
    distDataFile.print(',');
    distDataFile.println(count);
    //Distances
    distDataFile.print(',');
    distDataFile.println(dist1);
    distDataFile.print(',');
    distDataFile.print(dist2);
    //Crosslevel roll
    distDataFile.print(',');
    distDataFile.print(roll);
    //Distance odometry
    distDataFile.print(',');
    distDataFile.println(distanceValue);
    distDataFile.close();
    count = count + 1;
  }

}
```

A22- Arduino Encoder Circuit Code

```
//increment distance, record direction, indicate direction, give distance
when asked
int pinA = 2;
int pinB = 3;
volatile long distance = 0;
volatile boolean direct = 1;
int indicateDirect = ; //choose pin

void setup() { // put your setup code here, to run once:
  pinMode(pinA, INPUT);
  pinMode(pinB, INPUT);
  attachInterrupt(digitalPinToInterrupt(pinA), increment, RISING);
  pinMode(indicateDirect, OUTPUT);
  serial.begin(9600);
}

void loop() { // put your main code here, to run repeatedly:
  if (serial.available()) {
    Serial.write(distance)
  }
  if (direct == 1 && prev_direct != 1) {
    digitalWrite(indicateDirect, HIGH);
    prev_direct = 1;
  }
  else if (direct == 0 && prev_direct != 0) {
    digitalWrite(indicateDirect, LOW);
    prev_direct = 0;
  }
}

void increment() {
  if (digitalRead(pinB) == HIGH) {
    distance += 1;
    direct = 1;
  }
else {
    distance -= 1;
    direct = 0;
  }
}
```