



Preliminary Design Document

Tia McKenzie
Nicodemus Phaklides
Lachlan McManus
Jacob Woodruff
Emmanuel Jefferson
Alexander Hoppe

ME 407
Robotics Preliminary Design
Embry-Riddle Aeronautical University

Table of Contents

1.0 Introduction.....	5
2.0 Requirements.....	5
2.1 Definitions	5
3.0 Conceptual Design	6
3.1 Mechanical Concept.....	6
3.2 Electrical Conceptual.....	7
3.3 Software Concept.....	8
4.0 System Specifications.....	10
4.1 Measuring Soil Water Saturation.....	10
4.2 Measuring Soil Temperature	10
4.3 Application Efficiency.....	11
4.4 Maintain Plant Health.....	11
4.5 Exterminating Weeds	11
4.6 Plant Location Determination.....	12
4.7 IP55 Ingress Protection.....	13
4.8 Communication	13
5.0 Preliminary Design.....	13
5.1 Mechanical Design of the Robot	14
5.1.1 Robotic Linkages	17
5.1.2 Hydraulic System.....	18
5.1.3 Structural Analysis.....	18
5.2 Electrical Design of the Robot	25
5.2.1 Power Distribution	25
5.2.2 Mechatronic Interfacing	27
5.2.3 High Voltage Elimination Circuit (HVEC)	31
5.2.4 Data Distribution	32
5.2.5 Central Tower Electronics	33
5.2.6 Hardware PCB.....	34
5.2.7 Trolley Electronics	36
5.2.8 Trolley PCB.....	37
5.2.9 Electrical Testing	40
5.3 Software Design of the Robot.....	43
5.3.1 Growing Zone Inspection	44
5.4 Kinematics of the Robot.....	48
5.4.1 Forward Kinematic Equations.....	48
5.4.2 Velocity Kinematic Equations	51
5.4.3 Inverse Kinematic Solutions	52
5.5 Equations of Motion of the Robot	57
5.5.1 Dynamical Simulations of the Open-Loop Robot	59
5.6 Control System for the Robot	59
5.6.1 Dynamical Simulations of the Closed-Loop Robot.....	60
5.6.2 Stability Analysis.....	62
6.0 Project Management Review.....	62
6.1 Budget	63
7.0 Conclusion	63
8.0 Citations	64
Appendix A – Bill of Materials	67

A.1 Custom Components	67
A.2 Printed Circuit Boards	68
A.3 Purchased Parts	68
Appendix B – Electrical Schematics	71
B.1 Arduino Hardware PCB Schematic	71
B.2 Trolley PCB Schematic.....	72
Appendix C – Source Code	73
C.1 Interfacing camera with Jetson Nano [Python].....	73
C.2 Training machine learning classification algorithm on dataset of physical textures [Python].....	74
C.3 Loading still image and applying pre-trained texture classification algorithm on detected contours [Python].....	75
C.4 TCN75AVOA713 Driver library [C++].....	78
Header: "temp_i2c.h"	78
Source: "temp_i2c.ccp".....	78
C.5 AD7995 Driver library [C++]	79
Header: "adc_i2c.h"	79
Source: "adc_i2c.ccp"	79
C.6 LS7184N library [C++]	80
Header: "quad_enc.h".....	80
Source: "quad_enc.ccp"	81
C.7 Sensor Testing File [C++]	81
Source: "NILE_Hardware.ino"	81
C.8 Dynamic Simulation and Controller [MATLAB]	84
Appendix D – Mechanical Drawings	89

Table of Figures

Figure 1. Full System Conceptual Design Render	6
Figure 2. Electrical Block Diagram.....	7
Figure 3. Data Block Diagram	8
Figure 4. Plant Care Flowchart.....	9
Figure 5. Computer Vision and Machine Learning Diagram.....	9
Figure 6. <i>Valerianella Locusta</i>	14
Figure 7. NILE Mechanical Design Overview	14
Figure 8. Gantry Assembly	15
Figure 9. Trolley Assembly.....	16
Figure 10. End Effector Assembly	17
Figure 11. Robotics Linkages.....	17
Figure 12. Hydraulic System Diagram.....	18
Figure 13. Gantry Beam Analysis.....	19
Figure 14. Gantry Beam Stress	19
Figure 15. Gantry Beam Displacement.....	20
Figure 16. Motor Mount Gravitational Analysis	20
Figure 17. Motor Mount Gravitational Stress and FOS	21
Figure 18. End-Effector Analysis	22
Figure 19. End-Effector Stress	23
Figure 20. Vertical Translational Beam Analysis.....	23

Figure 21. Vertical Translational Beam Stress and Displacements.....	24
Figure 22. Motor Mount Stress and FOS.....	25
Figure 23. Power Distribution Diagram	25
Figure 24. High-Voltage Elimination Circuit [26].....	31
Figure 25. Peripheral Data Flow Diagram	32
Figure 26. Central Electrical Diagram.....	33
Figure 27. NVIDIA Jetson Nano.....	33
Figure 28. Arduino Mega	34
Figure 29. PCB for Arduino and Peripheral Interface	34
Figure 30. Trolley Electrical Diagram	36
Figure 31. Trolley Electronics PCB	37
Figure 32. Perspective View of Trolley PCB.....	37
Figure 33. Electrical Test Set-up	40
Figure 34. Temperature Sensor Reading Test.....	41
Figure 35. Potentiometer with ADC Test Output	41
Figure 36. Moisture Sensor Response Test.....	42
Figure 37. Rotary Encoder Test Graph	42
Figure 38. HVEC eliminating an undesired plant.....	43
Figure 39. NVIDIA Jetson and Display	44
Figure 40. Growing Zone Inspection Flowchart.....	45
Figure 41. Original Image Versus Gaussian Blur	46
Figure 42. Hue-Saturation-Value Color Space Image.....	46
Figure 43. Color Masked Image.....	46
Figure 44. Image Processing Algorithm.....	47
Figure 45. Machine Learning Correctly Identifying Mache	48
Figure 46. Temperature Sensor Inverse Kinematics	53
Figure 47. Moisture Sensor Inverse Kinematics.....	54
Figure 48. Nozzle Inverse Kinematics.....	55
Figure 49. Camera Inverse Kinematics	56
Figure 50. HVEC Inverse Kinematics.....	57
Figure 51. Open-Loop Dynamic Response	59
Figure 52. Closed-Loop Dynamic Response	60
Figure 53. Stepper Motor Modeling.....	61
Figure 54. Simulating Simultaneous Joints	61
Figure 55. Motor Driver Inputs	62

1.0 Introduction

Conventional agricultural methods overcompensate plant and soil needs through wasteful watering practices and excessive application of pesticides and fertilizers, leading to substantial environmental damage. This damage takes on various forms, including pollution via runoff, soil depletion, and the extinction of local pollinators.

The Novel Irrigation and Land use Efficiency (NILE) projects aims to combat this issue by taking a unique, robotic approach to precisely care for, and monitor the health of, various crops. By making optimal use of available resources, the NILE system will reduce the environmental impact of agriculture while increasing yield.

2.0 Requirements

To achieve the goals set by the NILE project and ultimately address the problem statement, system requirements must be defined. The requirements serve as the foundation for everything that follows in the design process by describing what system must do.

1. The system shall be capable of measuring the water saturation for the soil at any point within the growing zone.
2. The system shall be capable of measuring the temperature of the growing zone.
3. The system shall be capable of irrigating crops within the growing zone with a field application efficiency greater than 90%.
4. The system shall be capable of supplying the nutrients needed to maintain plant health within the growing zone.
5. The system shall be capable of exterminating weeds from any point in the growing zone.
6. The system shall be capable of determining the location of plant foliage, plant stems, and parasites within the growing zone.
7. The system shall be capable of being certified to an ingress protection level of IP55. [15]
8. The system shall be capable of communicating plant health, soil health, and the detection of parasites to the end user.

2.1 Definitions

The requirements stated above refer to several terms that have been explicitly defined for the context of this project.

Soil health is a myriad of measurable parameters relating to the soil's ability to provide water and nutrients to the plants. For the scope of this project these parameters include soil moisture levels and soil temperature.

Plant health is a variety of conditions that either inhibit or assist in the growing and continued life of the plant. For the scope of this project these parameters include water consumption, nutrient needs, and physical integrity.

The growing zone is the three-dimensional soiled volume from the surface to a depth of 5 centimeters that can be reached by the end effector.

Weeds are any unwanted plants that directly, or indirectly, derive some or all their nutritional requirements from the growing zone.

Field application efficiency is an industry measure of how well an irrigation system performs when directed to deliver a specific amount of water.

$$\text{Field Application Efficiency} = \frac{\sum \text{Water Delivered to the Growing Zone}}{\sum \text{Water Input to the System}} \times 100\%$$

3.0 Conceptual Design

The NILE team conducted extensive research into existing products and literature in our quest to address our problem statement and meet the requirements. From backyard solutions such as the FarmBot [14] to industrial greenhouses such as AgBotic [7] there have been a variety of attempts to modernize farming techniques. In addition, NILE has found numerous academic examples of the integration of sensors and irrigation systems to assess plant health [8] [12].

Based on the research the team created numerous morphological charts to examine the processes and products available then used decision matrices to select the optimal concepts. Through this process it was determined that the NILE system would consist of a polar robot operated in a raised bed growing zone. All the required tasks would be performed by an omnibus end effector consisting of a nozzle for watering, a taser type system for parasite elimination, and a stereoscopic imaging system. Furthermore, image processing would be done by a machine learning algorithm with communication to the user handled by a web app.

3.1 Mechanical Concept

Given the decision to create a polar robot, NILE took inspiration from the ubiquitous center-pivot irrigation system in our design. Our proposal consists of a rotational joint about the center, a horizontal translational joint along the gantry, and a vertical translational joint at the end effector. With these three joints the robot can reach any point within the circular growing zone.

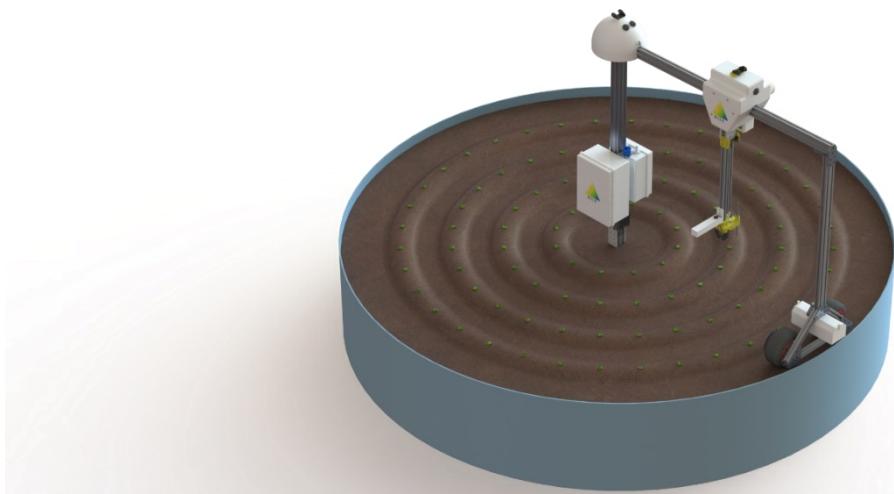


Figure 1. Full System Conceptual Design Render

As can be seen in Figure 1, rotation of the system will be achieved via motorized wheels at the end of the gantry to maximize torque with feedback provided by an encoder mounted to the central shaft. The trolley will move along the gantry via a motor driven, high friction wheel and determine position based on the rotation of an idler.

Watering, fertilizing, sensing, and weeding will be performed by an omnibus end effector that vertically translates on a lead screw system. The end effector will consist of a nozzle for watering and fertilizing, a High Voltage Elimination Circuit (HVEC) for weed elimination, and various sensors.

3.2 Electrical Conceptual

The system will be powered via a 120 Vac mains connection which will be regulated to the voltages required by the various actuators, computers, and sensors. The power supply will be capable of providing around 20A of continuous current in to ensure the motors are properly driven and to feed the HVEC. This is illustrated below in Figure 2.

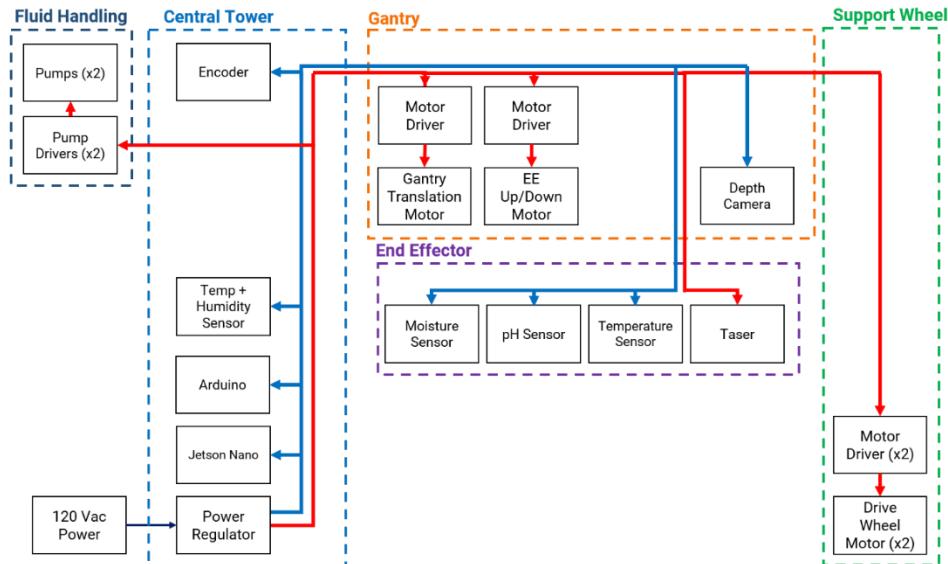


Figure 2. Electrical Block Diagram

The system will include two control units, a system determination computer (SDC) and a hardware microcontroller (HM). This system architecture allows for image and data processing to occur in parallel with low level motor control and data acquisition. The two computers will communicate via digital data line.

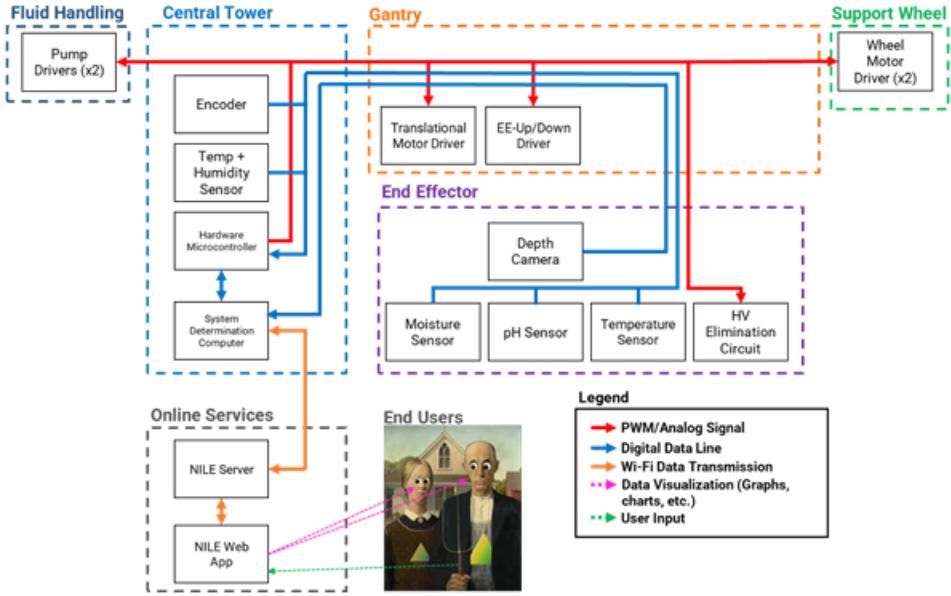


Figure 3. Data Block Diagram

As can be seen in Figure 3 above, the SDC will request data from the HM, process it, and then issue commands back to the HM. The SDC also uploads the processed data to an external server so it can easily be displayed to the end user via a web application. This server connection will be bidirectional so users can specify certain instructions (e.g., watering amounts, times, fertilizer thresholds, etc.) for the system to follow.

3.3 Software Concept

To tend to plants within the growing zone, the system will implement a waypoint inspection algorithm. When not active, the system will remain in a computational sleep state. At regular intervals, the SDC will initiate inspection of the growing zone by positioning the end effector at predetermined waypoints for analysis. Upon reaching a waypoint, the SDC will capture an image from the stereoscopic camera and perform a combination of image processing and machine learning synthesis. This flowchart can be seen below in Figure 4.

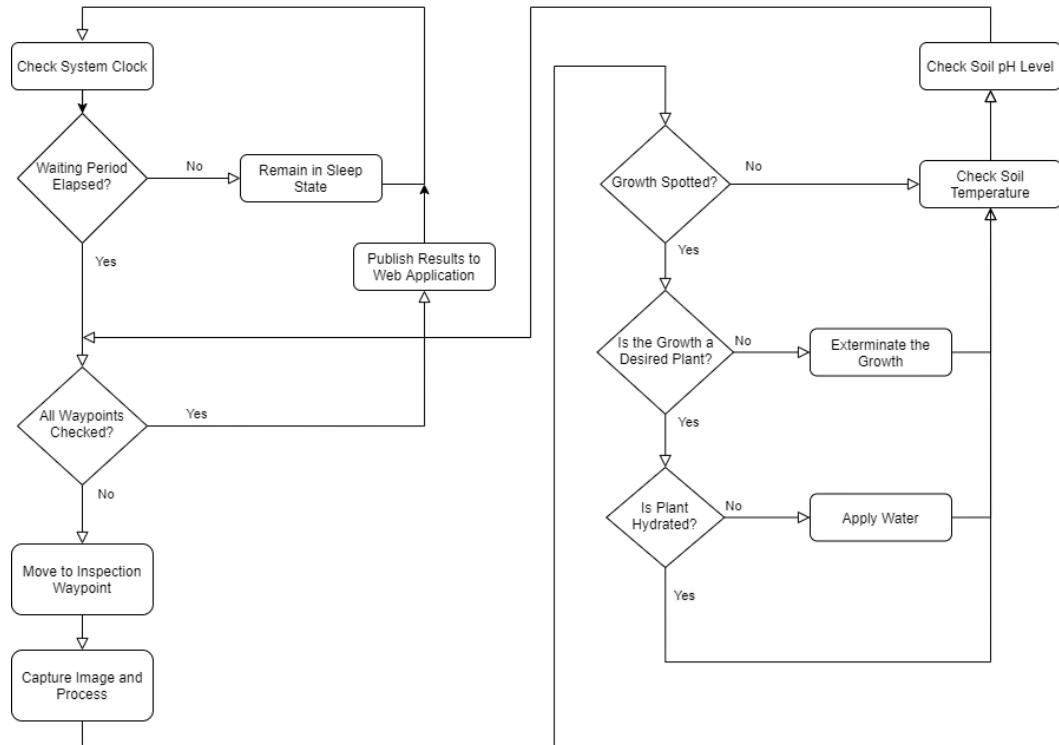


Figure 4. Plant Care Flowchart

The waypoints discussed above will be spaced such that a collage of images taken at each location will produce a complete picture of the growing zone. Then, using the capabilities of stereoscopic imaging, separation of foreground and background can be performed. Additionally, a point cloud of image depths can be applied to further distinguish individual leaves and plant structures that overlap and occlude one another.

The determination of plant location and health metrics within the growing zone will be accomplished via a combination of computer vision image processing and machine learning inference. An overview of this process can be seen in Figure 5 below.

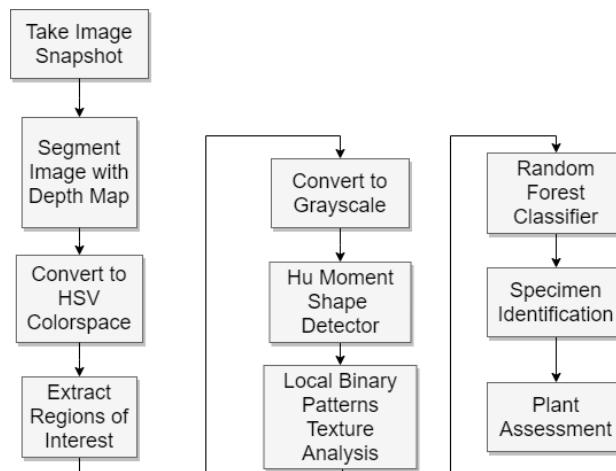


Figure 5. Computer Vision and Machine Learning Diagram

This algorithm, a form of supervised computer learning, will generate conclusions about the image contents in real time. Because the classifier is supervised, it will be pre-trained with a set of test images. This dataset will be a mixture of plants and weeds, healthy and unhealthy. When implemented, this classifier will be able to distinguish crops from weeds, as well as generating verdicts for healthy versus diseased leaf structures OpenCV computer vision library.

With image processing complete, the system will feed the adjusted images into an image-classification scheme powered by the Random Forests Classifier, a form of machine learning. Pre-trained with a vast database of plant and weed images, this classifier will reference the ROI shape and texture attributes and generate a conclusion on whether the prospective plant is a crop or weed, and upon identifying a plant, it will assess the health status of the foliage based upon the content of green, and texture patterns.

4.0 System Specifications

To ensure the preliminary design meets the requirements and ultimately resolves the problem NILE set out to solve, the following specifications have been defined.

4.1 Measuring Soil Water Saturation

The system must be capable of measuring the water saturation for the soil at any point within the growing zone. To achieve this, NILE defines the following specifications.

- 1) The moisture sensor shall have a minimum resolution of $\pm 5\%$ field saturation.
 - a) This value was chosen to ensure the system had sufficient resolution when determining how to water the soil.
- 2) The moisture sensor shall have an operating range of 0% – 100% field saturation.
- 3) The moisture sensor shall be capable of taking two independent readings within 10 cm of each plant's foliage exterior.

To verify the resolution of the water saturation sensor, a series of test systems will be set up with a known amount of soil and water. The sensor will be directed to measure the saturation of the test soil and the measurement will be compared to the known ratio of water to soil.

4.2 Measuring Soil Temperature

The system must be capable of measuring the temperature of the growing zone. To achieve this, NILE defines the following specifications.

- 1) The temperature sensor shall have a minimum resolution of $\pm 1^{\circ}\text{C}$.
- 2) The temperature sensor shall have an operating range including, but not limited to, -10°C to 50°C.
 - a) The soil will more than likely only vary temperature between -4°C and 17°C in Prescott, AZ [2].
- 3) The temperature sensor shall be capable of taking two independent readings within 10 cm of each plant's foliage exterior.

To verify the resolution of the temperature sensor chosen, the measurements taken from the soil will be compared to a known valid thermometer reading taken in the same spot. The two results will be compared to see if the chosen temperature sensor is within a 0.5°C accuracy of the control thermometer.

4.3 Application Efficiency

The system must be capable of irrigating crops within the growing zone with a field application efficiency greater than 90%. To achieve this, NILE defines the following specifications.

- 1) The liquid delivery system shall be capable of supplying water to the growing zone with a minimum rate of 40 mL/s.
- 2) The fluid distribution system shall be capable of measuring the flow rate of liquids input to the liquid delivery system and the flow rate liquids delivered to the end effector with a resolution of ± 2 mL/s.
 - a) ± 2 mL/s was chosen to ensure sufficient resolution to verify the field application efficiency requirement.

$$40 \frac{\text{mL}}{\text{s}} \times 90\% = 4 \frac{\text{mL}}{\text{s}} \text{ (max uncertainty)}$$

To verify the resolution of the flow meter, the input will be attached to a pump and the output to a beaker on a scale. The flow rate will be determined by the time taken to supply a certain volume of water to the beaker and compared against the flow rate measured by the sensor.

The flow meters will be used to verify the field application efficiency and flow rate through the fluid distribution system.

4.4 Maintain Plant Health

The system must be capable of supplying the nutrients needed to maintain plant health within the growing zone. To achieve this, NILE defines the following specifications.

- 1) The liquid delivery system shall be capable of supplying user defined liquid fertilizer to the growing zone with a minimum rate of 10 mL/s.

The flow meters will be used to verify the flow rate through the fluid distribution system.

4.5 Exterminating Weeds

The system must be capable of exterminating weeds from any point in the growing zone. To achieve this, NILE defines the following specifications.

- 1) The system shall be capable of positioning the end effector with a resolution of $\Delta e_{max} = \pm 0.5cm$ in the r, and z directions and a $\Delta\theta_{max} = \pm 0.145^\circ$ on the θ axis with respect to the growing zone origin.

- a) Given the 1-meter radius of the growing zone, the encoder shall have an angular resolution of at most 0.145° . This ensures that at maximum extension, the error in arc length is at most Δe .

$$\Delta\theta(\text{deg}) < \frac{\Delta e}{4\pi r} * 360 \rightarrow \Delta\theta(\text{deg}) < 0.145^\circ$$

- 2) The weed elimination end effector shall be capable of generating at least a 15kV pulsed arc with a discharge energy of at least 135 mJ [1].
 - a) This is sufficient to eliminate small weeds (4 – 6cm in height with a 1–3 mm stem diameter [1]).

To verify the precision of the end effector, the PhaseSpace motion tracking system will be used. With this system, tracking LEDs placed on the system can determine positions with under 1mm precision to comply with the specification.

To verify the discharge voltage, the maximum distance between the high voltage prongs that can produce an arc will be measured. Since air has a breakdown voltage of 30kV/cm, the voltage that the system can produce can be safely determined through that distance.

To verify the discharge energy ($E_{\text{discharge}}$), the voltage (V) derived previously used alongside the capacitance (C) of the HVEC can be used in the following equation [1].

$$E_{\text{discharge}} = \frac{1}{2} CV^2$$

Alongside this, if the HVEC can destroy weeds after the application of high voltage, its operation will be verified.

4.6 Plant Location Determination

The system shall be capable of determining the location of plant foliage, plant stems, and weeds within the growing zone. To achieve this, NILE defines the following specifications.

- 1) The detection algorithm shall be capable of determining the position of exterior plant and weed foliage with a resolution of +2 cm with respect to the stereoscopic camera in cartesian coordinates. [3]
- 2) The detection algorithm shall correctly differentiate between plants versus weeds with 99% identification accuracy.
- 3) The detection algorithm shall inspect the growing zone a minimum of twice per 24-hour period as the soil requires.
- 4) The detection algorithm shall complete all assessments prior to the start of the next inspection period.

To verify the precision of the plant detection, the PhaseSpace motion tracking system will be used. With this system, tracking LEDs placed in the growing zone can determine positions with under 1mm precision to comply with the specification.

4.7 IP55 Ingress Protection

The system shall be capable of being certified to an ingress protection level of IP55. To achieve this, NILE defines the following specifications.

- 1) The system shall maintain satisfactory operation in the presence of dust after exposure to sunlight over the course of 7 day/night cycles.
- 2) The system shall maintain satisfactory operation when subjected to pressurized water delivered from a nozzle with a minimum diameter of 6.3mm after exposure to sunlight over the course of 7 day/night cycles.

To verify that the system maintains satisfactory operation after the specified time, the system will be directed to operate as normal for an additional 7 days. As it will be operating in an inherently dusty environment this will verify the specification

After the dust testing is complete, the system will be moved from the growing zone and subjected to pressurized water over the course of 10 minutes. If the system maintains satisfactory operation after being subjected to dust and pressurized water, then the system will be verified.

4.8 Communication

The system shall be capable of communicating plant health, soil health, and the detection of weeds to the end user. To achieve this, NILE defines the following specifications.

- 1) The system shall send/receive data to and from a server through a WIFI connection, outgoing data will be displayed to the end-user via a web-application.
- 2) The web-application shall update the sensor readings within one minute of the system completing measurements.

The speed at which measurements are published to the end-user will be measured with a system clock that records the time between the system finishing all mechanical movements and receiving a data acknowledgement from the web-based application.

5.0 Preliminary Design

In the following sections NILE will define the robotics hardware and software developed from the conceptual design in accordance with the requirements and system specifications.

The conceptual design was developed with the goal of supporting a wide range of plants and climates. However, due to time and climate constraints, NILE has decided to choose a single plant to focus on for the preliminary design. Given the environment of northern Arizona, a cold-resistant and relatively fast-growing plant was needed, leading to *Valerianella locusta*.



Figure 6. *Valerianella Locusta*

Valerianella locusta, or mâche as it is more commonly known, is a small leafy green that grows in a low rosette structure. This can be seen above in Figure 6. Being able to survive temperatures as low as -28.9°F and only requiring partial sunlight, it is perfect for our needs [4]. Thus, the NILE system will be designed around, but not limited to, Mâche.

5.1 Mechanical Design of the Robot



Figure 7. NILE Mechanical Design Overview

As described previously and can be seen in Figure 7 above, the NILE system consists of a cylindrical robot that was heavily inspired by center-pivot irrigation systems. With a rotational joint about the center, a horizontal translational joint along the gantry, and a vertical translational joint at the end effector the robot can reach any point in the growing zone.

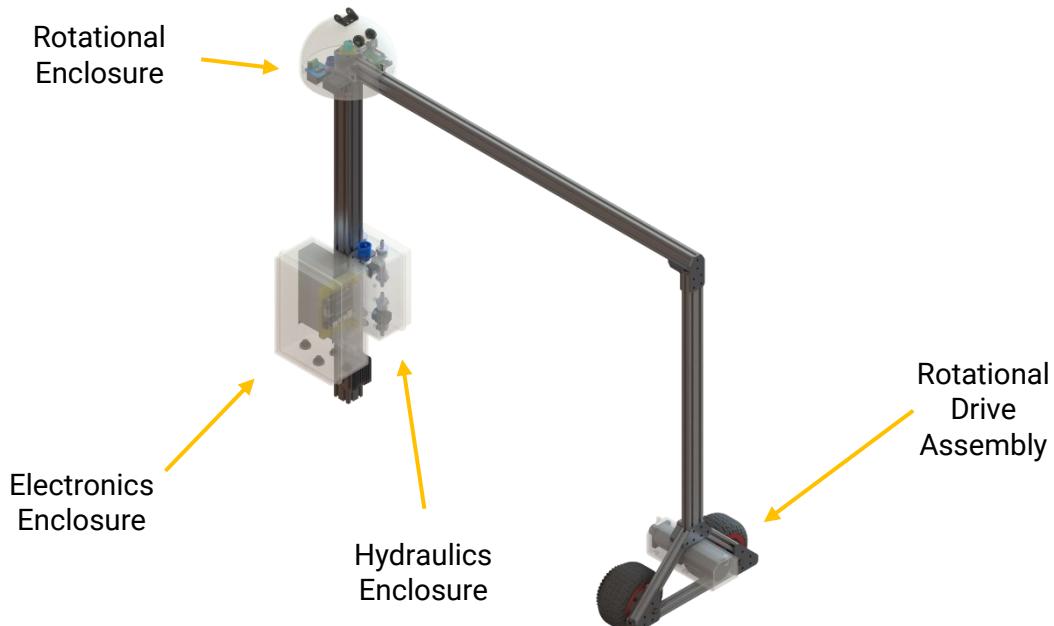


Figure 8. Gantry Assembly

The centerpiece of the mechanical design is the gantry system shown above in Figure 8. The central tower houses most of the electronic and hydraulic components in two weatherproof enclosures. In addition, the enclosure contains the power supply, system determination computer, and various hydraulic control systems. At the top of the tower sits the rotational enclosure and rotational joint. The enclosure protects the hardware microcontroller among other electrical components needed to control the rotation. The rotational joint itself consists of a central shaft upon which a flanged sleeve bearing rotates and supports the main gantry arm.

While the robot pivots about the central tower, the motion is not driven from there. The rotational drive assembly at the far end of the gantry consists of a geared DC motor that uses a chain drive to control two large pneumatic wheels. By increasing the moment arm, this configuration significantly decreases the power required to achieve stable control of the rotation. Furthermore, an absolute rotational encoder on the central shaft ensures that the angle of rotation is always known regardless of wheel slip.

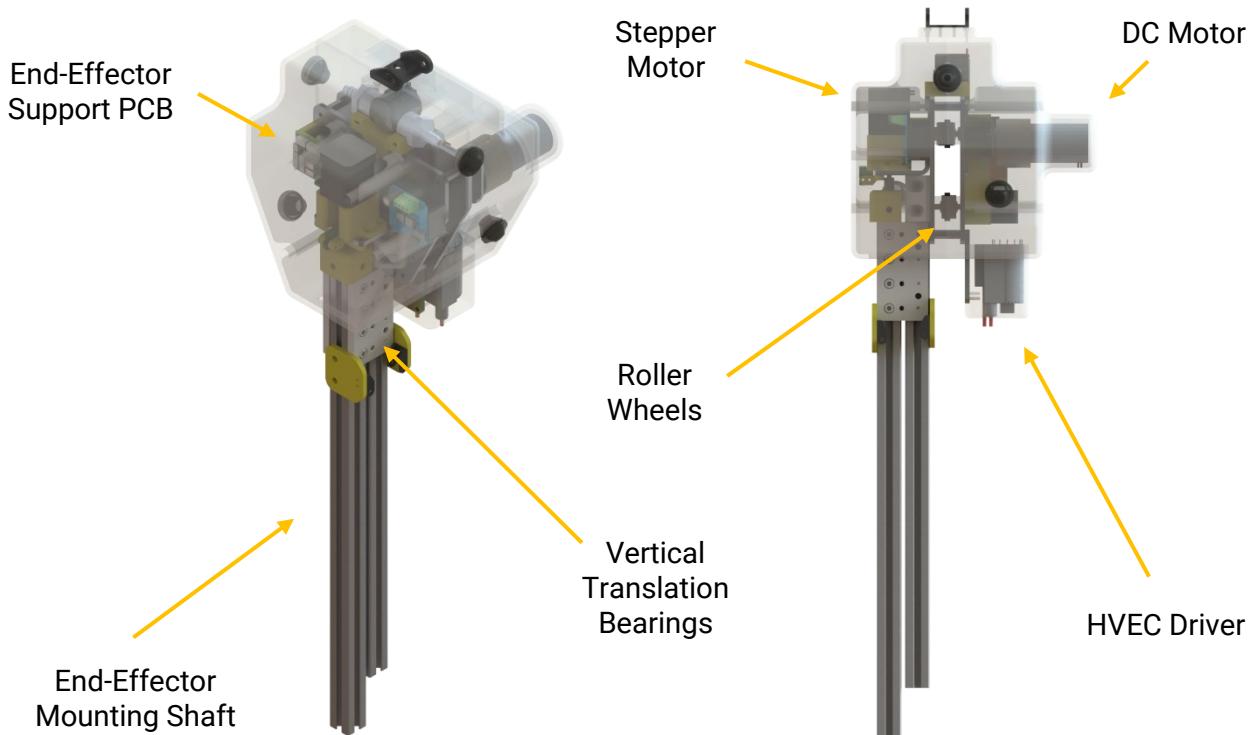


Figure 9. Trolley Assembly

The gantry assembly supports the most mechanically complex component of the NILE system, the trolley assembly, as seen above in Figure 9. The trolley manages the radial and vertical translation of the system in addition to housing the electronic hardware required to drive the end effector.

Radial translation is achieved by three high friction roller wheels that ride within the tracks of the gantry aluminum extrusion. One wheel is driven by a geared DC motor while the other two wheels stabilize the assembly and determine the location of the trolley via encoders attached to their drive shafts.

Vertical translation is accomplished via an ultra-precision lead screw driven by a high-torque stepper motor. The end effector mounting shaft rides on bearings slotted into the aluminum extrusion tracks and ensures that it is always secure.

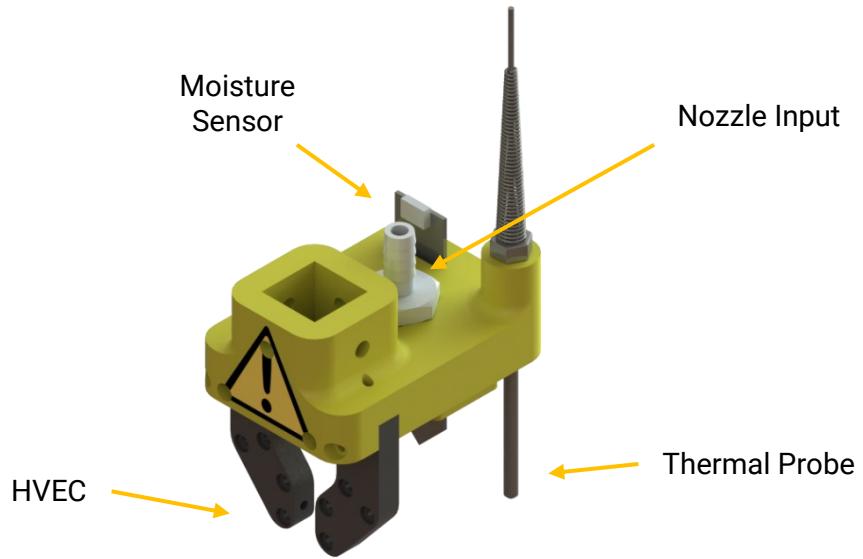


Figure 10. End Effector Assembly

Finally, there is the end effector assembly for which the entire system was designed to support. Mounted to the plastic base part with integrated nozzle, the capacitive moisture sensor and thermal probe record the required data. Furthermore, the heavily insulated HVEC system allows NILE to destroy all weeds in the growing zone.

5.1.1 Robotic Linkages

Throughout the rest of the document, NILE will refer to links one, two, and three on the robot. The links are separate from the major assemblies defined above as they are defined for mechanical and kinematic analysis. From left to right, Figure 12 below defines Links one, two, and three from left to right.

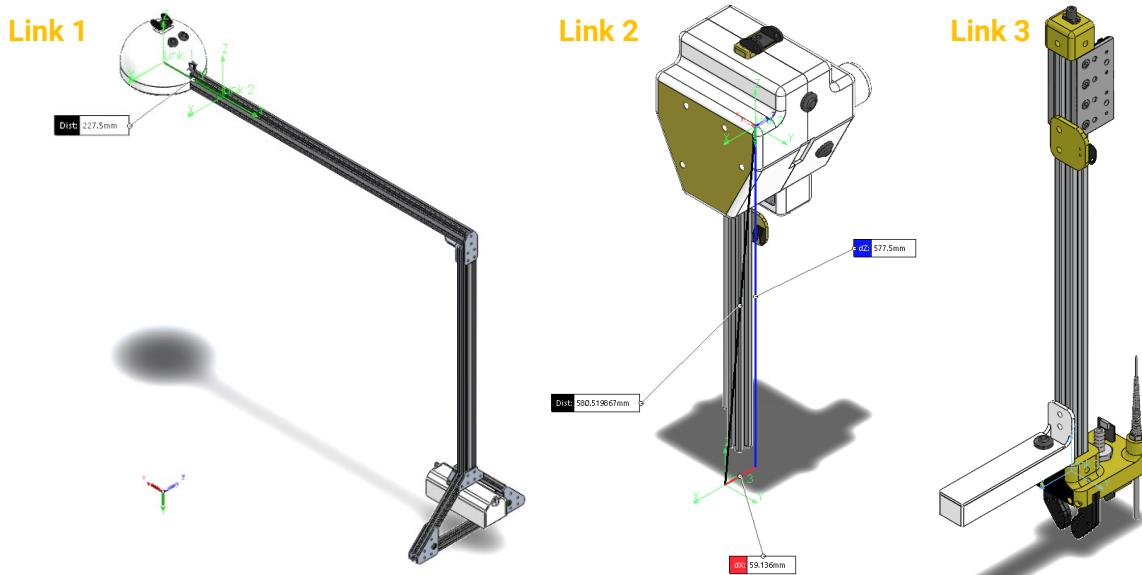


Figure 11. Robotics Linkages

5.1.2 Hydraulic System

The NILE hydraulic system supports the watering and fertilizing of all plants in the growing zone. To decrease the complexity of the system, it is designed to receive two pressurized inputs of water and liquid fertilizer. When fully deployed this would be as simple as a hose connected to mains water with an inline fertilizer douser. For our conceptual design NILE plans on constructing small water towers made of 55-gallon drums as we do not have access to mains water.

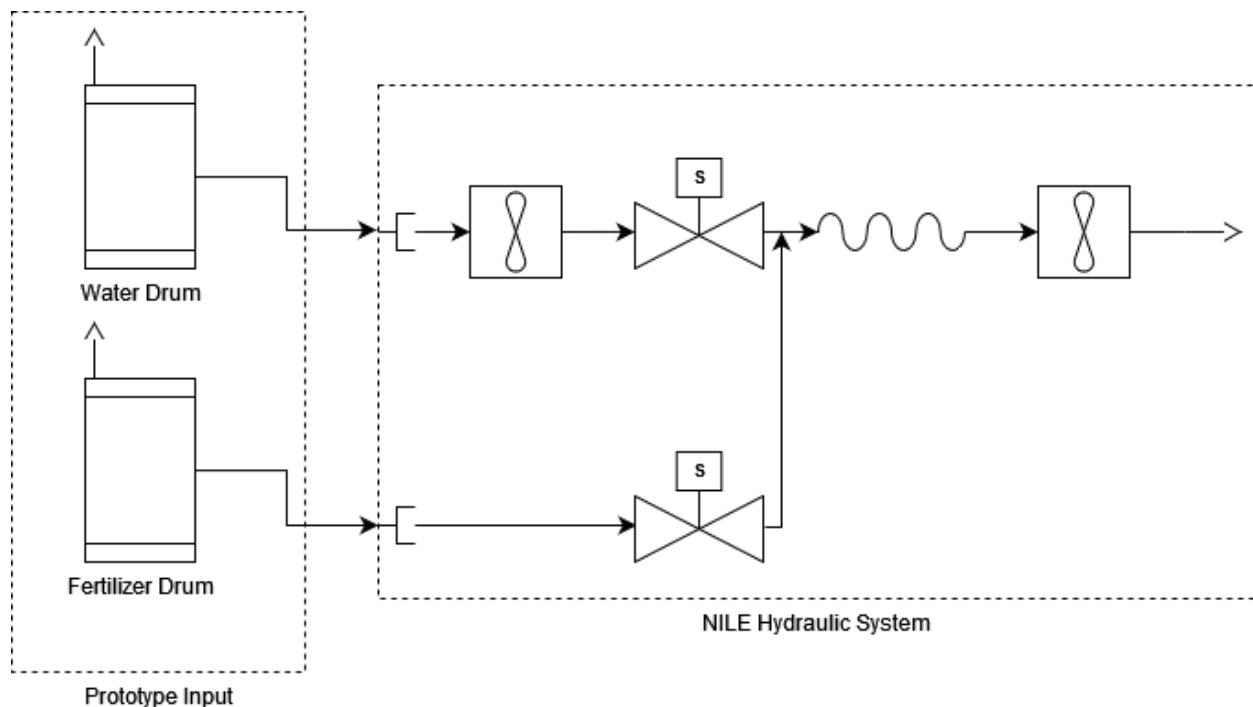


Figure 12. Hydraulic System Diagram

The complete hydraulic diagram for the NILE system can be seen in Figure 13 above. Upon input, the water flow rate will be immediately sampled with a turbine style flow meter before entering a solenoid valve. The fertilizer input passes through an additional solenoid valve before rejoining the nozzle line through a Y-fitting. The two solenoids control access to the nozzle based on the desired operation. Finally, the last flow meter provides feedback used to identify leaks and calculate the field application efficiency.

5.1.3 Structural Analysis

To ensure the proper operation of the system in nominal and edge case conditions NILE performed structural analysis on all the critical components. The load conditions were chosen as the system at rest with the trolley in the middle of the beam and the end effector driven into the ground during sensing operations.

5.1.3.1 System at Rest

Since our robot will be at rest for most of its life cycle, this state is critical to analyze and ensure stability and reliability. The four areas of concern identified for this state include the gantry beam, the various axles, and the stepper motor mount.

To analyze the main gantry beam, NILE performed a beam Finite Element Analysis [FEA]. A beam analysis was ideal for this application as it allows us to simulate the complex but uniform cross section of the aluminum extrusion over a long distance. For the simulation, the central shaft mount was assumed to be a fixed support, the rotational drive assembly as a roller joint, and the weight of the trolley / end effector as a remote load applied to the points where the wheels touch the extrusion. This can be seen below in Figure 13.

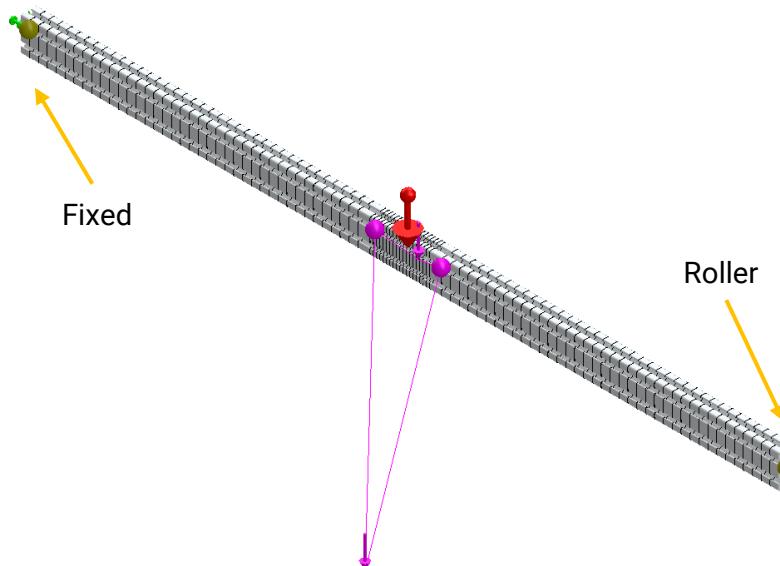


Figure 13. Gantry Beam Analysis

By applying a uniform gravitational load and running the simulation, the following stress and displacement plots were created. These can be seen below in Figures 14 and 15.

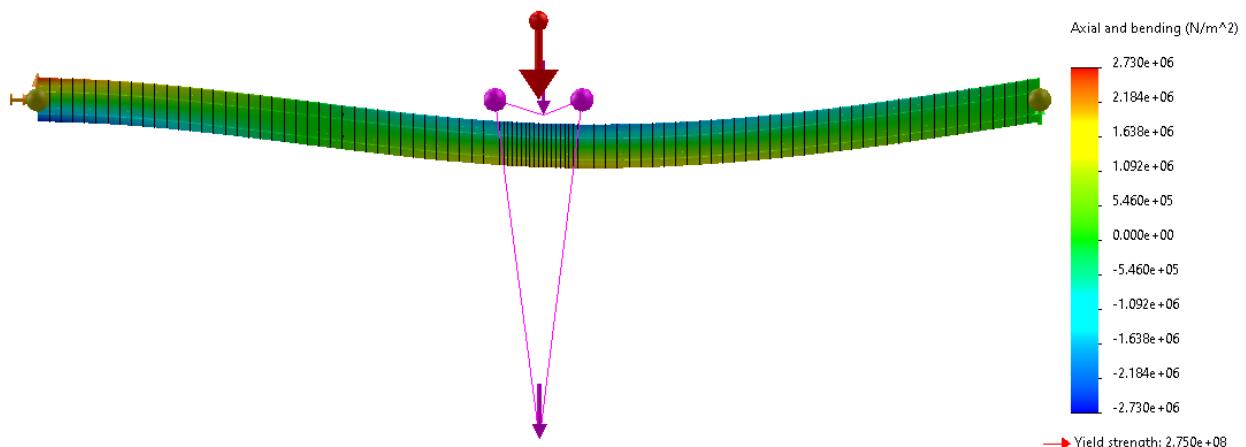


Figure 14. Gantry Beam Stress

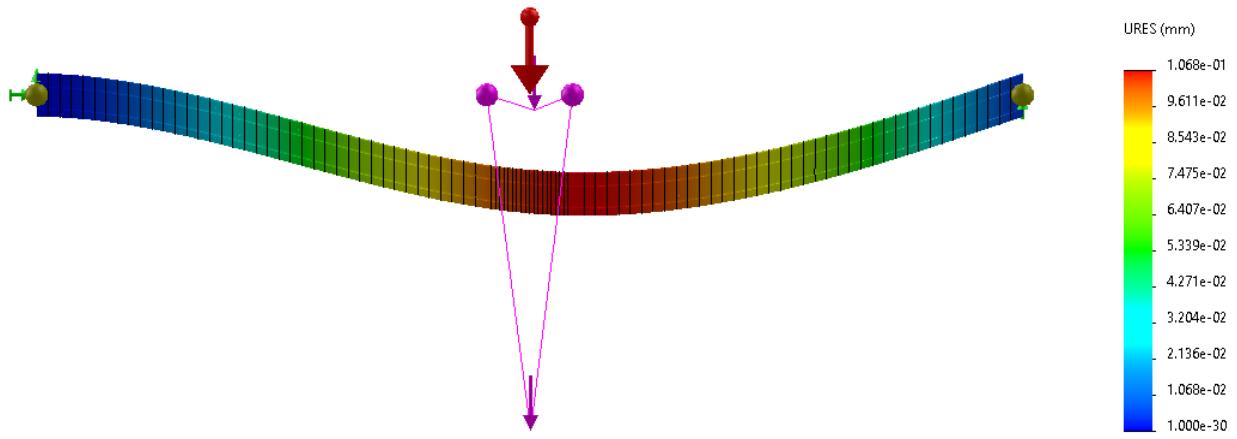


Figure 15. Gantry Beam Displacement

As seen in Figure 15, the maximum displacement is on the order of tenths of millimeters which is well within our positional accuracy specifications as defined in section 4.5. Furthermore, the maximum combined axial and bending moment stresses are significantly below the yield strength of the material.

Next, a finite element analysis was performed on the stepper motor mount to ensure it will be able to withstand the weight of the third link. For this analysis, the motor mount and the sheet metal baseplate share a no-interference bond and pre-torqued bolts fixing them together. In addition, the baseplate is fixed with foundation bolts and a virtual wall. Furthermore, the 3D printed mount was meshed as tetrahedral elements and the baseplate was modeled as shell element body. The mesh with an aspect ratio plot is shown below in Figure 16.

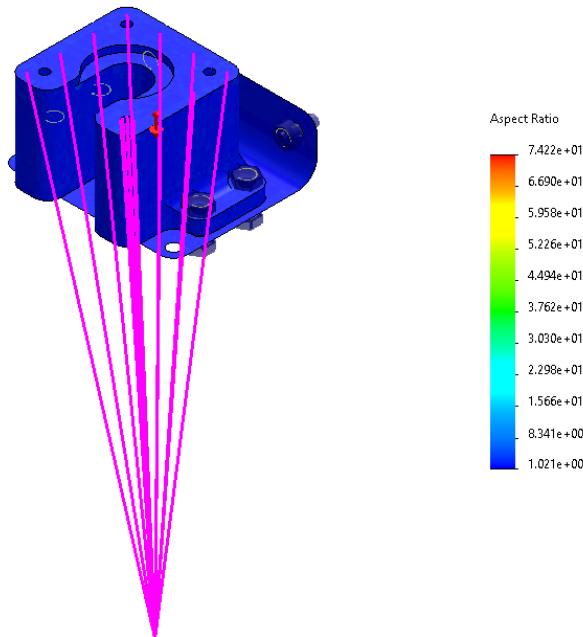


Figure 16. Motor Mount Gravitational Analysis

To simulate the load, a remote distributed mass corresponding to link three was applied to the surface of the motor mount and gravity applied to the whole model. 3D printed parts are difficult to model as the many layers and infill patterns are not easily defined in FEA. For this analysis NILE assumed the motor mount was homogeneous ABS as the loading is in compression and will be printed with near 100% infill.

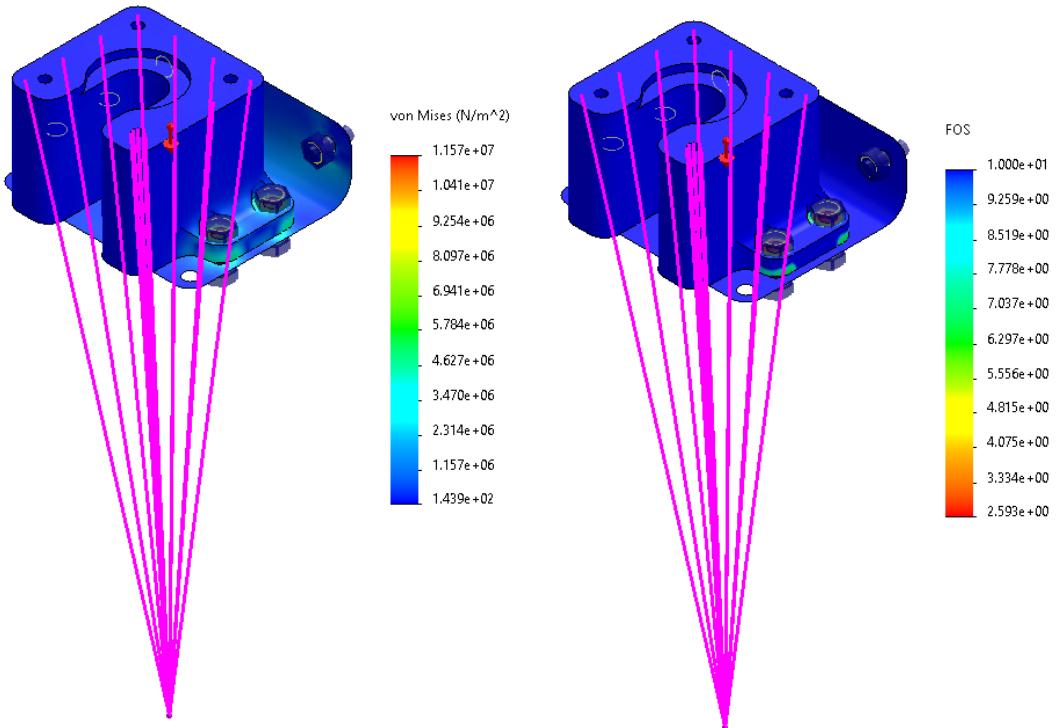


Figure 17. Motor Mount Gravitational Stress and FOS

As can be seen in Figure 17 above, the stresses in the parts are minimal with the lowest safety factors being due to the pre-torque as opposed to the load itself.

5.1.3.2 End Effector Driven into Ground

The second load condition NILE analyzed was the end effector being pressed into the ground by the stepper motor/lead screw. The first step of the process was to determine if the maximum force exerted by the stepper motor was greater than the weight of the trolley assembly.

The maximum pushing force that the stepper motor can supply through the lead screw can be derived by solving the power screw mechanics equation as presented in Shigley's [16, page 402].

$$F_{push} = \frac{2 \times T_{lower}}{D_{pitch} \times \left(\frac{\pi \times \mu \times D_{pitch} - L_{Lead}}{\pi \times D_{pitch} + \mu \times L_{Lead}} \right)} = 41.35 \text{ N}$$

Then, the force of gravity applied on the gantry by link two is given by the following.

$$F_{grav} = W_{link2} \times 9.81 \frac{m}{S^2} = 52.01 \text{ N}$$

As seen from the equations above $F_{push} < F_{grav}$. Therefore, the sensing operation does not have a significant impact on the loading of the rest of the system. Given that, the next step was to analyze the components it does effect: end effector, translational support beams, lead screw, and motor mount.

To analyze the end effector, NILE did not incorporate the sensors or HVEC in the analysis as those non-3D printed parts would experience relatively low insertion forces. Instead, NILE focused on the 3D printed end effector mounting part and the nozzle.

As the lowest, large flat surface on the end effector the load found above is applied evenly across the nozzle surface. The nozzle is attached to the mounting part by a no-interference bond and pre-torqued bolts and the mounting part is fixed using four foundation screws and five virtual walls representing the aluminum extrusion support beam. Both components were then meshed using tetrahedral elements as shown below in Figure 18.

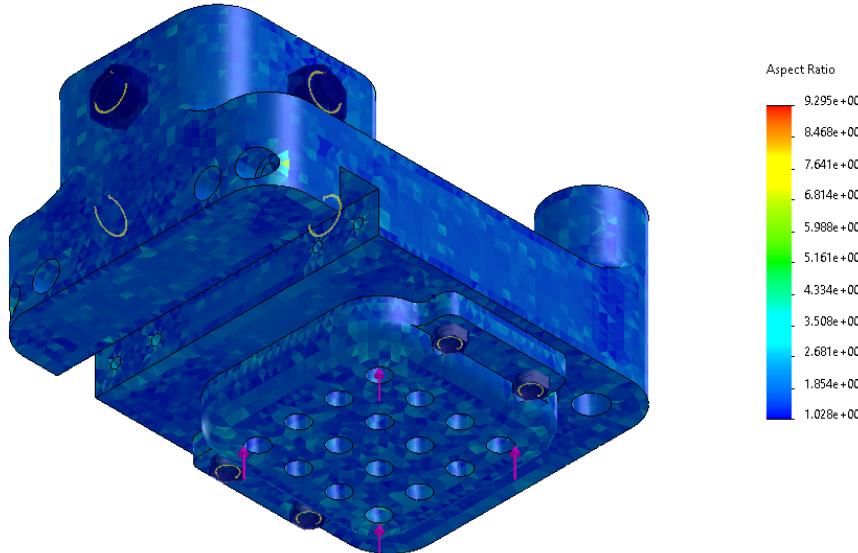


Figure 18. End-Effector Analysis

As mentioned previously, 3D printed parts are difficult to model and, as the parts will be experiencing significant cantilever loading, the results of this simulation must be critically examined.

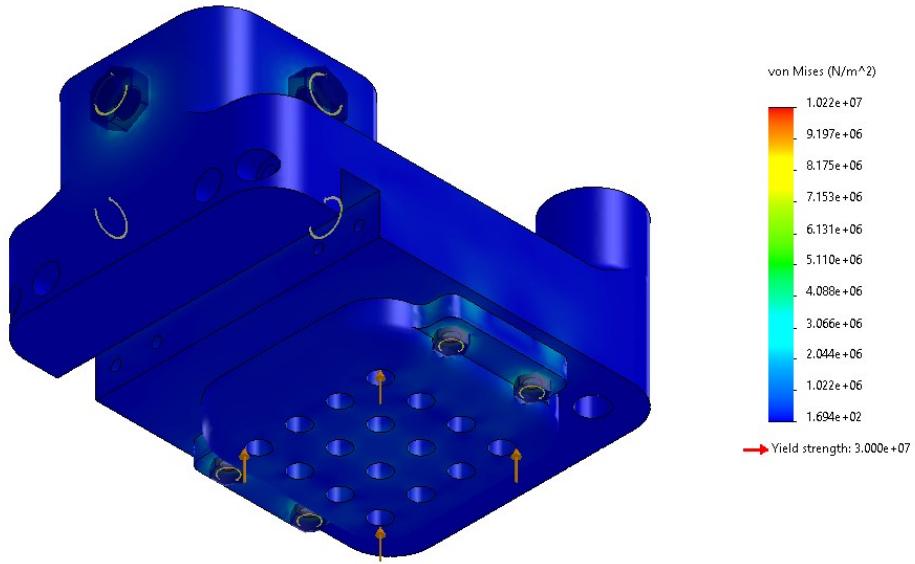


Figure 19. End-Effector Stress

Fortunately for this analysis, the loading is low enough that the components do not experience significant stresses and most of the geometry has a factor of safety greater than 5. Furthermore, the areas with the highest stresses appear to be stress concentrations linked to the screw pre-torques as seen in Figure 19 above.

With the end effector confirmed to be strong enough, the next components to model are the two support beams that allow for the vertical translation. The response of these components is somewhat difficult to model because of their large size and the complex interactions of the lead screw and linear bearings.

To simulate these interactions in FEA, both beams were modeled with beam elements. The static beam has one fixed end, representing its mounting point, with the other end rigidly attached to the translating beam, representing the bearing connection. The translating beam will have an addition fixture in the vertical direction representing the lead screw nut. This setup is shown below in Figure 20.

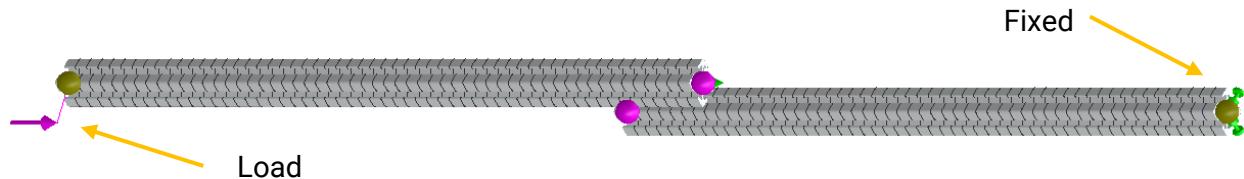


Figure 20. Vertical Translational Beam Analysis

Modeling the upward pushing force as a rigid, remote load and running the simulation, the following stress and displacement plots were created.

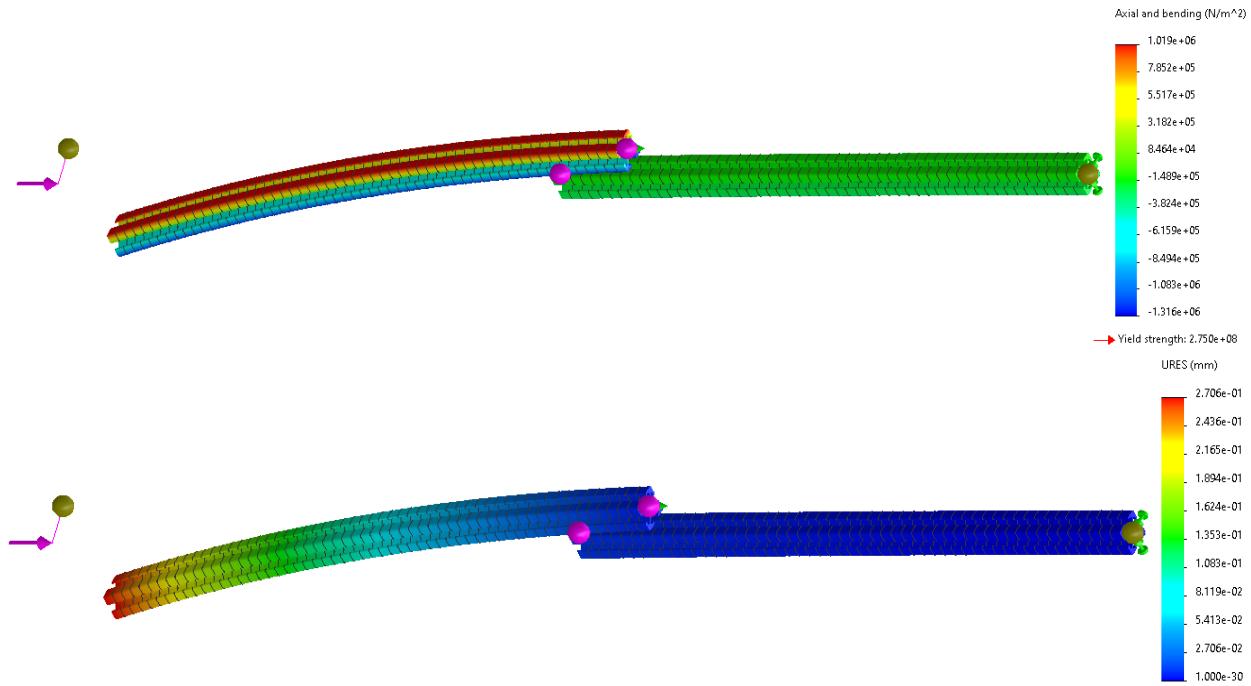


Figure 21. Vertical Translational Beam Stress and Displacements

As can be seen in Figure 21 above, the maximum combined axial and bending moment stresses are significantly below the strength of the aluminum and the maximum displacement is roughly 0.1mm, well within our accuracy specifications.

To ensure the lead screw could survive the compressive thrust load one could start from a basic machine design analysis to calculate the maximum stress at the roots of the screw threads. However, the plastic lead nut is more likely component to fail and the stainless-steel lead screw. The lead nut is rated by the manufacturer to withstand up to 25 lbf (111 N) of dynamic thrust load which is far more than the 41N the stepper can apply.

The final assembly to model is the motor mount and base plate discussed previously. For this analysis the components have been meshed and fixed in the same way. Here, the upwards force calculated above is applied to the screw holes that secure the motor to the mount as the force will be transmitted through the lead screws to the motor itself.

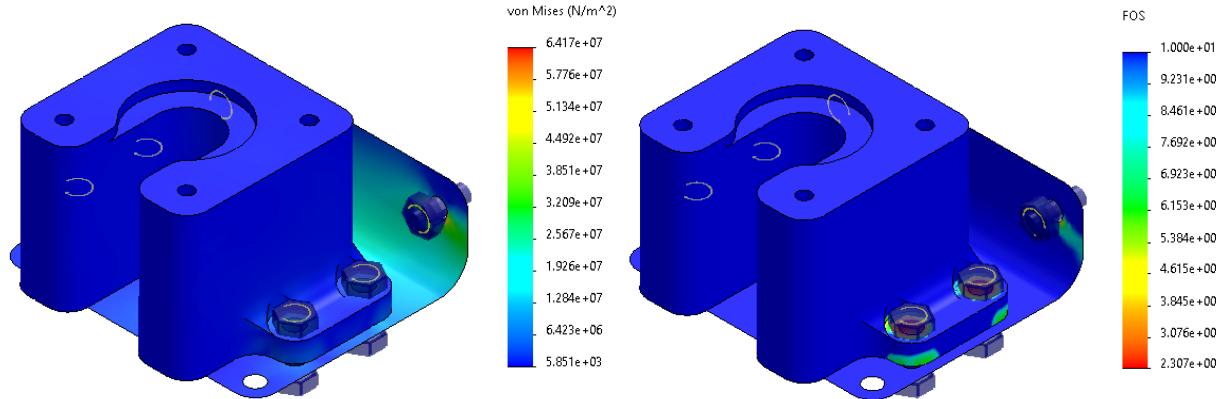


Figure 22. Motor Mount Stress and FOS

As can be seen in Figure 22 above the sheet metal base plate is experiencing significantly more loading than the simple gravitational load case but the stresses are still well within the factors of safety for both materials.

5.2 Electrical Design of the Robot

5.2.1 Power Distribution

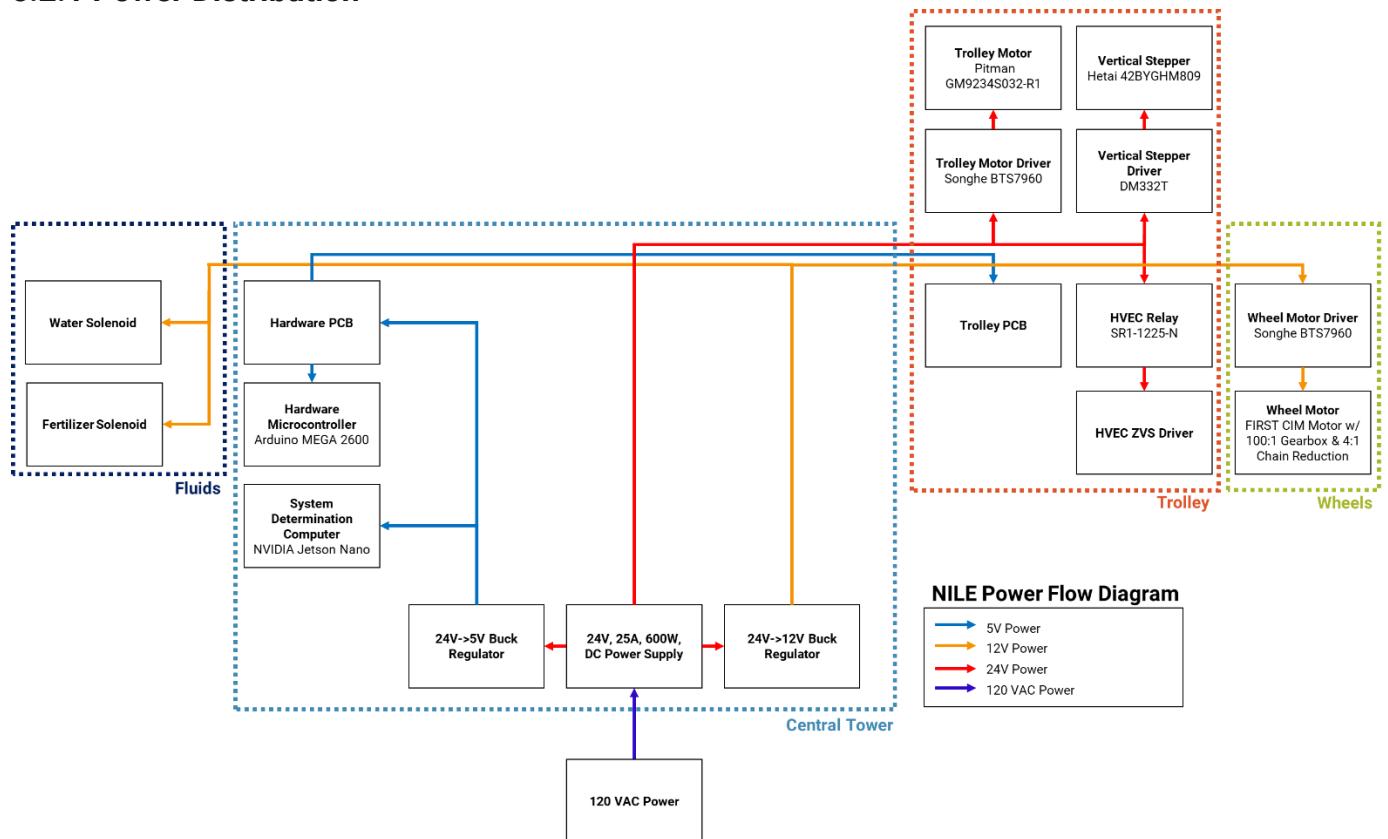


Figure 23. Power Distribution Diagram

The robot will operate off 120VAC power routed into a 24V, 25A, 600W DC power supply. This is more than adequate to drive all components on the robot, assuming that several of them aren't

run simultaneously. For example, the trolley motors, wheel motors, and HVEC have the potential to draw near 20A of current individually, therefore previsions will be added to the control software to ensure that only one of those is running at a time. Power is distributed across the system through three voltage levels. 5 volts is used to power microprocessors and sensors. 12 volts is used to power the fluid solenoids and wheel motor, and 24 volts is used to power both motors on the trolley as well as the HVEC. To regulate to the 5V and 12V levels, buck regulators are used. These boast high efficiencies (>90%) and can provide for the current requirements down the line. All this can be seen above in Figure 23.

Table 11. Worst Case Power Budget

Power Budget Motors and Solenoids Running			
Device	Voltage (V)	Maximum Current (A)	Power (W)
Main Power Supply	24.00	25.00	600.00
Vertical Stepper Motor/Driver	24.00	3.20	76.80
Trolley Motor/Driver	24.00	5.20	124.80
HVEC	24.00	0.00	0.00
24V-12V Buck Regulator	12.00	30.00	360.00
Wheel Motor/Driver	12.00	25.00	300.00
Water Solenoid	12.00	0.45	5.40
Fertilizer Solenoid	12.00	0.45	5.40
Total Consumption		25.90	310.80
% of Maximum Device Rating		86%	86%
Input Side Consumption @ 95% Efficiency	24.00	13.63	327.16
24V-5V Buck Regulator	5.00	5.00	25.00
NVIDIA Jetson Nano	5.00	2.00	10.00
Arduino Mega	5.00	0.20	1.00
Total Consumption		2.20	11.00
50% Variance from PCB Components		3.30	16.50
% of Maximum Device Rating		66%	66%
Input Side Consumption @ 90% Efficiency	24	0.76	18.33
Total Consumption		22.80	547.09
% of Maximum Power Supply Rating		91%	91%

Table 2. HVEC Power Budget

Power Budget HVEC Running			
Device	Voltage (V)	Maximum Current (A)	Power (W)
Main Power Supply	24.00	25.00	600.00
Vertical Stepper Motor/Driver	24.00	0.00	0.00
Trolley Motor/Driver	24.00	0.00	0.00
HVEC	24.00	15.00	360.00
24V-12V Buck Regulator	12	30	360
Wheel Motor/Driver	12	0	0
Water Solenoid	12	0	0
Fertilizer Solenoid	12	0	0
Total Consumption		0.00	0.00
% of Maximum Device Rating		0%	0%
Input Side Consumption @ 95% Efficiency	24.00	0.00	0.00
24V-5V Buck Regulator	5	5	25
NVIDIA Jetson Nano	5	2	10
Arduino Mega	5	0.2	1
Total Consumption		2.2	11
50% Variance from PCB Components		3.3	16.5
% of Maximum Device Rating		66%	66%
Input Side Consumption @ 90% Efficiency	24.00	0.76	18.33
Total Consumption		15.76	378.33
% of Maximum Power Supply Rating		63%	63%

As can be seen above in Tables 1 and 2, two power budgets were calculated to prove that power supply and its buck regulators would be able to handle the loading of a system. These include the worst-case condition of all motors and solenoids running, and the case of standard HVEC operation. While the HVEC is producing arcs, the root shall never be in motion. This is primarily

for the safety of nearby personal and for ensuring that the power supply doesn't get overdrawn. Further testing also unveiled that a steady arc was more effective at weed extermination, see Section 5.9.2.4.

5.2.2 Mechatronic Interfacing

5.2.2.1 Motor Selection

Table 3. Trolley Motor - Pitman GM9234S032-R1 [17]

Continuous Output Torque	500 oz*in	
Output Speed	16.6 RPM	
Gear Reduction	218:1	
Gear Efficiency	0.73	
Supply Voltage	24V	
Max Current	5.2A	

Table 4. Vertical Stepper – Hetai 42BYGHM809 [18]

Step Angle	0.9°	
Step Accuracy	±5%	
Holding Torque	420 mN*m	
Detent Torque	22.5 g*cm	
Rate Voltage	2.77 V	
Rate Current	2.4 A	

Table 5. Wheel Motor – FIRST CIM FR801-001 [19]

Continuous Output Torque	64 oz*in	
Output Speed	4320 RPM	
Gear Reduction	1:1	
Gearbox + Chain Reduction	400:1	
Supply Voltage	12V	
Max Current	27 A	

5.2.2.2 Motor Driver Selection

Table 6. DC Motor Driver – Songhe BTS7960 [20]

Operating Voltage:	6V – 27V	
Maximum Current:	43A	
Control Voltage:	3.3V – 5V	
Control Mode:	PWM	

Two BTS7960 DC motor drivers will be on the robot, powering the trolley motor (Pitman GM9234S032-R1) and the wheel motor (FIRST CIM FR801-001). It was selected as it had adequate current and voltage ranges, simple PWM control, and a built-in cooling solution.

Table 7. Stepper Motor Driver – STEPPERONLINE DM332T [21]

Operating Voltage:	10V – 30V	
Maximum Current:	3.2A	
Control Voltage:	3.3V – 5V	
Control Mode:	PWM	

The DM322T will be used to drive the vertical stepper motor, allowing control of the end-effector height with precision. This device was selected as its output current of 2.4A_{RMS} can produce the necessary torques on the 42BYGHM09 stepper.

5.2.2.3 Motor Capabilities

To ensure the trolley motor, Pitman GM9234S032-R1, and its driver were sufficient, NILE chose to analyze the system in the worst-case scenario. This was the motor driving the wheel in a full slip condition.

First, NILE calculated the torque loss due to friction in the axle bearings:

$$\tau_{\text{fric}} = W_{L2+L3} \times \mu_{\text{bearing}} \times r_{\text{shaft}} = \left(5.31\text{kg} \times 9.81 \frac{\text{m}}{\text{s}^2}\right) \times .08 \times 3\text{mm} = 0.0125\text{Nm}$$

Then found the torque applied by the wheel in a full slip condition:

$$\tau_{\text{slip}} = W_{L2+L3} \times \mu_{\text{rubber}} \times r_{\text{wheel}} = \left(5.31\text{kg} \times 9.81 \frac{\text{m}}{\text{s}^2}\right) \times .8 \times 12\text{mm} = 0.5\text{Nm}$$

Thus, the maximum torque that can be expected to act against the motor is found to be:

$$\tau_{\text{max}} = \tau_{\text{fric}} + \tau_{\text{slip}} = .5125\text{Nm} < \tau_{\text{motor,continuous}} = 3.53\text{Nm}$$

As can be seen above, the maximum torque the motor will experience is significantly less than the continuous output torque of the motor [17].

Next, to ensure the capabilities of the rotational drive motor, FIRST CIM FR801-001, NILE found the maximum torque expected of it under similar slip conditions.

First, the torque loss due to friction in the axle bearings assuming the full weight of links one, two, and three were applied:

$$\tau_{\text{fric}} = W_{L1+L2+L3} \times \mu_{\text{bearing}} \times r_{\text{shaft}} = \left(13.67\text{kg} \times 9.81 \frac{\text{m}}{\text{s}^2}\right) \times .08 \times 4.76\text{mm} = 0.0255\text{Nm}$$

Then, the torque applied by the wheel in a full slip condition was found assuming a fiction coefficient of 0.65 in dirt:

$$\tau_{\text{slip}} = W_{L1+L2+L3} \times \mu_{\text{dirt}} \times r_{\text{wheel}} = \left(13.67\text{kg} \times 9.81 \frac{\text{m}}{\text{s}^2}\right) \times .65 \times 12\text{mm} = 3.59\text{Nm}$$

Finally, the maximum torque required of the motor was found given our 4:1 chain and 100:1 gear reduction, assuming 98% and 85% efficiency respectively.

$$\tau_{max} = \frac{1}{N_{chain} N_{gear}} \times \frac{\tau_{fric} + \tau_{slip}}{\eta_{gear} \times \eta_{chain}} = .012Nm < \tau_{motor,continuous} = 0.45Nm$$

As can be seen above, the maximum torque the motor will experience is significantly less than the continuous output torque of the motor [19].

Lastly, when deciding upon the motor for the vertical translation, the first step was to determine if the frictional forces in the lead screw are greater than the weight of link 3. If so, the link would not fall under its own weight and a DC motor would be sufficient. Otherwise, a constant holding torque would be needed to maintain position which would make stepper motors the ideal choice.

To accomplish this a power screw frictional analysis was performed as presented in Shigleys [16, page 402]:

$$\begin{aligned}\tau_{lock} &= \left(W_{link3} \times \frac{D_{pitch}}{2} \right) \times \left(\frac{\pi \times \mu \times D_{pitch} - L_{Lead}}{\pi \times D_{pitch} + \mu \times L_{Lead}} \right) \\ \tau_{lock} &= (.9kg \times 9.81 \frac{m}{s^2}) \times \left(\frac{\pi \times .19 \times 4.76mm - 9.53mm}{\pi \times 4.76mm + .19 \times 9.53mm} \right) < 0\end{aligned}$$

Plugging in our system parameters, τ_{lock} is negative indicating that the third link will lower under its own weight [16]. Given that result, the Hetai 42BYGHM809 stepper motor was chosen to drive the lead screw. To ensure it was sufficient, the torque required to raise and lower the lead screw was found [16, page 402]:

$$\tau_{raise} = \left(W_{L3} \times \frac{D_{pitch}}{2} \right) * \frac{(1 + \pi \times \mu \times D_{pitch})}{(\pi \times D_{pitch} + \mu \times L_{Lead})} = 0.0357Nm < \tau_{stepper,continuous} = 0.42Nm$$

As can be seen above, the torque is well within the capabilities of the chosen stepper [18].

5.2.2.4 Encoder Selection

Table 8. Rotary Encoder – Avago AEAT-6012 [22]

Encoder Type	Absolute Magnetic	
Counts per rotation	4096	
Interface	SSI	
Supply Voltage	5V	

Table 9. Trolley Encoder – Avago HEDS-5640 [23]

Encoder Type	Quadrature	
Counts per rotation	1024	
Interface	Digital	
Supply Voltage	5V	

5.2.2.5 Positional Accuracy for Motors

To ensure the chosen encoders and stepper motor met the requirements and specification, a mechatronic analysis was performed on all three. The first linkage to be analyzed was link one whose angle is determined by an absolute, magnetic, rotary encoder with 4096 steps per rotation [22]. Thus, the resolution is as follows:

$$\Delta\theta = \frac{360^\circ}{4096} = \pm 0.087^\circ$$

This is significantly smaller than the $\Delta\theta_{max} = \pm 0.145^\circ$ as defined in the specifications and therefore meets our needs.

Second, the resolution of link two is to be determined by a three channel, optical, rotary encoder with 1024 steps per rotation [23]. It is affixed to an idler wheel on the rotational platform such that linear positions can be found by tracking rotations after leaving a known position.

Assuming no slip, the linear resolution can be calculated as:

$$\frac{\pi \times D_{wheel}}{N_{count/rev}} = \frac{\pi \times 2.4cm}{1024} = .007 \frac{cm}{count} \rightarrow \Delta e = \pm .007cm$$

Thus, the radial resolution of the NILE system is significantly higher than the $\Delta e_{max} = \pm 0.5cm$ defined in the specifications.

Finally, the vertical translation of link three is accomplished by a NEMA-17, Hybrid Stepper Motor with a step angle of 0.9° [18]. It is linked to an Ultra-Precision Lead Screw with a travel distance per turn of 0.375 inches. Using these properties, the linear resolution can be calculated as:

$$\frac{D_{per-rev}}{360^\circ} \times \frac{0.9^\circ}{step} = .002 \frac{cm}{step} \rightarrow \Delta e = \pm .002cm$$

As can be seen above, the vertical resolution of the NILE system is significantly higher than the $\Delta e_{max} = \pm 0.5cm$ defined in the specifications.

5.2.2.5 Flow Meter and Solenoid Selection

Table 10. Solenoid Valve– 997 [24]

Valve Type	Solenoid		
Nominal State	Closed		
Working Pressure	0.02 MPa - 0.8 MPa		
Supply Voltage	12V		

Table 11. Flow Meter– 828 [25]

Flow Meter Type	Turbine		
Volume per Pulse	2.25 mL		
Flow Rate	1 – 30 L/min		
Supply Voltage	5 – 18V		

5.2.3 High Voltage Elimination Circuit (HVEC)

As per requirement 6, the system must be able to eliminate weeds within the growing area. The NILE robot therefore will generate a high voltage electrical arc to burn those undesired plants. By doing this, it destroys the xylem tissue of the plant, disabling its ability to receive water and outright killing it. To produce the high voltage required for the HVEC, a flyback transformer is powered by a Zero-Voltage Switching (ZVS) driver. A ZVS Driver is a high efficiency switching device which utilizes two MOSFETs in an astable multivibrator configuration. Alongside several capacitors and the inductance of the transformer, this oscillates at resonance, supplying a much larger current to the transformer than what the supply can provide. This large current is reflected as an incredibly high (10s of kilovolts) voltage on the secondary of the transformer, creating electrical arcs. Below is a circuit of a ZVS driver with generic components. Our system will be using a prebuilt driver which contains all necessary components and a cooling solution.

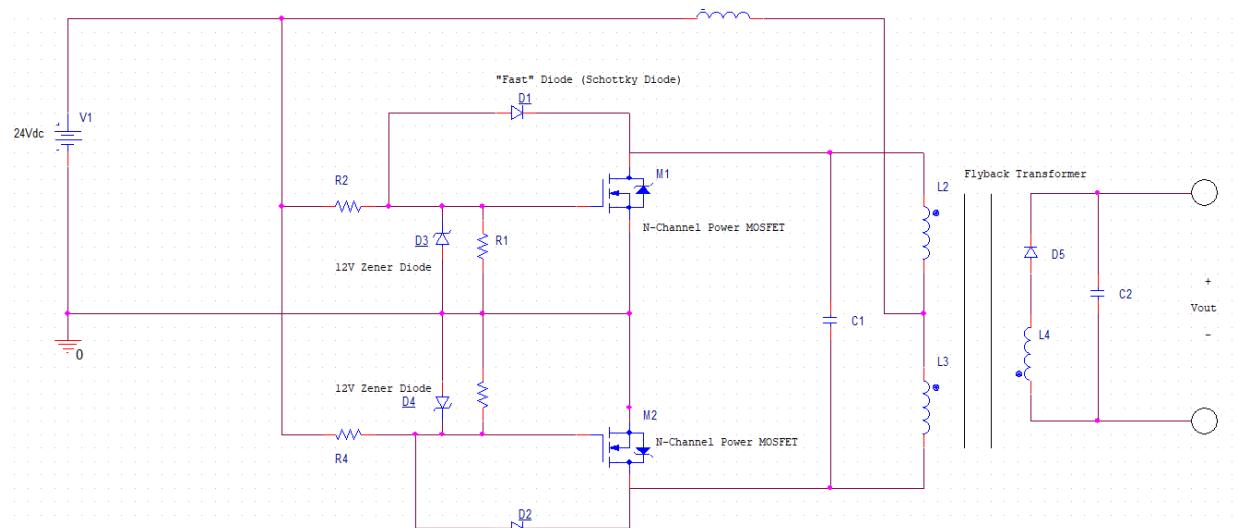


Figure 24. High-Voltage Elimination Circuit [26]

5.2.4 Data Distribution

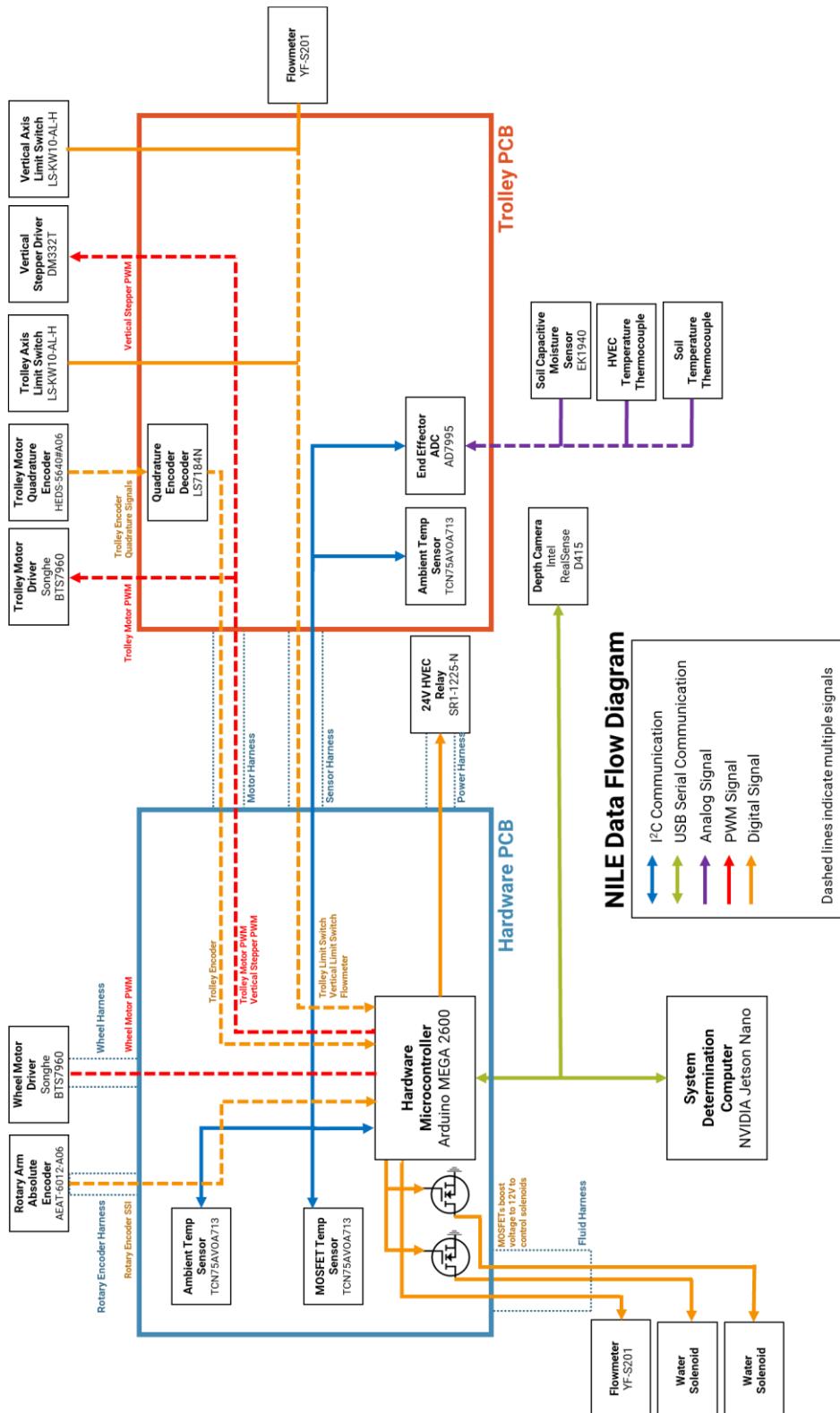


Figure 25. Peripheral Data Flow Diagram

5.2.5 Central Tower Electronics

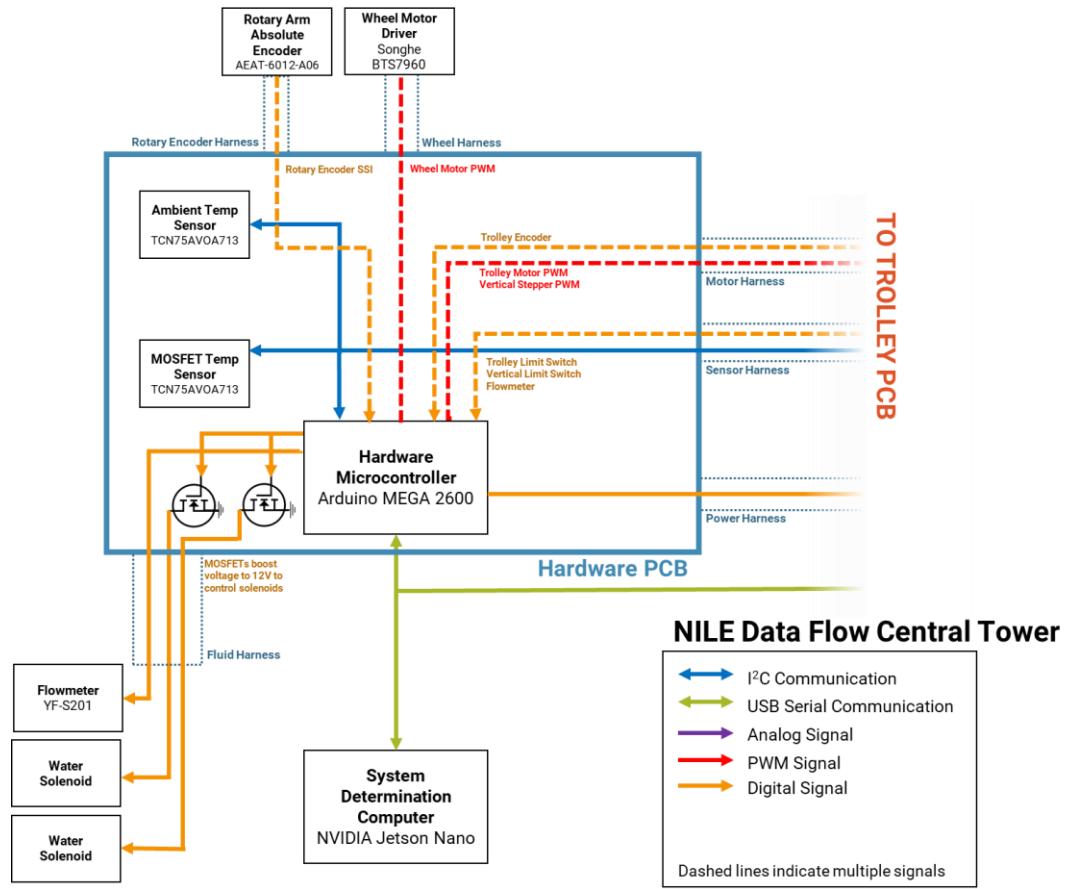


Figure 26. Central Electrical Diagram

The central tower of the robot contains all the *command-and-control components of the system as well as several important sensors and actuators.*

5.2.5.1 Microcontrollers



Figure 27. NVIDIA Jetson Nano

The System Determination Computer is the NVIDIA Jetson Nano, as seen in Figure 27, and serves as the main control unit of the system. It handles all data processing, command sequencing, computer vision, machine learning, and server hosting that the system requires. It interfaces with the robotic system through a USB Serial connection to the Hardware

Microcontroller. A NVIDIA Jetson Nano was chosen for this task as it is a powerful microcontroller with integrated support for computer vision/machine learning tasks.

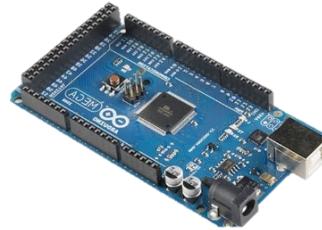


Figure 28. Arduino Mega

The Hardware Microcontroller is the Arduino MEGA 2560 and it handles all data acquisition and robotic control of the system. Sensor values are routed through specific harnesses over I²C and digital data lines. It also connects to and controls all motor drivers. A Serial connection between it and the System Determination Computer allows for the transmission of sensor information and reception of coordinates to bring the robot to. An Arduino MEGA 2560 was selected as its large number of digital pins allowed for interfacing with all necessary sensors.

5.2.6 Hardware PCB

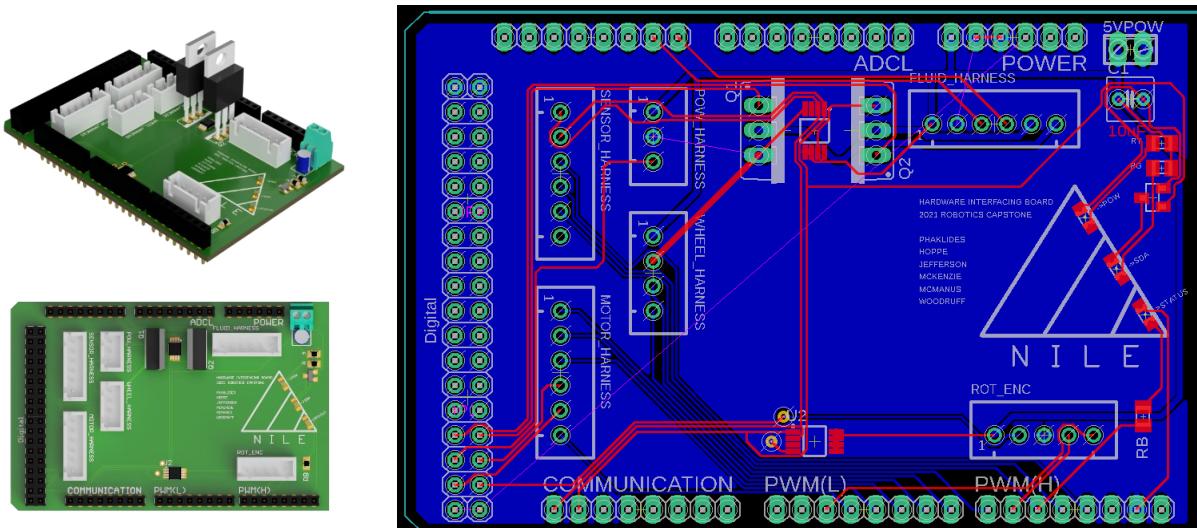


Figure 29. PCB for Arduino and Peripheral Interface

The Hardware PCB is designed as a hat which plugs directly on top of the Arduino Mega. It is responsible for routing all of the system's sensor and control signals to the Mega. This is done through several ribbon cable harnesses which interface to the board with JST-XH-A connectors. Two TCN75AVOA713 temperature sensors will record ambient and device temperatures. Indicator LEDs for power, I²C data, and a user-controlled states are included for testing purposes. The board and Mega are delivered 5V power through a screw terminal.

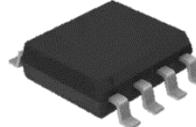
5.2.6.1 Electrical Harnesses

Table 124. Electrical Harnesses

<p>Rotary Encoder Harness</p> <p>Interfaces with the central tower rotational encoder (AEAT-6012-A06) through synchronous serial interface (SSI). Harness pins match the pins of the encoder. Incoming CLK and DO signals routed to interrupt pins to avoid polling.</p>									
<p>Wheel Harness</p> <p>Interfaces with the wheel motor driver through a forward/backward PWM signal. Provides 5V power.</p>									
<p>Fluid Harness</p> <p>Interfaces and provides power to the two flowmeters and solenoids on the fluid assembly. Flowmeter signals are routed to appropriate interrupt pins to avoid polling. Solenoids are driven through an N-Channel MOSFET (IRF540) passing 12V power. Under a maximum load of 27.8mW, heat sinks are most likely unnecessary. [28]</p> <table border="1" data-bbox="208 1083 780 1157"> <thead> <tr> <th>$i_{max,solenoid}$</th><th>R_{DS_on}</th><th>$P_{dissipated}$</th><th>P_{max}</th></tr> </thead> <tbody> <tr> <td>320mA</td><td>0.27Ω</td><td>27.8mW</td><td>60W</td></tr> </tbody> </table>	$i_{max,solenoid}$	R_{DS_on}	$P_{dissipated}$	P_{max}	320mA	0.27Ω	27.8mW	60W	
$i_{max,solenoid}$	R_{DS_on}	$P_{dissipated}$	P_{max}						
320mA	0.27Ω	27.8mW	60W						
<p>Power Harness</p> <p>Provides 5V power to the hardware PCB. Contains an external driving signal to the HVEC relay which allows power into the HVEC ZVS driver.</p>									
<p>Motor Harness</p> <p>Routes all necessary PWM control signals for the trolley motor and vertical stepper to the hardware PCB. Receives encoder information from the trolley quadrature encoder as a COUNT and UP/DOWN direction (decoded by LS7184N). Due to this, only one interrupt pin is required on the COUNT signal.</p>									
<p>Sensor Harness</p> <p>Interfaces the I²C Bus to the trolley PCB and its sensors. The signal for the flowmeter is routed to the interrupt pin. An additional sensor pin is reserved for potential use.</p>									

5.2.6.2 Device Descriptions

Table 13. I²C Temperature Sensor - TCN75AVOA713 [27]

Interface	I ² C	
Temperature Range	-40°C - 125°C	
Temperature +/-	0.0625°C	
Sample Rate	4 Hz	
Package	8-SOIC	

Measures the ambient temperature near the trolley board and transmits value over I²C bus.

Table 14. Solenoid Driver – IRF540 [28]

V _{DS,max}	100V	
I _{D,max}	28A	
V _{GS,thresh,max}	4.0V	
R _{DS,on,max}	0.27Ω	
P _{dissipation@25C}	60W	

Boosts solenoid control signal to 12V to properly drive the solenoids.

5.2.7 Trolley Electronics

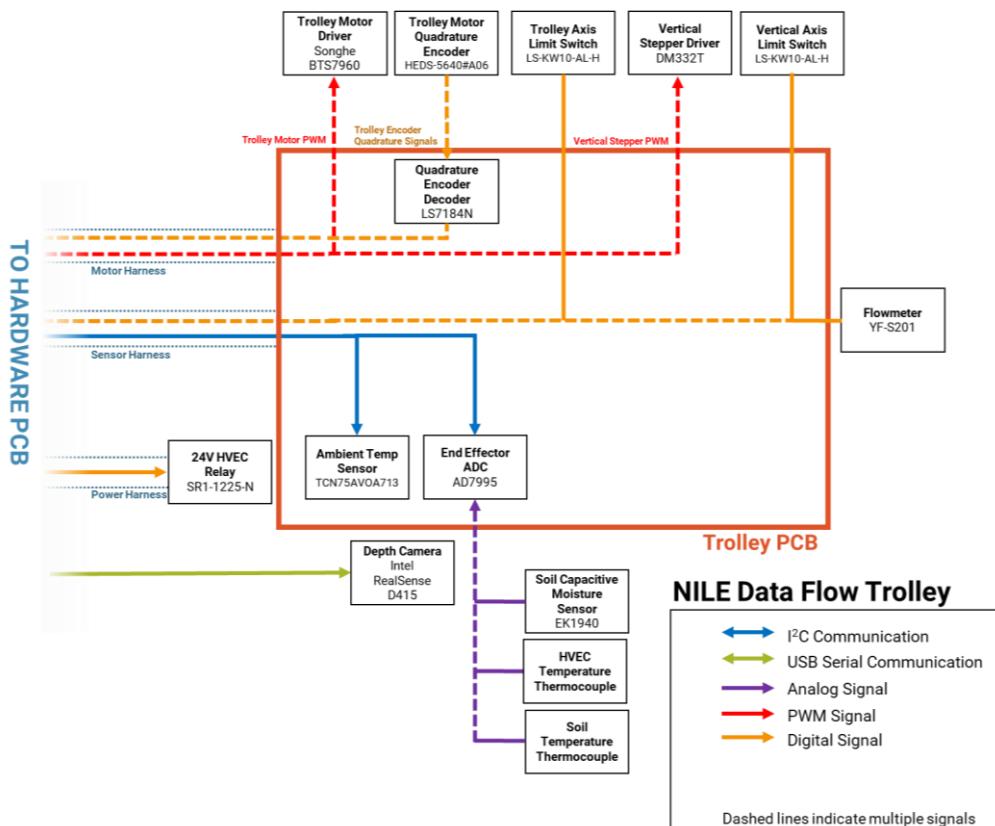


Figure 30. Trolley Electrical Diagram

5.2.8 Trolley PCB

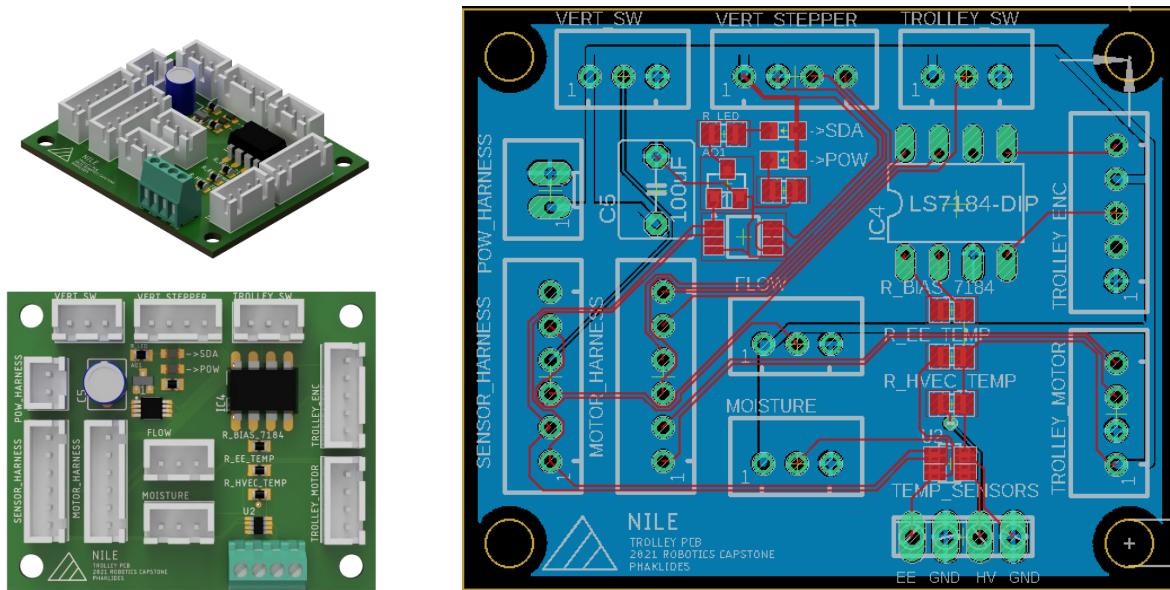


Figure 31. Trolley Electronics PCB

The Trolley PCB interfaces with all sensors and drivers near the trolley and routes those signals to/from the Hardware PCB. This is again done mainly through ribbon cable harnesses and JST-XH-A connectors. The board contains a TCN75AVOA713 temperature sensor to monitor ambient temperature, a LS7184N quadrature clock converter to convert quadrature encoder signals into a signed count, and an AD7995 ADC to read in the end effector's thermocouple measurements onto the I²C bus. Indicator lights for power and I²C data are included for testing purposes.

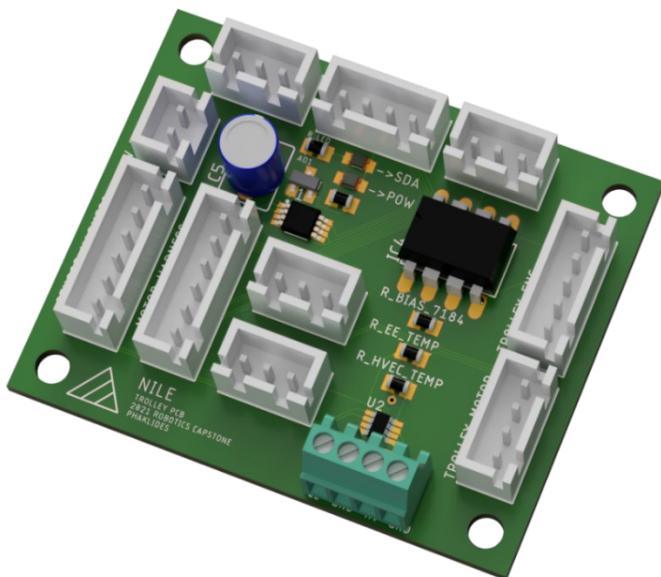


Figure 32. Perspective View of Trolley PCB

5.2.8.1 Electrical Harnesses

Table 15. Electrical Harnesses

Sensor Harness Interfaces the I ² C Bus to the hardware PCB. Signals for the trolley/vertical limit switches and trolley flowmeter are routed to interrupt pins. An additional sensor pin is reserved for potential use.	<pre> graph LR SH1[SENSOR HARNESS-1] --> SCL1[SCL] SH2[SENSOR HARNESS-2] --> SDA1[SDA] SH3[SENSOR HARNESS-3] --> TRANS_SW1[TRANS_SW] SH4[SENSOR HARNESS-4] --> VERT_SW1[VERT_SW] SH5[SENSOR HARNESS-5] --> TROLLEY_FLOW1[TROLLEY_FLOW] SH6[SENSOR HARNESS-6] --> GND6[GND] </pre>
Motor Harness Routes all necessary PWM control signals for the trolley motor and vertical stepper from the hardware PCB. Receives encoder information from the LS7184N.	<pre> graph LR MH1[MOTOR HARNESS-1] --> TROLLEY_PWM_F1[TROLLEY_PWM_F] MH2[MOTOR HARNESS-2] --> TROLLEY_PWM_B1[TROLLEY_PWM_B] MH3[MOTOR HARNESS-3] --> VERT_STEP1[VERT_STEP] MH4[MOTOR HARNESS-4] --> VERT_DIR1[VERT_DIR] MH5[MOTOR HARNESS-5] --> TROLLEY_ENC_UPDOWN1[TROLLEY_ENC_UP/DOWN] MH6[MOTOR HARNESS-6] --> TROLLEY_ENC_COUNT1[TROLLEY_ENC_COUNT] </pre>
Trolley Switch Harness Powers and receives a digital signal from the trolley limit switch.	<pre> graph LR TS1[TROLLEY_SW-1] --> V5V1[5V] TS1[TROLLEY_SW-1] --> GND1[GND] TS1[TROLLEY_SW-1] --> TSW1[TROLLEY_SW] TS2[TROLLEY_SW-2] --> V5V2[5V] TS2[TROLLEY_SW-2] --> GND2[GND] TS2[TROLLEY_SW-2] --> TSW2[TROLLEY_SW] TS3[TROLLEY_SW-3] --> V5V3[5V] TS3[TROLLEY_SW-3] --> GND3[GND] TS3[TROLLEY_SW-3] --> TSW3[TROLLEY_SW] </pre>
Trolley Motor Harness Powers logic level and delivers PWM signals to the trolley motor driver.	<pre> graph LR TM1[TROLLEY_MOTOR-1] --> V5V1[5V] TM1[TROLLEY_MOTOR-1] --> GND1[GND] TM1[TROLLEY_MOTOR-1] --> TROLLEY_PWM_F1[TROLLEY_PWM_F] TM1[TROLLEY_MOTOR-1] --> TROLLEY_PWM_B1[TROLLEY_PWM_B] TM2[TROLLEY_MOTOR-2] --> V5V2[5V] TM2[TROLLEY_MOTOR-2] --> GND2[GND] TM2[TROLLEY_MOTOR-2] --> TROLLEY_PWM_F2[TROLLEY_PWM_F] TM2[TROLLEY_MOTOR-2] --> TROLLEY_PWM_B2[TROLLEY_PWM_B] TM3[TROLLEY_MOTOR-3] --> V5V3[5V] TM3[TROLLEY_MOTOR-3] --> GND3[GND] TM3[TROLLEY_MOTOR-3] --> TROLLEY_PWM_F3[TROLLEY_PWM_F] TM3[TROLLEY_MOTOR-3] --> TROLLEY_PWM_B3[TROLLEY_PWM_B] TM4[TROLLEY_MOTOR-4] --> V5V4[5V] TM4[TROLLEY_MOTOR-4] --> GND4[GND] TM4[TROLLEY_MOTOR-4] --> TROLLEY_PWM_F4[TROLLEY_PWM_F] TM4[TROLLEY_MOTOR-4] --> TROLLEY_PWM_B4[TROLLEY_PWM_B] </pre>
Trolley Encoder Harness Powers and receives a quadrature encoder signal from the trolley quadrature encoder.	<pre> graph LR TE1[TROLLEY_ENC-1] --> GND1[GND] TE1[TROLLEY_ENC-1] --> ENC_A1[ENC_A] TE1[TROLLEY_ENC-1] --> ENC_B1[ENC_B] TE2[TROLLEY_ENC-2] --> GND2[GND] TE2[TROLLEY_ENC-2] --> ENC_A2[ENC_A] TE2[TROLLEY_ENC-2] --> ENC_B2[ENC_B] TE3[TROLLEY_ENC-3] --> GND3[GND] TE3[TROLLEY_ENC-3] --> ENC_A3[ENC_A] TE3[TROLLEY_ENC-3] --> ENC_B3[ENC_B] TE4[TROLLEY_ENC-4] --> GND4[GND] TE4[TROLLEY_ENC-4] --> ENC_A4[ENC_A] TE4[TROLLEY_ENC-4] --> ENC_B4[ENC_B] TE5[TROLLEY_ENC-5] --> GND5[GND] TE5[TROLLEY_ENC-5] --> ENC_A5[ENC_A] TE5[TROLLEY_ENC-5] --> ENC_B5[ENC_B] </pre>
Vertical Switch Harness Powers and receives a digital signal from the vertical limit switch.	<pre> graph LR VS1[VERT_SW-1] --> V5V1[5V] VS1[VERT_SW-1] --> VERT_SW1[VERT_SW] VS1[VERT_SW-1] --> GND1[GND] VS2[VERT_SW-2] --> V5V2[5V] VS2[VERT_SW-2] --> VERT_SW2[VERT_SW] VS2[VERT_SW-2] --> GND2[GND] VS3[VERT_SW-3] --> V5V3[5V] VS3[VERT_SW-3] --> VERT_SW3[VERT_SW] VS3[VERT_SW-3] --> GND3[GND] </pre>
Vertical Stepper Harness Powers logic level and delivers STEP and DIR signals to the vertical stepper driver.	<pre> graph LR VS1[VERT_STEPPER-1] --> V5V1[5V] VS1[VERT_STEPPER-1] --> GND1[GND] VS1[VERT_STEPPER-1] --> VERT_DIR1[VERT_DIR] VS1[VERT_STEPPER-1] --> VERT_STEP1[VERT_STEP] VS2[VERT_STEPPER-2] --> V5V2[5V] VS2[VERT_STEPPER-2] --> GND2[GND] VS2[VERT_STEPPER-2] --> VERT_DIR2[VERT_DIR] VS2[VERT_STEPPER-2] --> VERT_STEP2[VERT_STEP] VS3[VERT_STEPPER-3] --> V5V3[5V] VS3[VERT_STEPPER-3] --> GND3[GND] VS3[VERT_STEPPER-3] --> VERT_DIR3[VERT_DIR] VS3[VERT_STEPPER-3] --> VERT_STEP3[VERT_STEP] VS4[VERT_STEPPER-4] --> V5V4[5V] VS4[VERT_STEPPER-4] --> GND4[GND] VS4[VERT_STEPPER-4] --> VERT_DIR4[VERT_DIR] VS4[VERT_STEPPER-4] --> VERT_STEP4[VERT_STEP] </pre>
Trolley Flow Harness Powers and receives digital clock signal from the trolley flowmeter.	<pre> graph LR F1[FLOW-1] --> V5V1[5V] F1[FLOW-1] --> TFLOW1[TROLLEY_FLOW] F1[FLOW-1] --> GND1[GND] F2[FLOW-2] --> V5V2[5V] F2[FLOW-2] --> TFLOW2[TROLLEY_FLOW] F2[FLOW-2] --> GND2[GND] F3[FLOW-3] --> V5V3[5V] F3[FLOW-3] --> TFLOW3[TROLLEY_FLOW] F3[FLOW-3] --> GND3[GND] </pre>

Moisture Harness Powers and receives analog data from the end effector moisture sensor.	
Temperature Screw Terminals Receives analog signals from the end effector and HVEC thermocouples.	

5.2.8.2 Device Descriptions

Table 16. I²C Temperature Sensor - TCN75AV0A713 [25]

Interface	I ² C	
Temperature Range	-40°C - 125°C	
Temperature +/-	0.0625°C	
Sample Rate	4 Hz	
Package	8-SOIC	

Measures the ambient temperature near the trolley board and transmits value over I²C bus.

Table 17. Four Channel 10 Bit I²C ADC - AD7995 [27]

Interface	I ² C	
Measurement Range	0 - 5V	
Measurement +/-	4.88mV	
Sample Rate	140 kHz	
Package	SOT-23-8	

Digitizes end effector temperature thermocouples and transmits values over I²C bus.

Table 18. Quadrature Clock Converter - LS7184N [28]

Interface	Digital	
CLK Pulse Width	225ns	
Sample Rate	588.24 kHz	
Package	DIP-8	

Converts quadrature encoder signals A, B into a clock signal and direction, which can be read as a counter in software. This reduces the number of required interrupt pins from two to one.

Table 19. Limit Switch - LS-KW10-AL-H [29]

Interface	Digital	
Features	Debouncer	
Ingress Protection	IP54	
Mechanical Life	500,000 Cycles	

Used to calibrate the origin of the trolley and vertical movements. Contains a built-in debouncer and is rated to IP54 (dust protected, water resistant).

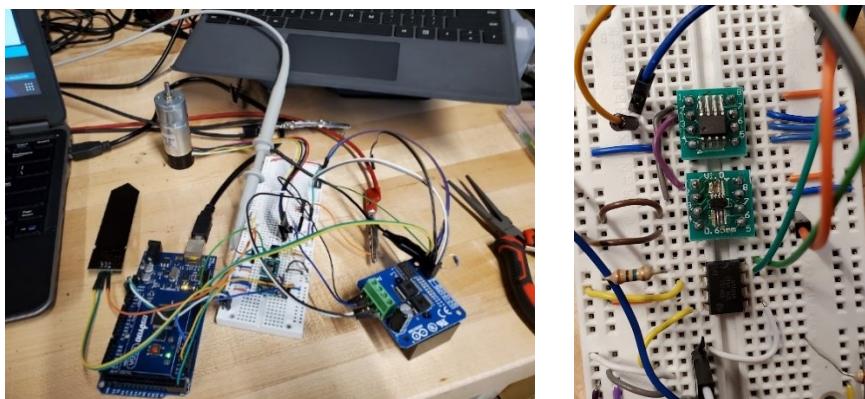
Table 20. Capacitive Moisture Sensor – EK1940 [30]

Interface	Analog	
Operating Voltage	3.3V – 5V	
Operating Current	5mA	
Output Voltage	0 – 5V	

Provides an analog output related to the capacitive effects of moisture in soil. Typically, higher voltages correspond to relatively higher moisture contents and vice versa.

5.2.9 Electrical Testing

To ensure that electrical components would work prior to ordering the PCBs, several were purchased to develop libraries for the Arduino Mega. In similar manner, the HVEC and ZVS Driver were purchased to test the effectiveness of that system. The test setup can be seen below in Figure 33.

**Figure 33. Electrical Test Set-up**

5.2.9.1 TCN75AVOA713 Temperature Sensor

The library `<temp_i2c.h>` was created to handle all I²C interfacing with the TCN75AVOA713. It allows the user to specify new TCN75AVOA713 devices via their addresses, which automatically initializes them with the maximum precision setting ($\pm 0.0625^\circ\text{C}$). The class contains a function to read the temperature of the device as a double. Below, a test program outputs the recorded temperature in Celsius over the serial plotter. In this case, the sensor at ambient temperature is touched by a finger. It then cools off to ambient temperature.

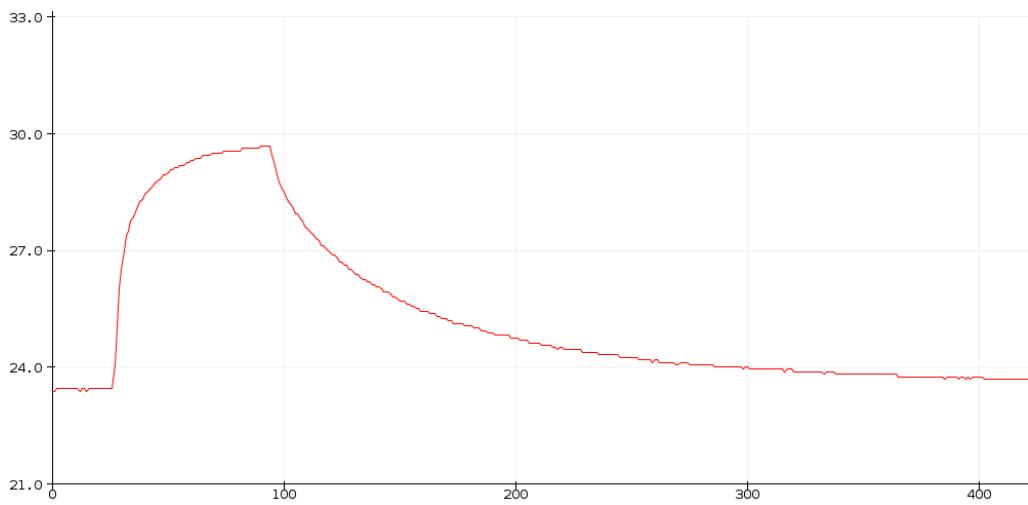


Figure 34. Temperature Sensor Reading Test

5.2.9.2 AD7995 ADC

The library `<adc_i2c.h>` was created to handle all I²C interfacing with the AD7995. It allows the user to specify an AD7995 device with its address. There is a function that allows the user to read a specified ADC channel as an integer. Below, a test program outputs all three ADC channels as separate colors on the serial plotter. A potentiometer is simultaneously wired to all ADC pins, with phase offsets in the signal attributing to speed of the acquisition of each channel. In this case, there was also an unavoidable error in the serial plot at the beginning of data display, resulting in the vertical blue lines.

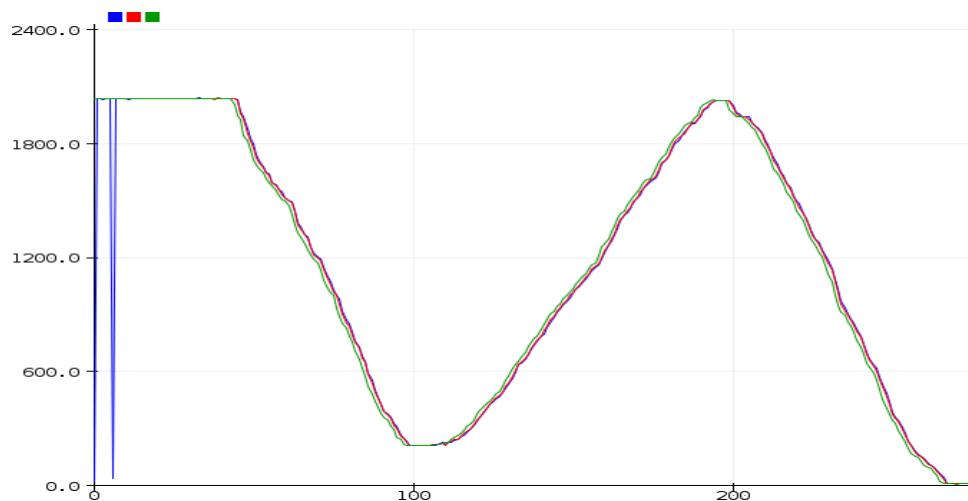


Figure 35. Potentiometer with ADC Test Output

5.2.9.3 EK1940 Capacitive Moisture Sensor

The EK1940 outputs an analog signal based on the moisture content of the soil it is in. In the plot below, is measured values from the sensor through the AD7995. It begins dry and is inserted into a pot of soil. Water is then added to the soil, resulting in a large spike in voltage

which eventually levels out as moisture distributes. The red line represents the max value of the ADC (5V) and the blue represents ground.

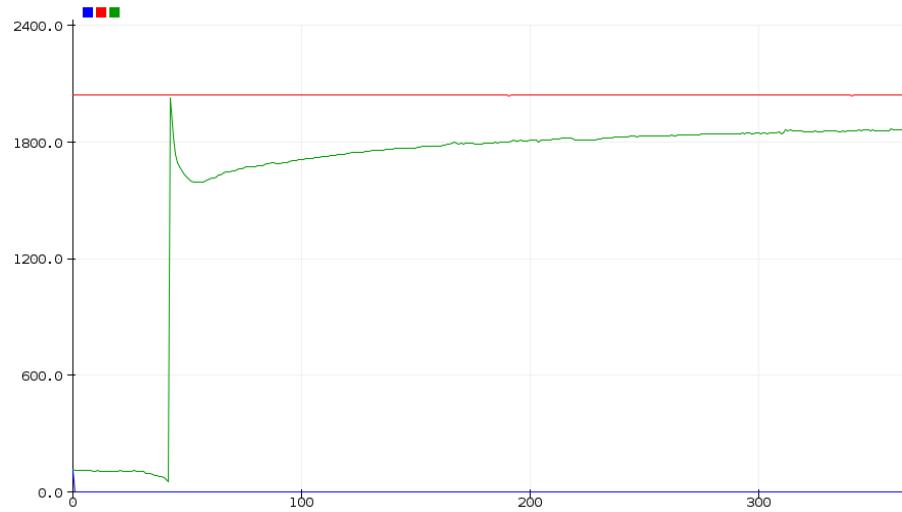


Figure 36. Moisture Sensor Response Test

5.2.9.4 LS7184N Quadrature Clock Converter

The library `<quad_enc.h>` provides useful functions for the processing of the LS7184N signals. For every complete tick of the quadrature encoder, the LS7184 will output a pulse as well as the direction of that tick as a binary value. In software, this pulse is registered by a pin change interrupt and a counter is incremented in the direction of that tick. Angular position of the motor can then be determined by dividing the number of counts by the resolution of the encoder. In the case below, a 12V DC motor with a built-in quadrature encoder was interfaced with the LS7184 and Songhe BTS7960 Driver. Therefore, a simple proportional controller was created to drive the motor to 1, 3, -2, and 0 rotations where one rotation was 4096 counts.



Figure 37. Rotary Encoder Test Graph

5.2.9.5 HVEC Testing

As stated previously, a prebuilt ZVS driver was purchased to drive the flyback transformer. This driver was tested with a 12V car battery to ensure it was not current limited. This produces tremendously large (2cm) arcs. Assuming the standard breakdown voltage of air of 30kV/cm, this means the driver was able to produce around 60kV on the secondary transformer. This arc was incredibly destructive to any plant life in its way, fulfilling its requirement on parasite destruction.



Figure 38. HVEC eliminating an undesired plant

5.3 Software Design of the Robot

The primary data processing computer of the system utilizes an NVIDIA Jetson Nano embedded computer, which links to the Arduino microcontroller responsible for interfacing with system sensors and actuators. The development setup can be seen below in figure 39. The Jetson Nano itself runs on Linux Ubuntu 18.04 and communicates with the Arduino over a serial cable using the Robotic Operating System (ROS) application. For all computer vision applications, the open source OpenCV library is installed and configured in a virtual environment for implementation in the Python scripting language. Additional libraries installed for machine learning development include Python-compatible archives such as sci-Kit-learn, TensorFlow, and Keras.



Figure 39. NVIDIA Jetson and Display

5.3.1 Growing Zone Inspection

The Jetson Nano clock will be responsible for determining when the system visually inspects the growing zone. Twice per day, the Jetson will restart the ROS serial interface with the Arduino via a Python script. To direct the end effector to navigate over the entire growing zone, the Jetson will reference a pre-programmed list of task-space waypoints to physically maneuver to. Using the ROS serial node, the Jetson will send these coordinates to the Arduino board, which handles the motor drivers to reach the specified position. Upon reaching a waypoint, the Arduino will send a destination acknowledgement to the Jetson via the bi-direction ROS node. From here, the Jetson will receive moisture and temperature readings through ROS from the Arduino sensory interface, while simultaneously launching a separate Python script to start a video stream for the camera.

A flowchart of the above procedure can be found below in Figure 40.

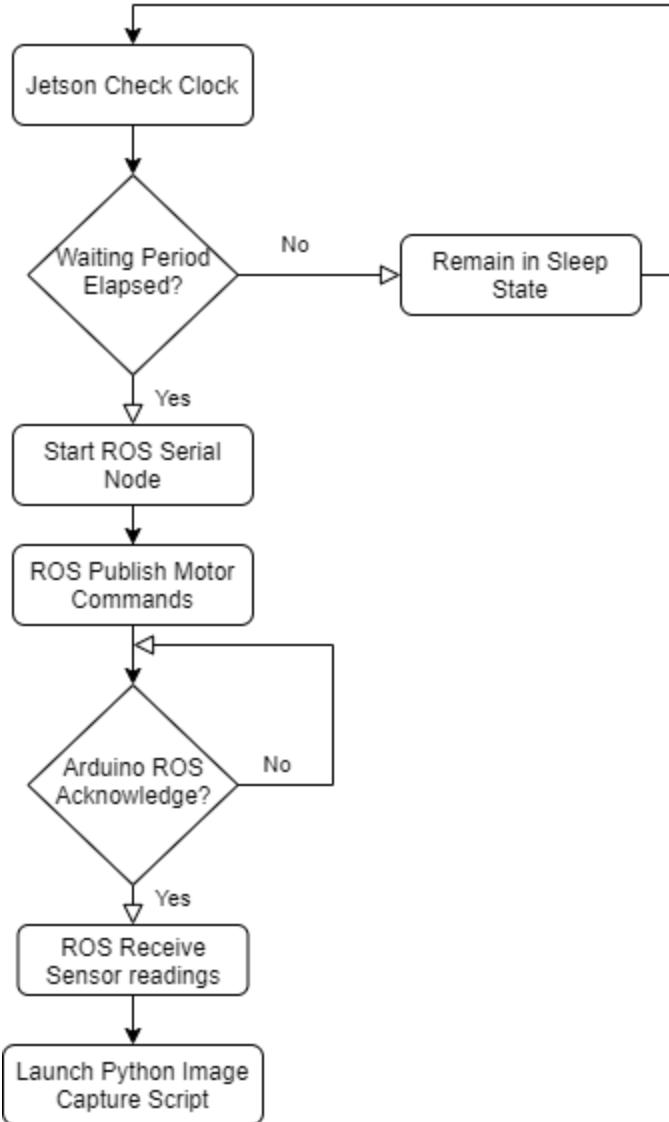


Figure 40. Growing Zone Inspection Flowchart

5.3.2 Computer Vision Implementation

The Jetson Nano is responsible for all image processing. To care for crops based on individual needs, while also eliminating all visible weeds, the system must be able to find and identify all plants within the growing zone. Upon capturing a still image from the camera video feed, the system uses a Python script to pre-process the image and isolate Regions of Interest (ROI). Feature extraction preparation consists of four major steps: Gaussian blurring, RGB-to-HSV conversion, color masking, and contour detection. The first stage, Gaussian blurring, is performed to counteract the data noise and background of the image itself and get rid of very small contours.



Figure 41. Original Image Versus Gaussian Blur

From there, the image format is transferred from Red-Green-Blue to Hue-Saturation-Value representation, where hue describes the color category, saturation defines how vivid the color is, and value correlates to the brightness. This will allow the vision system to properly identify colors even in the presence of shadows and varied lighting conditions.

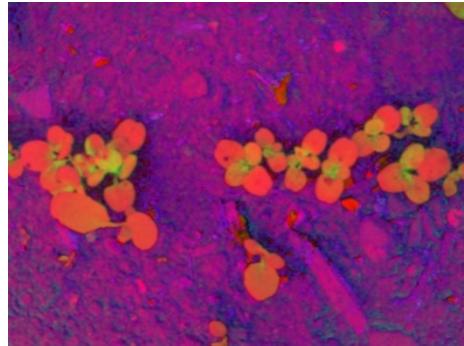


Figure 42. Hue-Saturation-Value Color Space Image

To isolate foliage within the image, the Python script then applies a color mask with ranges that have been determined heuristically. The result of masking is a binary image containing only black and white, where white segments correspond to colors within the desired range. This binary image is passed into a contour detection algorithm that groups the blobs into individual objects called contours. These contours define the regions where plant foliage has been detected and is ready for classification by the machine learning inference.



Figure 43. Color Masked Image

To allow the system to categorize individual plants, the list of contours is iterated through, and each contour is stored as a separate image in a Python list. This is the list to be passed into the texture analysis function.

5.3.3 Machine Learning Identification

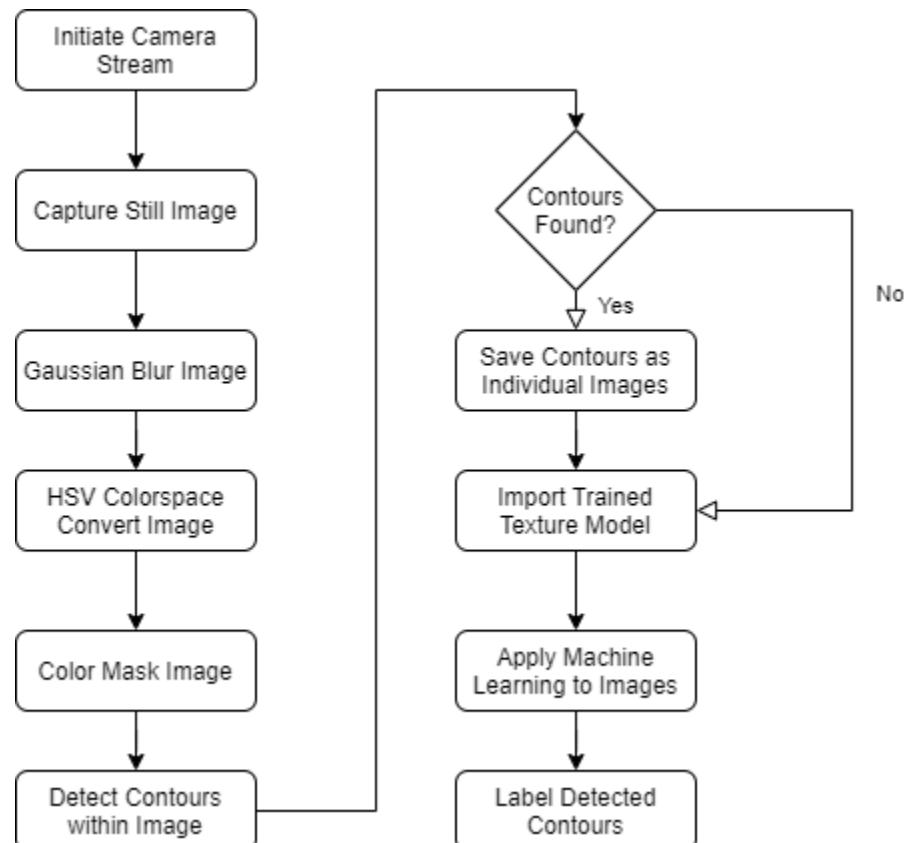


Figure 44. Image Processing Algorithm

To analyze a plethora of real-life plants, the system must first be trained to recognize the texture patterns crop foliage. This will enable it to distinguish leaves from foreign green items and differentiate crops from weed structures. Using the Python Sci-Kit-learn library, a separate Python script must generate mathematical models for the categories of texture. For proof-of-concept purposes, the system was initially trained to distinguish five arbitrary textures: Mache sprouts, grass, brick wall, pebbles, and fabric. To describe textures, the Local Binary Patterns algorithm searches for local pixel patterns within a grayscale image and saves a generalized model for each labelled category. After being trained on a set of known images, the model is saved as an external file and imported into the image processing script. It is here that the system applies the machine learning texture model to each image in the list of contour images, and classifies each contour based on the categories it was trained against. Figure 45 below displays the pre-trained classification algorithm being applied to a stock image of Mache plants.

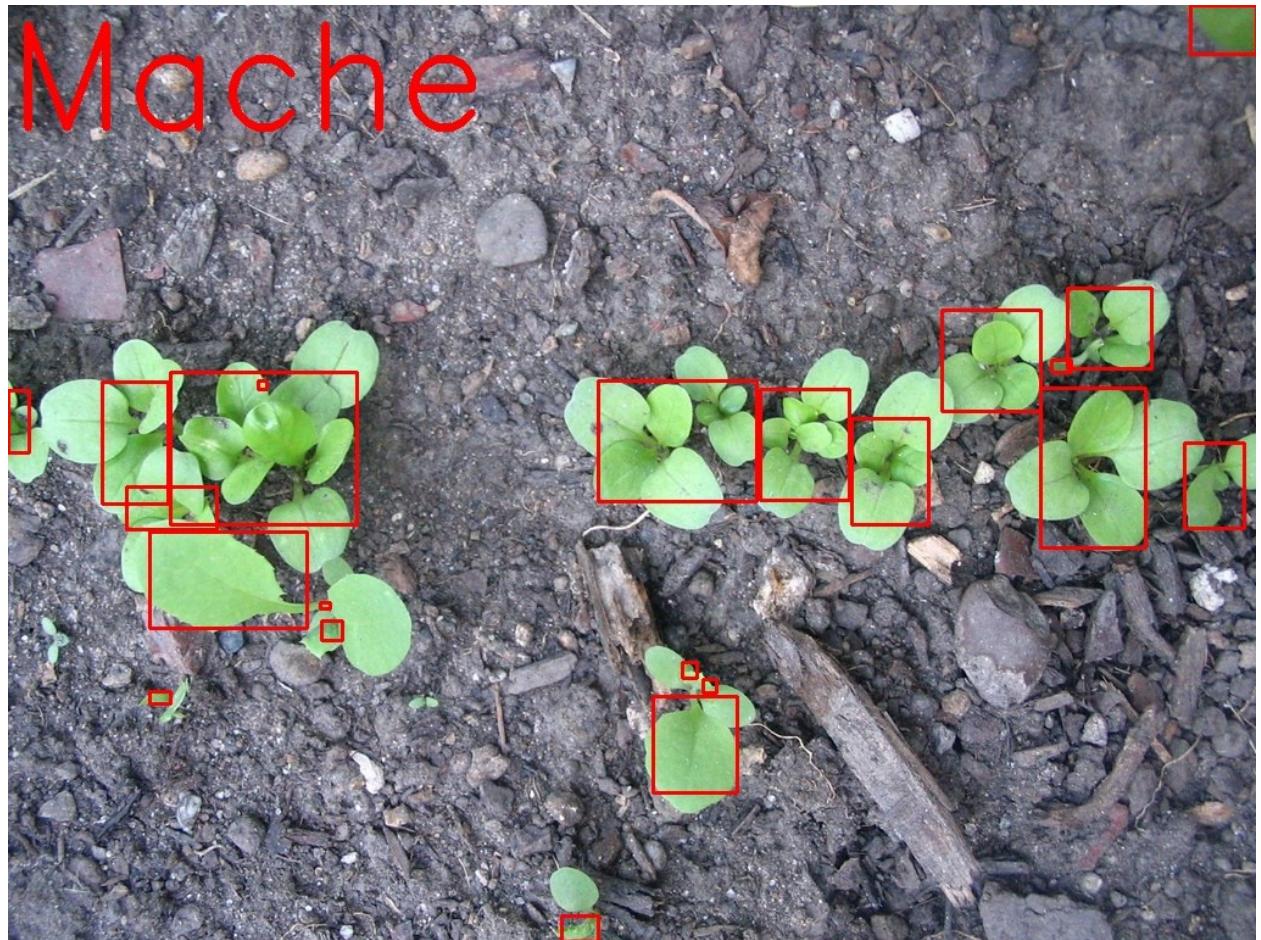


Figure 45. Machine Learning Correctly Identifying Mache

As of right now, the algorithm has only been exposed to five images of each texture category and is therefore very crude. However, even in its infancy the analysis can locate and recognize the major plant structures of an arbitrary image input. Upon completion of machine learning inference at a particular location, the Jetson sends the coordinates for the next waypoint through ROS to the Arduino to repeat the inspection process. Once all waypoints have been visited, the Jetson Nano terminates the ROS serial node, ends the camera interface script, and returns to a computational sleep state.

5.4 Kinematics of the Robot

Below one can find the kinematic equations, as well as the equations of motion, for the system.

5.4.1 Forward Kinematic Equations

The generalized coordinates that are used to calculate the forward kinematics are as follows:

$$\gamma = [\theta_1 \quad d_2 \quad d_3 \quad \theta_4]^T$$

θ_1 and θ_4 , the rotation of link one and the rotation of the wheels respectively, have the following kinematic relationship:

$$\theta_4 = \theta_1 \frac{r_L}{r_W} \frac{N_2}{N_1}$$

Table 21. Coordinate Descriptions

Coordinate	Description
θ_1	Gantry (link one) rotation
d_2	Trolley (link two) displacement
d_3	End effector (link three) displacement
θ_4	Rotational Drive wheel rotation

Below are the vectors that describe the position of the next frame relative to the current frame.

$$\underline{^I R_1} = \begin{bmatrix} 0 \\ 0 \\ 1425e - 3 \end{bmatrix} (m)$$

$$\underline{^1 R_{WheelA}} = \begin{bmatrix} 143.305826e - 3 \\ 1095.9432e - 3 \\ -944.561165e - 3 \end{bmatrix} (m)$$

$$\underline{^1 R_{WheelB}} = \begin{bmatrix} -143.305826e - 3 \\ 1095.9432e - 3 \\ -944.561165e - 3 \end{bmatrix} (m)$$

$$\underline{^1 R_2} = \begin{bmatrix} 0 \\ 227.5e - 3 + d_2 \\ 0 \end{bmatrix} (m)$$

$$\underline{^2 R_3} = \begin{bmatrix} 0 \\ 0 \\ -577.5e - 3 - d_3 \end{bmatrix} (m)$$

Below are the forward kinematic equations that describe the position of the end effector components relative to Frame 3.

$$\underline{^3 R_{Nozzle}} = \begin{bmatrix} -40e - 3 \\ 0 \\ -30e - 3 \end{bmatrix} (m)$$

$$\underline{^3 R_{HVEC}} = \begin{bmatrix} 0 \\ 0 \\ -65 \end{bmatrix} (m)$$

$$\underline{^3 R_{Temp}} = \begin{bmatrix} -67.5e - 3 \\ 25e - 3 \\ -80e - 3 \end{bmatrix} (m)$$

$$\underline{\underline{^3R_{Camera}}} = \begin{bmatrix} 110.35e - 3 \\ 0 \\ 25e - 3 \end{bmatrix} (m)$$

Below are the transformation matrices that describe the orientation of the next frame relative to the current frame.

$$T_1^I = \begin{bmatrix} \cos \theta_1 & -\sin \theta_1 & 0 \\ \sin \theta_1 & \cos \theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T_2^1 = I_3$$

$$T_3^2 = I_3$$

$$T_{Wheel}^1 = \begin{bmatrix} \cos \theta_1 & 0 & \sin \theta_1 \\ 0 & 1 & 0 \\ -\sin \theta_1 & 0 & \cos \theta_1 \end{bmatrix}$$

Below are the transformation matrices that describe the orientation of the current frame relative to the inertial frame.

$$T_2^I = T_1^I T_2^1$$

$$T_3^I = T_2^I T_3^2$$

Below are the forward kinematic equations that describe the position of each frame relative to the inertial frame.

$$\underline{\underline{^I R_2}} = \underline{\underline{^I R_1}} + T_1^I \begin{bmatrix} 0 \\ 227.5e - 3 + d_2 \\ 0 \end{bmatrix} (m)$$

$$\underline{\underline{^I R_3}} = \underline{\underline{^I R_2}} + T_2^I \begin{bmatrix} 0 \\ 0 \\ -577.5e - 3 - d_3 \end{bmatrix} (m)$$

$$\underline{\underline{^I R_{Moist}}} = \underline{\underline{^I R_3}} + T_3^I \begin{bmatrix} 67.5e - 3 \\ 17.5e - 3 \\ -80e - 3 \end{bmatrix} (m)$$

$$\underline{\underline{^I R_{HVEC}}} = \underline{\underline{^I R_3}} + T_3^I \begin{bmatrix} 0 \\ 0 \\ -65e - 3 \end{bmatrix} (m)$$

$$\underline{\underline{^I R_{Temp}}} = \underline{\underline{^I R_3}} + T_3^I \begin{bmatrix} -67.5e - 3 \\ 17.5e - 3 \\ -80e - 3 \end{bmatrix} (m)$$

$$\underline{\underline{I}R_{Nozzle}} = \underline{\underline{I}R_3} + T_3^I \begin{bmatrix} -40e - 3 \\ 0 \\ 30e - 3 \end{bmatrix} (m)$$

$$\underline{\underline{I}R_{Camera}} = \underline{\underline{I}R_3} + T_3^I \begin{bmatrix} 110.35e - 3 \\ 0 \\ 25e - 3 \end{bmatrix} (m)$$

$$\underline{\underline{I}R_{WheelA}} = \underline{\underline{I}R_{WheelA}} + T_1^I \underline{\underline{I}R_{WheelA}}$$

$$\underline{\underline{I}R_{WheelB}} = \underline{\underline{I}R_{WheelB}} + T_1^I \underline{\underline{I}R_{WheelB}}$$

5.4.2 Velocity Kinematic Equations

The velocity kinematics do not change much between frames. The angular velocity and the rate of change of the orientation of the first frame, second frame, and third frame are all equal. The first frame has zero translational velocity. The only difference between the second and third frame is a component appearing in the z position.

Below is the angular velocity of Frame 3 relative to the inertial frame measured in Frame 3.

$$\underline{\underline{^3\omega_I}} = \begin{bmatrix} 0 \\ 0 \\ RW\dot{\theta}_4 \cos\left(\frac{RW\theta_4}{RL}\right)^2 + \frac{RW\dot{\theta}_4 \sin\left(\frac{RW\theta_4}{RL}\right)^2}{RL} \end{bmatrix}$$

Below is the vector that describes the velocity of Frame 3 relative to the inertial frame.

$$\underline{\dot{r}_3} = \begin{bmatrix} -\dot{d}_2 \sin\left(\frac{RW\theta_4}{RL}\right) - \frac{RW\dot{\theta}_4 \cos\left(\frac{RW\theta_4}{RL}\right)(d_2 + 0.23)}{RL} \\ -\dot{d}_2 \cos\left(\frac{RW\theta_4}{RL}\right) - \frac{RW\dot{\theta}_4 \sin\left(\frac{RW\theta_4}{RL}\right)(d_2 + 0.23)}{RL} \\ \dot{d}_3 \end{bmatrix}$$

Below is the matrix that describes the rate of change of the orientation

$$\dot{T}_3^I = \begin{bmatrix} \frac{RW\dot{\theta}_4 \sin\left(\frac{RW\theta_4}{RL}\right)}{RL} & -\frac{RW\dot{\theta}_4 \cos\left(\frac{RW\theta_4}{RL}\right)}{RL} & 0 \\ \frac{RW\dot{\theta}_4 \cos\left(\frac{RW\theta_4}{RL}\right)}{RL} & \frac{RW\dot{\theta}_4 \sin\left(\frac{RW\theta_4}{RL}\right)}{RL} & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Below is the angular velocity of the wheel frames relative to the inertial frame measured in the wheel frame.

$${}^w_w\omega_I = \begin{bmatrix} -\frac{RW\dot{\theta}_4 \sin(\theta_4)}{RL} \\ \dot{\theta}_4 \\ -\frac{RW\dot{\theta}_4 \cos(\theta_4)}{RL} \end{bmatrix}$$

Below is the vector that describes the velocity of the wheels relative to the inertial frame.

$$\begin{aligned} {}^I\dot{r}_{WA} &= \begin{bmatrix} -\frac{RW\dot{\theta}_4 (1.0959) \cos\left(\frac{RW\theta_4}{RL}\right)}{RL} + \frac{(0.1433)RW \sin\left(\frac{RW\theta_4}{RL}\right)}{RL} \\ \frac{RW\dot{\theta}_4 (0.1433) \cos\left(\frac{RW\theta_4}{RL}\right)}{RL} - \frac{(1.0959)RW \sin\left(\frac{RW\theta_4}{RL}\right)}{RL} \\ 0 \end{bmatrix} \\ {}^I\dot{r}_{WB} &= \begin{bmatrix} -\frac{RW\dot{\theta}_4 (1.0959) \cos\left(\frac{RW\theta_4}{RL}\right)}{RL} - \frac{(0.1433)RW \sin\left(\frac{RW\theta_4}{RL}\right)}{RL} \\ \frac{RW\dot{\theta}_4 (0.1433) \cos\left(\frac{RW\theta_4}{RL}\right)}{RL} + \frac{(1.0959)RW \sin\left(\frac{RW\theta_4}{RL}\right)}{RL} \\ 0 \end{bmatrix} \end{aligned}$$

Below is the matrix that describes the rate of change of the orientation

$$\begin{aligned} \dot{T}_w^I &= \begin{bmatrix} -\frac{\dot{\theta}_4 \cos\left(\frac{RW\theta_4}{RL}\right)}{RL} \sin(\theta_4) + \frac{RW \sin\left(\frac{RW\theta_4}{RL}\right) \cos(\theta_4)}{RL} & -\frac{RW\dot{\theta}_4 \cos\left(\frac{RW\theta_4}{RL}\right)}{RL} & \dot{\theta}_4 \cos\left(\frac{RW\theta_4}{RL}\right) \cos(\theta_4) - \frac{RW \sin\left(\frac{RW\theta_4}{RL}\right) \sin(\theta_4)}{RL} \\ -\frac{\dot{\theta}_4 \sin\left(\frac{RW\theta_4}{RL}\right)}{RL} \sin(\theta_4) - \frac{RW \cos\left(\frac{RW\theta_4}{RL}\right) \cos(\theta_4)}{RL} & -\frac{RW\dot{\theta}_4 \sin\left(\frac{RW\theta_4}{RL}\right)}{RL} & 0 \\ -\dot{\theta}_4 \cos(\theta_4) & 0 & -\dot{\theta}_4 \sin(\theta_4) \end{bmatrix} \end{aligned}$$

5.4.3 Inverse Kinematic Solutions

Listed below are the derivations for the inverse kinematic solutions for each component of the end effector.

5.4.3.1 Temperature Sensor Solution

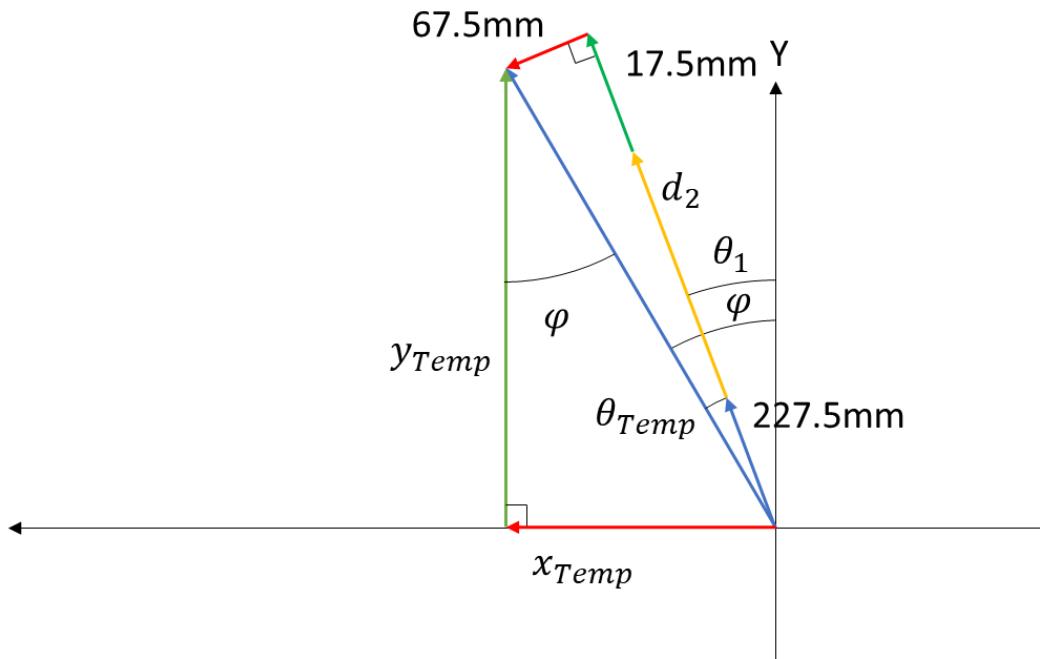


Figure 46. Temperature Sensor Inverse Kinematics

$$\varphi = -\text{atan}2(x_{Temp}, y_{Temp})$$

$$\theta_{Temp} = \arcsin \left(\frac{67.5e - 3m}{\sqrt{x_{Temp}^2 + y_{Temp}^2}} \right)$$

$$\theta_1 = \varphi - \theta_{Temp}$$

$$d_2 = \left(\sqrt{x_{Temp}^2 + y_{Temp}^2} \right) \cos \theta_{Temp} - 252.5e - 3m$$

$$d_3 = 767.5e - 3m - z_{Temp}$$

5.4.3.2 Moisture Sensor Solution

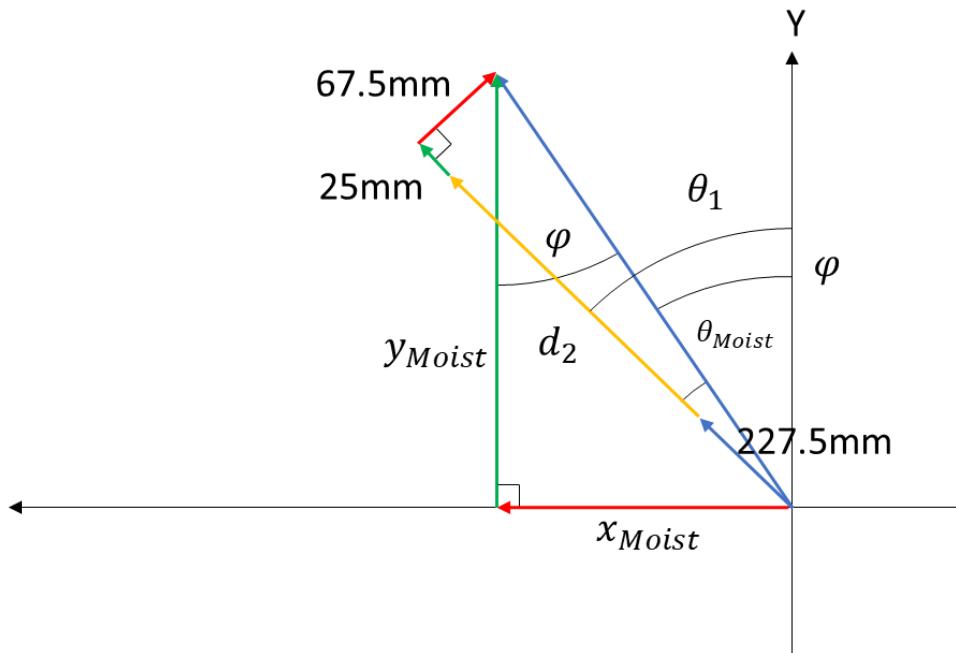


Figure 47. Moisture Sensor Inverse Kinematics

$$\varphi = -\text{atan}2(x_{Moist}, y_{Moist})$$

$$\theta_{Moist} = \arcsin \left(\frac{67.5e - 3m}{\sqrt{x_{Moist}^2 + y_{Moist}^2}} \right)$$

$$\theta_1 = \varphi + \theta_{Temp}$$

$$d_2 = \left(\sqrt{x_{Moist}^2 + y_{Moist}^2} \right) \cos \theta_{Moist} - 245e - 3m$$

$$d_3 = 767.5e - 3m - z_{Moist}$$

5.4.3.3 Nozzle Solution

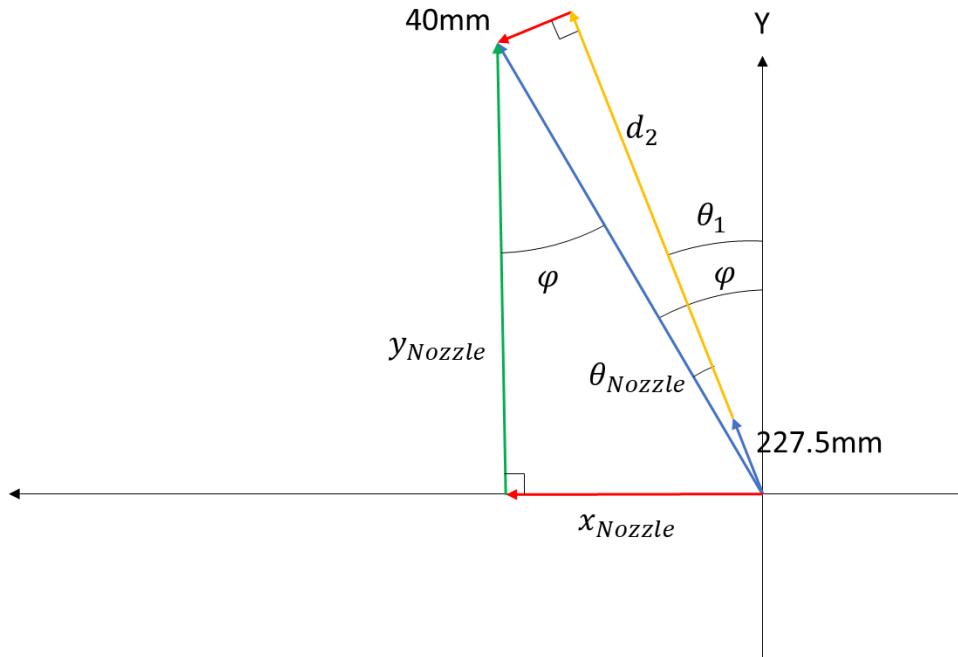


Figure 48. Nozzle Inverse Kinematics

$$\varphi = -\text{atan}2(x_{\text{Nozzle}}, y_{\text{Nozzle}})$$

$$\theta_{\text{Nozzle}} = \arcsin\left(\frac{40e - 3m}{\sqrt{x_{\text{Nozzle}}^2 + y_{\text{Nozzle}}^2}}\right)$$

$$\theta_1 = \varphi - \theta_{\text{Nozzle}}$$

$$d_2 = \left(\sqrt{x_{\text{Nozzle}}^2 + y_{\text{Nozzle}}^2} \right) \cos \theta_{\text{Nozzle}} - 227.5e - 3m$$

$$d_3 = 817.3e - 3m - z_{\text{Nozzle}}$$

5.4.3.4 Camera Solution

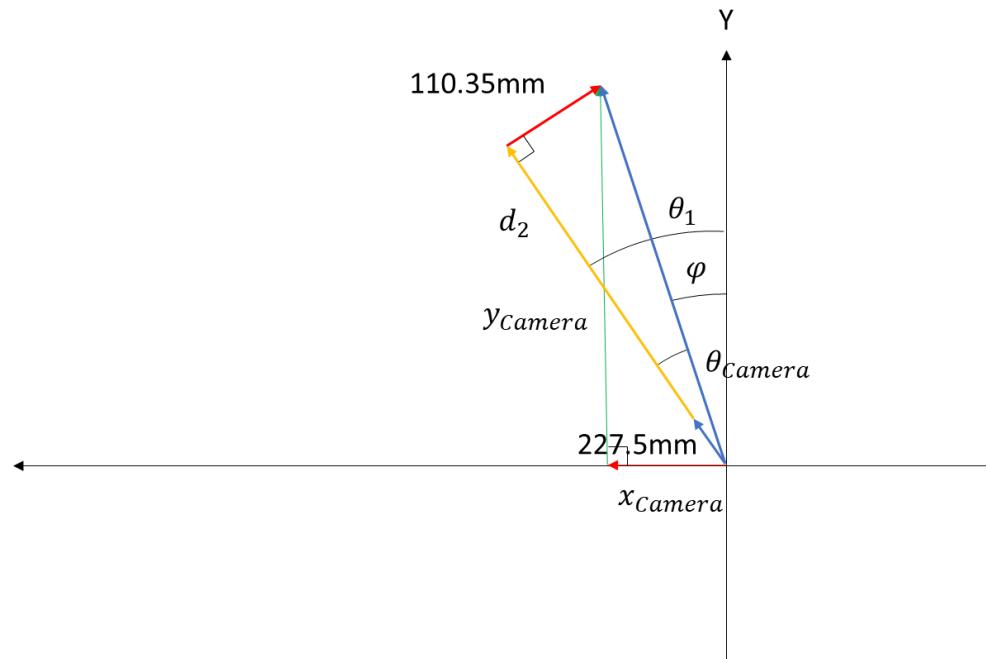


Figure 49. Camera Inverse Kinematics

$$\varphi = -\text{atan}2(x_{\text{Camera}}, y_{\text{Camera}})$$

$$\theta_{\text{Camera}} = \arcsin\left(\frac{110.35e - 3m}{\sqrt{x_{\text{Camera}}^2 + y_{\text{Camera}}^2}}\right)$$

$$\theta_1 = \varphi + \theta_{\text{Camera}}$$

$$d_2 = \left(\sqrt{x_{\text{Camera}}^2 + y_{\text{Camera}}^2} \right) \cos \theta_{\text{Camera}} - 227.5e - 3m$$

$$d_3 = 872.5e - 3m - z_{\text{Camera}}$$

5.4.3.5 HVEC Solution

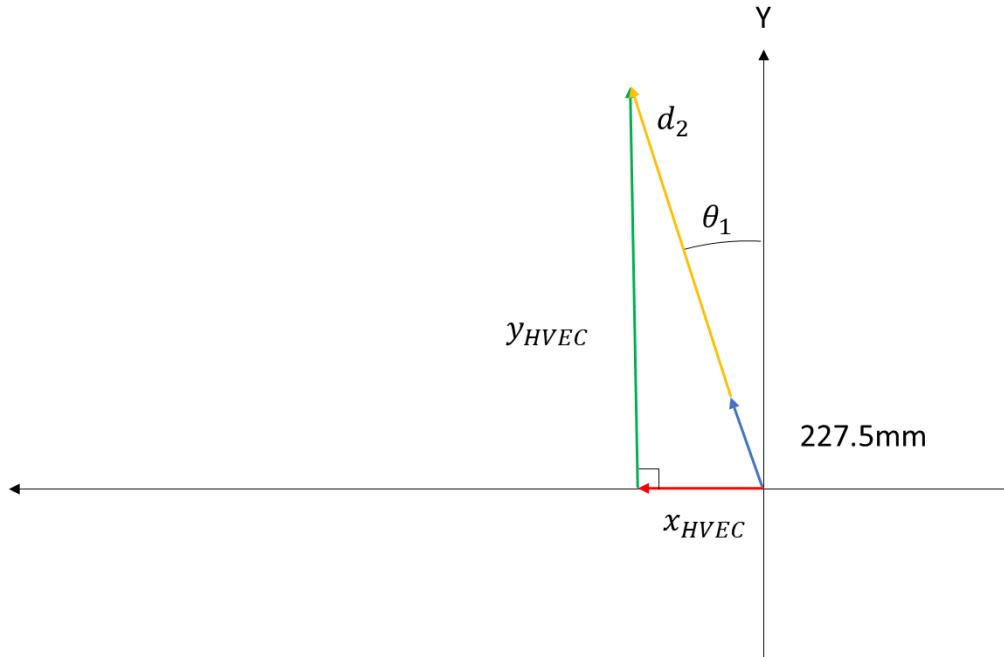


Figure 50. HVEC Inverse Kinematics

$$\theta_1 = -\text{atan2}(x_{HVEC}, y_{HVEC})$$

$$d_2 = \left(\sqrt{x_{HVEC}^2 + y_{HVEC}^2} \right) - 227.5e - 3m$$

$$d_3 = 782.5e - 3m - z_{HVEC}$$

5.5 Equations of Motion of the Robot

The equations below were derived using the Newton-Euler method

$$H(\gamma)\ddot{\gamma} + d(\gamma)\dot{\gamma} + G(\gamma) = F_\gamma$$

The system mass matrix, $H(\gamma)$, is listed below. Large constants and expressions were simplified to single variables to increase readability.

$$H(\gamma) = \begin{bmatrix} m_1 + m_2 & 0 & \frac{rW(Gam2_x + Gam3_x)}{rL} \\ 0 & m_3 & 0 \\ \frac{rW(Gam2_x + Gam3_x)}{rL} & 0 & H_{33} \end{bmatrix}$$

$$H_{33} = f_1 + (\sin(\theta_4)f_2 - \cos(\theta_4)f_1)RL \cdot RW + mWx_6RW^2 + k_5f_3 + GamWy \cdot k_2 + k_1f_4 + k_3f_4 + k_4f_5$$

$$\begin{aligned}
f_1 &= (JWyy \cdot x_4 \cdot RL^2) \\
f_2 &= (GamWx \cdot x_2 - JWxy \cdot x_7) \\
f_3 &= (GamWz \cdot x_2 - JWxy \cdot x_7) \\
f_4 &= (m_2 + m_3) \\
f_5 &= (Gam2y + Gam3y + d_2m_2 + d_2m_3) \\
f_6 &= (J1zz + J2zz + J3zz + d_2^2m_2 + d_2^2m_3) \\
f_7 &= ((JWzz - JWxx)(\cos(\theta_4))^2 + JWxx + Gam2y \cdot d_2 + Gam3y \cdot d_2 - JWxz \sin(2\theta_4))
\end{aligned}$$

$$\begin{aligned}
k_1 &= RW^2x_1 \\
k_2 &= RW^2x_2 \\
k_3 &= RW^2x_3 \\
k_4 &= RW^2x_4 \\
k_5 &= RW^2x_5
\end{aligned}$$

$$\begin{aligned}
x_1 &= 2884285410699290357225448803192012800000 \\
x_2 &= 277854611059209293708063984844800000000000 \\
x_3 &= 63382530011411470074835160268800000000000 \\
x_4 &= 126765060022822940149670320537600000000000 \\
x_5 &= 328130729748204767489753183095139336192 \\
x_6 &= 15485975402083386940193234943429541015625 \\
x_7 &= 253530120045645880299340641075200000000000
\end{aligned}$$

The Coriolis and Centripetal vector, $d(\gamma)$, is listed below. Large constants and expressions were simplified to single variables to increase readability.

$$d(\gamma) = \begin{bmatrix} r_W^2 \dot{\theta}_4^2 (100000(Gam2_y + Gam3_y) + 22753(m_2 + m_3) + 100000(d_2m_2 + d_2m_3)) \\ 100000r_L^2 \\ 0 \\ d(3) \end{bmatrix}$$

$$d(3) = \frac{k_3(k_1x_1f_1 + d_2k_1x_1f_2 + k_2\cos(\theta_4)f_3 + k_2\sin(\theta_4)f_4 + k_1x_2f_2 - JWxz k_3 x_1 \cos(2\theta_4) + k_3 x_4 f_4 \sin(2\theta_4))}{RL^2 x_4}$$

$$\begin{aligned}
f_1 &= Gam2y + Gam3y \\
f_2 &= (m_2 + m_3) \\
f_3 &= (GamWx \cdot x_3 - JWxy \cdot x_1) \\
f_4 &= (GamWz \cdot x_3 - JWyz \cdot x_1) \\
f_5 &= (JWxx - JWzz)
\end{aligned}$$

$$\begin{aligned}
k_1 &= RW\dot{d}_2 \\
k_2 &= RL\dot{\theta}_4 \\
k_3 &= RW\dot{\theta}_4
\end{aligned}$$

$$\begin{aligned}
x_1 &= 14073748835532800000 \\
x_2 &= 3202200072548777984 \\
x_3 &= 14073748835532800000 \\
x_4 &= 7036874417766400000
\end{aligned}$$

Below is the vector for generalized gravitational forces.

$$G(\gamma) = \begin{bmatrix} 0 \\ -9.81m_3 \\ -20 \cdot GamWx \cos(\theta_4) - 20 \cdot GamWz \sin(\theta_4) \end{bmatrix}$$

5.5.1 Dynamical Simulations of the Open-Loop Robot

The open-loop dynamical simulation shows the response of the robot to generalized forces. The simulation that produced the graph below had basic joint limits implemented, the generalized gravitational forces, and a torque applied to the drive wheels. Springs were used to model the joint limits to more accurately model collisions without needing momentum equations.

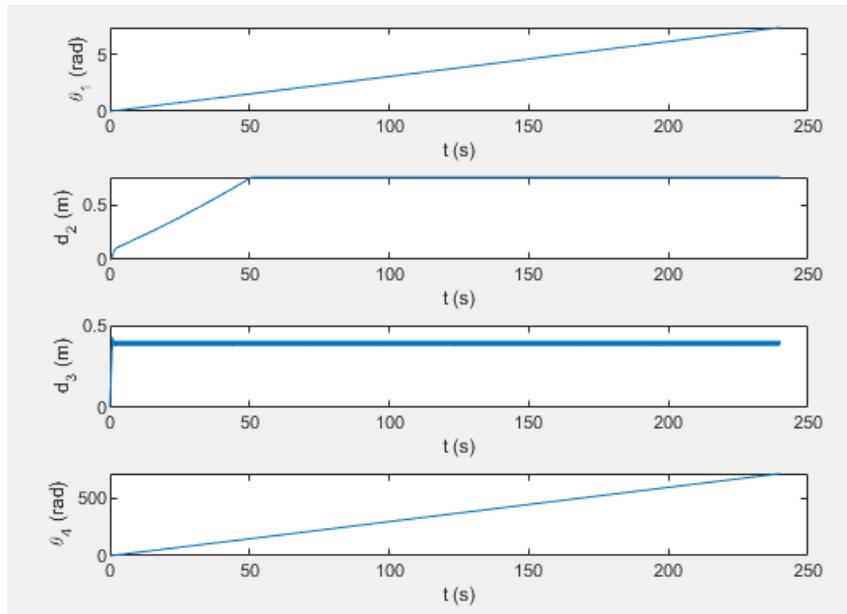


Figure 51. Open-Loop Dynamic Response

5.6 Control System for the Robot

The control system was implemented using a PID controller. This ensured reasonable response times with little overshoot and steady state error. Desired joint variables are calculated using the robot's inverse kinematic functions and fed into the control rule. To minimize the amount that the end effector arm brushes against potential plants, the following control scheme was implemented:

- 1) The end effector is retracted to its zeroed position.
- 2) The trolley is driven to its zeroed position
- 3) The wheel driven to rotate the gantry arm into its desired position.
- 4) The trolley is driven to its desired position.
- 5) The end effector is lowered to its desired position.

Motor voltages (trolley, vertical, wheel respectively) use the following PID gains to implement control. These gains were determined through trial and error.

$$K_p = [5 \quad 40 \quad 40]$$

$$K_d = [7 \quad 5 \quad 1]$$

$$K_i = [1 \quad 1 \quad 1]$$

5.6.1 Dynamical Simulations of the Closed-Loop Robot

Observing the joint variables of the system, the following graphs demonstrate the system moving to a desired configuration. The red dashed line indicates the desired joint angle.

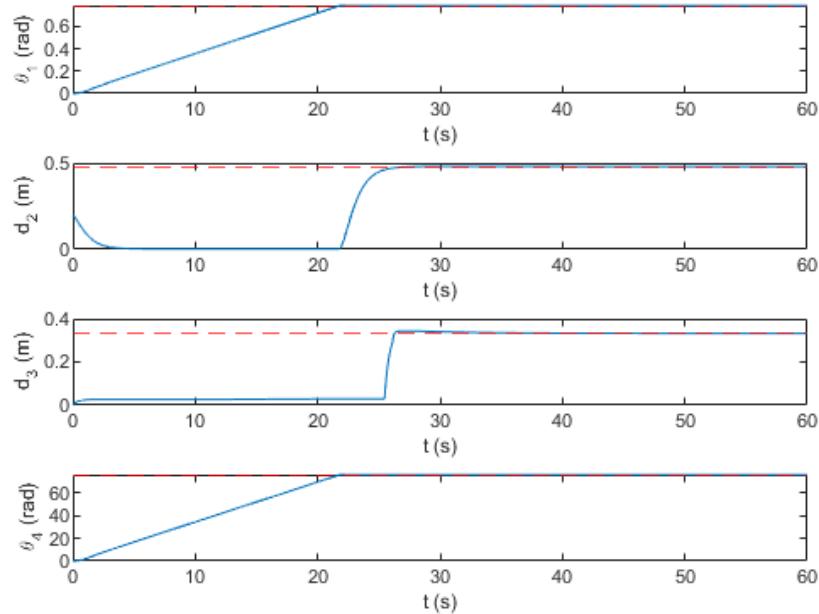


Figure 52. Closed-Loop Dynamic Response

The voltage inputs into the system were converted PWM signals with duty cycles. This was because all drivers for the system require a control PWM signal. Several limitations apply. For example, the vertical link, d_3 is driven by a stepper motor on the system. Due to the difficulty of modeling steppers, this link was replaced by a DC motor. This assumption is valid within the bounds of the system as it was proven that the stepper motor could provide enough torque to drive the end effector axis, see section 5.2.2.2.

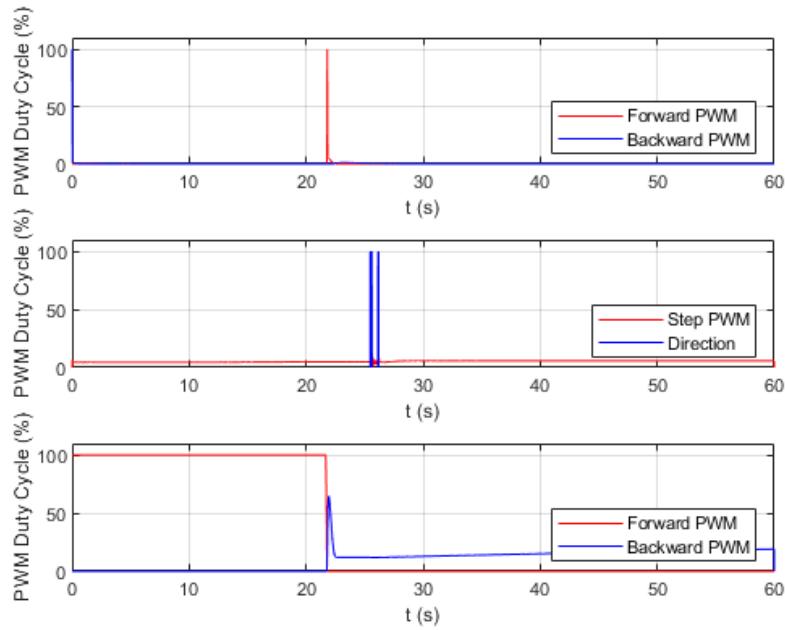


Figure 53. Stepper Motor Modeling

A simulation was also conducted involving multiple desired positions and the robot moving to each and tapping them with the end effector. In this test, the joint variables and their desired positions are the following.

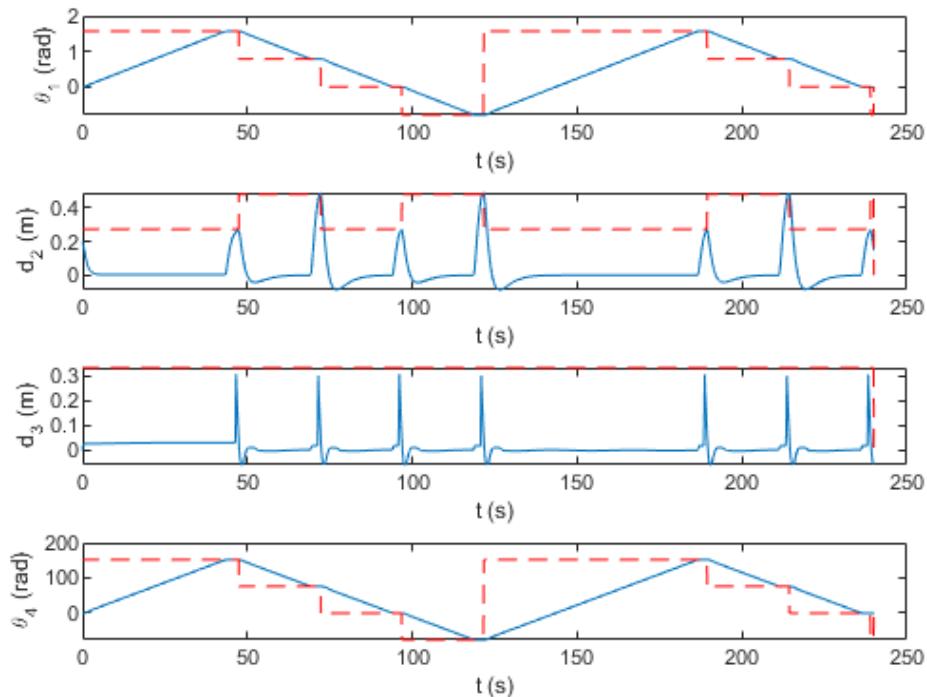


Figure 54. Simulating Simultaneous Joints

The motor driver inputs for this case are the following.

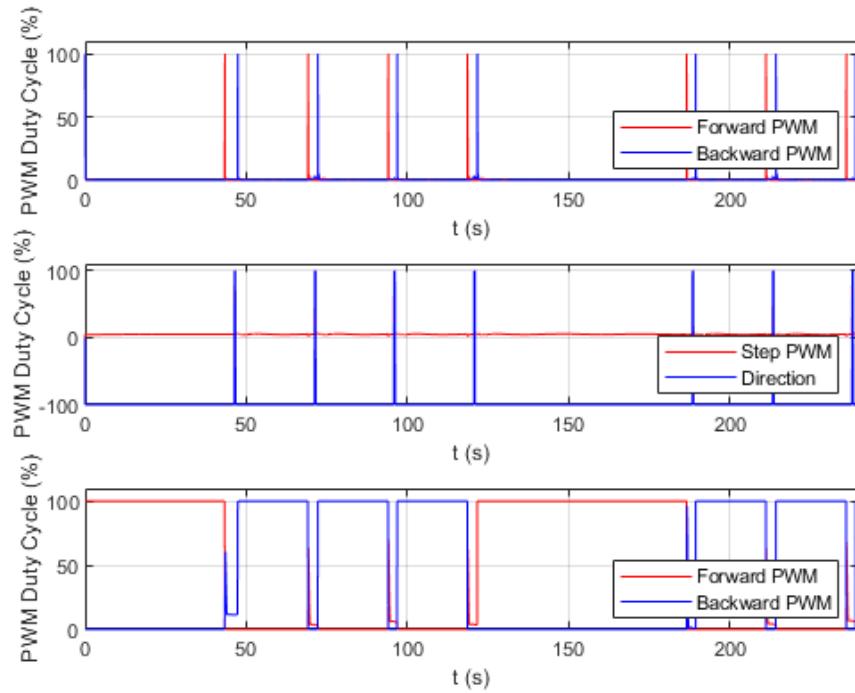


Figure 55. Motor Driver Inputs

5.6.2 Stability Analysis

A stability analysis of the robotic system was attempted using Lyapunov's first method. Unfortunately, all attempts to linearize the closed loop system resulted in a highly unstable system with right hand poles. This is in opposition to the actual nonlinear system, which proved to be stable for all cases it was tested in. More work will be done with the actual system next semester in determining stability. Due to the simplicity of the system, NILE is confident that its controller is stable based upon the simulation results and to be confirmed in future testing.

6.0 Project Management Review

As of December 10th, the NILE team has all CAD drawings completed with full mechanical buildout ready to begin fabrication next semester. Already this semester, the computer vision foundations have been laid and extremely successful using images of Mache grown on campus. Circuit designs have been bench-tested and PCBs are ready to be manufactured and populated.

Given all this progress the NILE team should be ready to hit the ground running beginning January 12th and immediately begin buildout of the prototype stem. Our plans for the next semester are laid out below in the Gantt Chart, Table 3. Provided all fabrication goes as planned, the NILE team should have plenty of time to complete full mechanical and software testing plans before the end of the semester.

Table 195. Detail Design Gantt Chart

Week	1	2	3	4	5	6	7	8	9	10	11	12	13
Requisition													
Fabrication													
Mech Assembly													
Electric assembly													
Software integration													
Stress testing													
Requirement testing													
Endurance testing													
Edge case testing													
Paper writing													
Presentation													

6.1 Budget

Given the high level of detail in the NILE mechanical and electrical design we have a complete, accurate budget for all commercial off the shelf components [COTS]. \$704. However, that COTS value does not include the 50 custom machined and 3D printed components, 2 custom PCBs, and the numerous fasteners and consumables. Next semester we will gain a deeper understanding of these cost areas.

7.0 Conclusion

Overall, as the semester and preliminary design concludes, NILE is in a hopeful position as we continue into the critical design phase and fabrication commences. The final design presented in this document successfully meets all the system requirements and specifications.

Designed with budget and scalability in mind, it represents a mechanically simple solution for tending to a variety of plants and growing zone configurations.

Combining a semi-mobile gantry with advanced image processing techniques, our system will be able to inspect and tend to plants during different stages of growth and apply nutrients and hydration to specific areas as needed. With this emphasis on precision of analysis and application, the design is on track to exceed our requirement of water application efficiency greater than 90%. As noted in the Project Management Review section, NILE has already begun growing Mache and training our unique machine learning algorithm to identify precise locations of the plant leaves.

To conclude, NILE is excited to be tackling the issues surrounding conventional agricultural methods with such cutting-edge technology and modern solutions. The overcompensation of plant and soil needs through wasteful watering practices and excessive application of pesticides and fertilizers, leading to substantial environmental damage, cannot continue much longer if we are to have a sustainable farming industry. NILE is thrilled to provide a sustainable solution to change the wasteful ways of farming.

8.0 Citations

- [1] A. Mizuno, T. Tenma and N. Yamano, "Destruction of weeds by pulsed high voltage discharges," Conference Record of the 1990 IEEE Industry Applications Society Annual Meeting, 1990, pp. 720-727 vol.1, doi: 10.1109/IAS.1990.152264.
- [2] J. Schalau "The Soils and Climate of Yavapai County," University of Arizona Cooperative Extension, [Online], Available: <https://cals.arizona.edu/yavapai/anr/hort/mastergardener/mgcourseresources/soilsandclimateofyavapaico.pdf> [Accessed Oct. 21, 2021]
- [3] MartyG, "Achieving high accuracy with L515 or D415," [Forum] Available: <https://support.intelrealsense.com/hc/en-us/community/posts/360049091914-Achieving-high-accuracy-with-L515-or-D415> [Accessed: Oct. 18, 2021].
- [4] S. Christman, "Valerianella Locusta," Floridata, Jan. 23, 2021, [Online], Available: <https://floridata.com/plant/732> [Accessed Oct. 21, 2021]
- [5] E. Gray, "Texas crop circles from space", NASA, [Online Article], 2012. Available: <https://climate.nasa.gov/news/729/texas-crop-circles-from-space/>. [Accessed September 24, 2021].
- [6] International Electrotechnical Commission. (2013). *IEC 60529, Edition 2.2*.
- [7] J. P. Gaus, "Robotic Gantry Bridge for Farming," US Patent 9,622,398 B2, 18 Apr., 2017
- [8] K. C. Bares, "Rowbot in the field," rowbot.com, Sept. 8, 2021. [Online]. Available: <https://www.rowbot.com/blog-posts/2014/9/8/rowbot-in-the-field> [Accessed Sept 22, 2021]
- [9] P. Mansson, "Low cost NDVI analysis using RaspberryPi and PiNoIR", Public Lab, 9, April 11.
- [10] R. Finnerty, "Robot farmers' pioneer climate-resilient farming in the North Country", North Country Public Radio, [Online Article], 2021 [Online]. Available: <https://www.northcountrypublicradio.org/news/story/44377/20210903/robot-farmers-pioneer-climate-resilient-farming-in-the-north-country> [Accessed Sept. 15, 2021].
- [11] S. Herrick, "NDVI vs. False NDVI: What's better for analyzing crop health?" Botlink, Nov. 30, 2017.
- [12] S. Ramesh et al., " Plant Disease Detection Using Machine Learning," 2018 International Conference on Design Innovations for 3Cs Compute Communicate Control (ICDI3C), 2018, pp. 41-45
- [13] T. Zafiroski, "How to Make a DIY Plasma Speak Using an Arduino Uno" Maker Pro, May 1, 2020. <https://maker.pro/arduino/tutorial/how-to-make-a-diy-plasma-speaker-using-an-arduino-uno>

- [14] "FarmBot | Open-Source CNC Farming," FarmBot. <https://farm.bot/>.
- [15] International Electrotechnical Commission. (2013). IEC 60529, Edition 2.2.
- [16] Budynas, R. G., Nisbett, J. K., & Shigley, J. E. (2011). Shigley's mechanical engineering design. New York: McGraw-Hill.
- [17] Pittman, "DC Gearmotor," GM9234S032-R1-SP datasheet.
<https://www.alliedelec.com/m/d/19ae6a6a5550fed97008a6e51e514219.pdf>
- [18] Hetai Motor, "Hybrid Stepper Motor," 42BYGHM Series datasheet. <https://www.promoco-motors.com/products/StepperMotors/42BYGHM%20Series.pdf>
- [19] CCL Industrial Motor Limited, FR801-001 datasheet.
<https://web.archive.org/web/20070203122606/http://www2.usfirst.org/2005comp/Specs/CIM.pdf>
- [20] Handson Technology, BTS7960 datasheet.
<https://www.handsontec.com/dataspecs/module/BTS7960%20Motor%20Driver.pdf>
- [21] STEPPERONLINE, "2-Phase Digital Stepper Drive," DM332 datasheet. <https://www.omc-stepperonline.com/download/DM332T.pdf>
- [22] Avago Technologies, "12-bit Angular Detection Device", AEAT-6012 datasheet.
<https://docs.broadcom.com/doc/AV02-0188EN>
- [23] Avago Technologies, "Quick Assembly Two Channel Optical Encoders," HEDS-5640 datasheet. <https://docs.broadcom.com/doc/AV02-1046EN>
- [24] Adafruit, "Plastic Water Solenoid Valve - 12V - 1/2" Nominal",
<https://www.adafruit.com/product/997>
- [25] Adafruit, "Liquid Flow Meter - Plastic 1/2" NPS Threaded",
<https://www.adafruit.com/product/828>
- [26] V. Mazzillim "Flyback Driver," schematic.
<https://content.instructables.com/ORIG/FIY/EAZN/G9NH8XOG/FIYEZNG9NH8XOG.png?auto=webp&frame=1&width=1024&fit=bounds&md=062c2b9b1bc4ad4673cbcd7a375527f9>
- [27] Microchip, "2-Wire Temperature Sensor," TCN75A datasheet.
<https://ww1.microchip.com/downloads/en/DeviceDoc/21935D.pdf>
- [28] STMicroelectronics, IRF540 datasheet.
<https://media.digikey.com/pdf/Data%20Sheets/ST%20Microelectronics%20PDFS/IRF540.pdf>

- [29] Analog Devices, "4-Channel, 12-/10-/8-Bit ADC with I₂C-Compatible Interface in 8-Lead SOT-23," AD7995 datasheet. https://www.analog.com/media/en/technical-documentation/data-sheets/AD7991_7995_7999.pdf
- [30] LSI Computer Systems, "Quadrature Clock Converter," LS7183N datasheet. https://lsicsi.com/datasheets/LS7183N_LS7184N.pdf
- [31] LogiSwitch, LS-KW10 Series Limit Switch datasheet. <https://www.logiswitch.com/wp-content/uploads/LS-KW10-Technical-Details.pdf>
- [32] DFRobot, "Capacitive Soil Moisture Sensor" datasheet. https://media.digikey.com/pdf/Data%20Sheets/DFRobot%20PDFs/SEN0193_Web.pdf

Appendix A – Bill of Materials

A.1 Custom Components

Part #	Description	Qty.
NILE_P1004	Base Frame Plate	1
NILE_P1005	Base Frame Vertical Support	1
NILE_P1006	Base Frame Central Post	1
NILE_P1007	Gantry Beam	1
NILE_P1008	Gantry Pivot Plate	2
NILE_P1009	Gantry Pivot Support	2
NILE_P1010	Rotational Joint Shaft	1
NILE_P1012	Gantry Drive Frame	1
NILE_P1013	Gantry Wheel Mount	4
NILE_P1014	CIM Motor Adapter Mount	1
NILE_P1015	Gantry Wheel Axel	2
NILE_P1016	Rotational Joint Encoder Mount	1
NILE_P1017	Gantry Drive Frame Mount, Flat	1
NILE_P1018	Gantry Drive Frame Mount, Angled	1
NILE_P1019	Gantry Drive Enclosure, Top	1
NILE_P1020	Gantry Drive Enclosure, Rear	1
NILE_P1021	Gantry Drive Frame Support	1
NILE_P1024	Rotational Joint Electrical Box, Bottom	1
NILE_P1025	Rotational Joint Electrical Box, Top	1
NILE_P1026	Gantry Drive Enclosure, Front	1
NILE_P2000	Trolley Base Plate	2
NILE_P2001	Idle Wheel Axel	1
NILE_P2002	Vertical Translation Frame	2
NILE_P2003	Lead Nut Mount	1
NILE_P2004	Driven Wheel Insert	4
NILE_P2005	Translational Motor Mount	1
NILE_P2006	Driven Wheel Axel	2
NILE_P2007	Vertical Translation Shaft Mount	1
NILE_P2008	Stepper Motor Mount	1
NILE_P2009	Trolley Enclosure	2
NILE_P2010	Flow Meter Mount	1
NILE_P2011	E-Chain Mounting Plate, Vertical Translation	2
NILE_P2012	E-Chain Mounting Plate, Horizontal Translation	1
NILE_P2013	Vertical Translation Switch Mount	1
NILE_P2014	Trolley Electrical Module Mount	1
NILE_P2015	HVEC Driver Mount	1

NILE_P2016	HVEC Coil Mount	1
NILE_P2017	Back Plate for HVEC Coil Mount	1
NILE_P3000	End Effector Mounting Platform	1
NILE_P3001	HVEC Spark Wire Mounting Arm	4
NILE_P3002	End Effector Nozzle	1
NILE_P3003	Camera Enclosure, Base	1
NILE_P3004	Camera Enclosure, Lid	1
NILE_P5000	NVIDIA Jetson Mounting Bracket	1
NILE_P5001	Flexible Conduit Mounting Bracket	1

A.2 Printed Circuit Boards

Electrical Part #	Description	Qty.
NILE_E5000	Hardware PCB	1
NILE_E2000	Trolley PCB	1

A.3 Purchased Parts

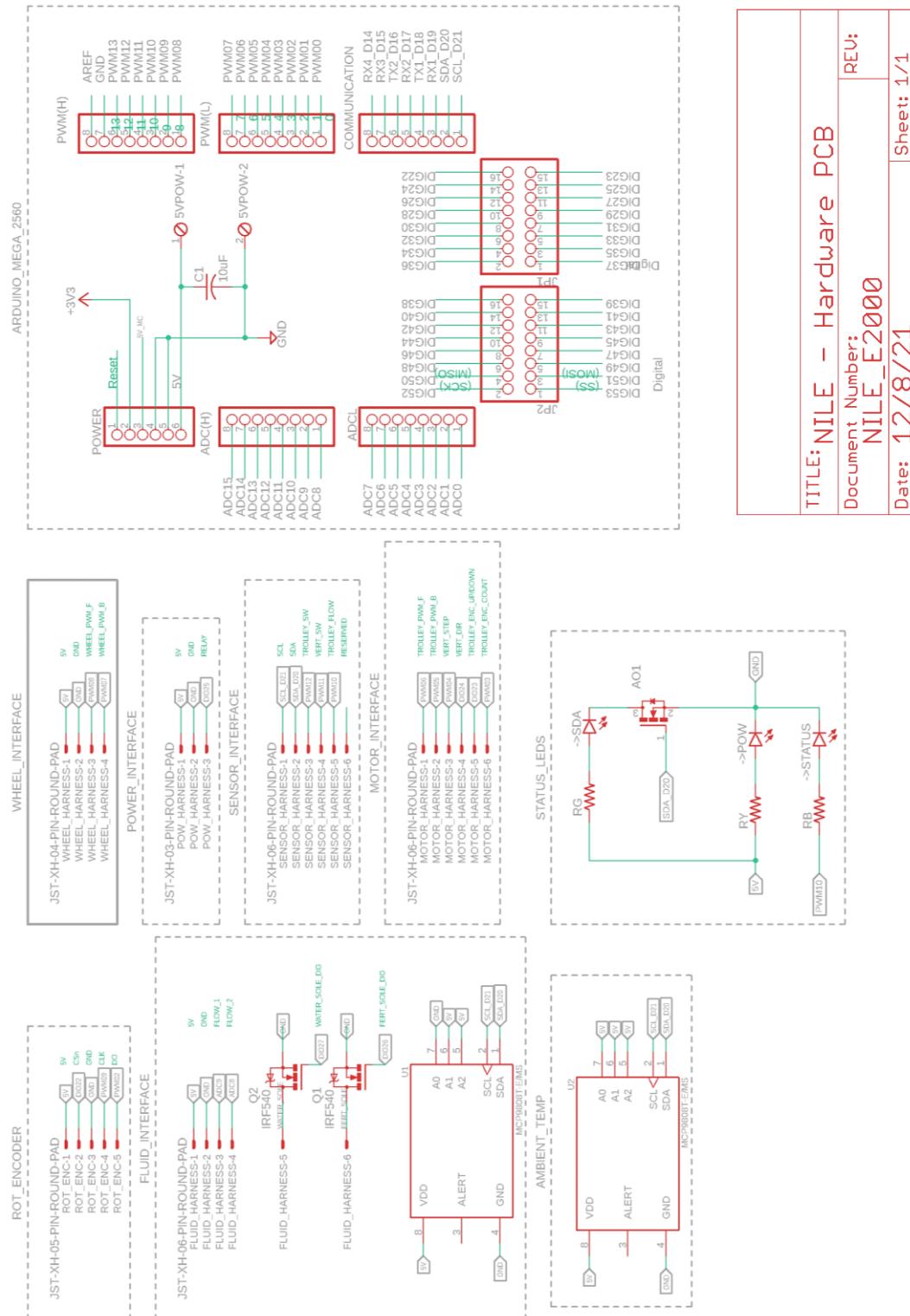
Manufacturer Part #	Description	Qty.	Unit Cost	Total Cost
95947A536	Aluminum Female Threaded Hex Standoff, 8mm Hex, 26mm Long, M4 x 0.70 mm Thread	5	1.85	9.25
25-4134	25 Series 4 Hole - Wide Gusseted Inside Corner Bracket	2	6.73	13.46
25-6706	25 Series Single-Keyed UniBearing™ Pad	2	2.45	4.9
25-6762	25 Series Single Long UniBearing™	2	19.97	39.94
6642K47	Ultra-Precision Lead Screw, Fast-Travel, 3/16"-42.7 Thread Size, 18" Long	1	56.79	56.79
6642K57	Fast-Travel Flange, 3/16"-42.7 Thread Size for Ultra-Precision Lead Screw	1	19.37	19.37
12024	10 & 25 Series Roller Wheel with Steel Hub	3	6.48	19.44
6659K673	Oil-Embedded Flanged Sleeve Bearing for 6 mm Shaft Diameter and 10 mm Housing ID, 6 mm Long	5	1.9	9.5
25-4165	25 Series 8 Hole External Flat Plate	2	7.08	14.16
6659K687	Oil-Embedded Flanged Sleeve Bearing for 18 mm Shaft Diameter and 22 mm Housing ID, 12 mm Long	2	2.26	4.52
98543A113	Side-Mount External Retaining Rings for 6 mm-8 mm OD, Black-Phosphate Spring Steel	6	N/A	4.24
GM9234S032-R1	DC Gear Motor pitman	1	N/A	0
5395T111	Set Screw Shaft Coupling, Black-Oxide Steel, for 6mm Diameter Round Shafts	1	3.66	3.66
42BYGHM809	Hybrid Stepper Motor	1	N/A	0
6412K9	Set Screw Shaft Coupling, Black-Oxide Steel, for 3/16" Diameter Round Shaft	1	7.91	7.91
FR801-001	FIRST CIM Motor	1	N/A	0

B08XX6Y65D	K-Type Thermocouple Temperature Sensor	1	7.69	7.69
EK1940	Capacitive Soil Moisture Sensor	1	N/A	0
P00L	Intex 8ft X 18inch Snapset Pool	1	15.5	15.5
023VPX1000-XX-23N08	CGI 100:1 Gear Box	1	N/A	0
AEAT-6012-A06	Rotary Encoder Magnetic 4096 Binary (Absolute) Vertical	1	37.46	37.46
828	Liquid Flow Meter - Plastic 1/2" NPS Threaded	2	9.95	19.9
QL-281913AGH	Hinged Cover Stainless Steel Latch 285x195x130mm Junction Box with Mounting Plate	2	35.99	71.98
110/220VAC-DC24V	24V 25A 600W LED Driver Switching Power Supply	1	46.9	46.9
-	NVIDIA Jetson Nano	1	N/A	0
6280K331	Roller Chain Sprocket for ANSI 35 Chain, 11 Teeth, for 3/8" Shaft Diameter	1	N/A	0
6236K251	Roller Chain Sprocket for ANSI 35 Chain, 40 Teeth, for 5/8" Shaft Diameter	2	N/A	0
-	1:8 Scale Master Truck Tires	2	35	70
5372K119	Plastic Barbed Tube Fitting for Air and Water Straight Adapter, for 3/8" Tube ID x 1/2 NPT Male	1	N/A	5.96
-	DC Voltage Regulator Buck Converter DC 24V Step Down to 12V 30A 360W	1	24.99	24.99
Songhe BTS7960 43A	12V DC Motor Driver	2	N/A	13.99
-	Plastic Water Solenoid Valve - 12V - 1/2" Nominal	2	6.95	13.9
5372K175	Plastic Barbed Tube Fitting for Air and Water Straight Adapter, for 3/8" Tube ID x 1/2 NPT Female	8	N/A	15.87
9675T47	Odor-Resistant Plastic Barbed Tube Fitting for Air and Water, Wye Connector, for 3/8" Tube ID	1	N/A	8.51
7291N12	Wet-Location Snap-in Sealing Grommet Cable Tie Point, for 16mm Hole Diameter, 20mm Flange Diameter	4	0.74	2.96
7291N14	Wet-Location Snap-in Sealing Grommet with Cable Tie Point, for 23 mm Hole Diameter	10	0.87	8.7
6126K47	Nylon Shaft Grommet for 9/16" Hole Diameter	1	3.5	3.5
HEDS-5640#A06	Three Cannel Optical Encoder	1	N/A	0
6338K414	Oil-Embedded Flanged Sleeve Bearing for 3/8" Shaft Diameter and 1/2" Housing ID, 1/4" Long	4	0.88	3.52
98408A134	Side-Mount External Retaining Ring 15-7 PH Stainless Steel, for 3/8" OD	9	N/A	5.56
DM332T	Digital Stepper Motor Driver 1.0-3.2A 10-30VDC	1	17.99	17.99
-	Igus Chain, .39in x .26in ID	1	N/A	0
-	R28 15mm x 40mm(Inner H x Inner W) Black Plastic Cable Wire Carrier Drag Chain	2	13.99	27.98
A000067	Arduino Mega 2560 Rev3	1	N/A	0
8065K32	3/4 Female Conduit x Knockout Male Adapter for Flexible Plastic Conduit	2	6.66	13.32
LS-KW10-AL-H	DEBOUNCE MINI LIMIT SW ARC LEVER	2	4.59	9.18

D415	Intel RealSense Depth Camera	1	N/A	0
SR1-1225-N	SSR RELAY SPST-NO 25A 24V-240V	1	18.95	18.95
-	DC 15A-20A ZVS Driver Ignition Tesla Coil	1	N/A	0
1487T3	UV-Resistant Firm PVC Tubing for Air and Water, Clear, 3/8" ID, 1/2" OD, 25FT	1	19	19
LS7184N	QUADRATURE CLOCK CONVERTER	1	5.13	5.13
TCN75AV0A71 3	Temperature Sensor Digital, Local -40°C ~ 125°C 11 b 8-SOIC	2	0.81	1.62
AD7995	10 Bit Analog to Digital Converter 3, 4 Input 1 SAR SOT-23-8	2	3.05	6.1

Appendix B – Electrical Schematics

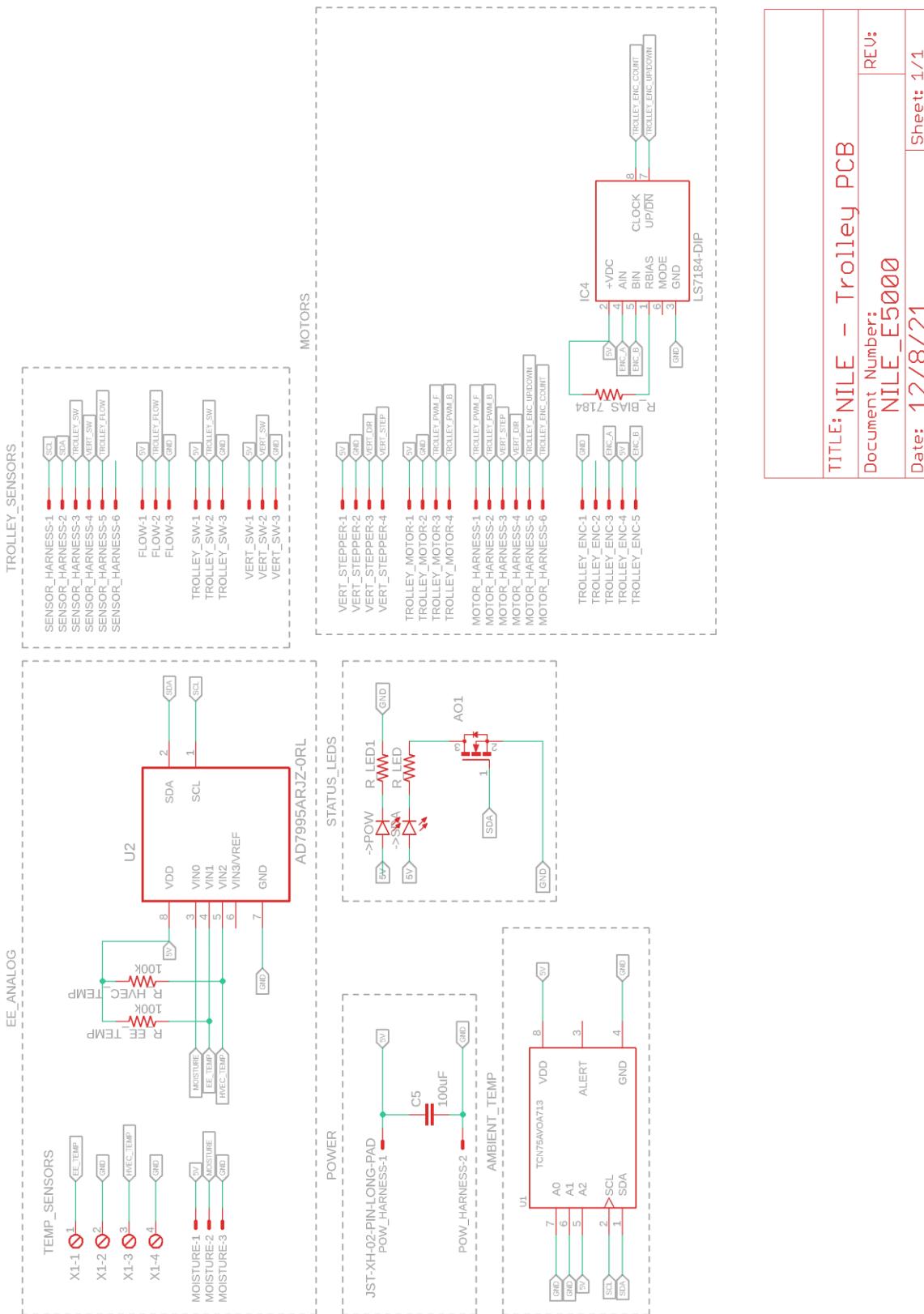
B.1 Arduino Hardware PCB Schematic



TITLE: NILE – Hardware PCB

Document Number: NILE_E2000
REV: 1/1
Date: 12/8/21
Sheet: 1/1

B.2 Trolley PCB Schematic



TITLE: NILE – Trolley PCB

Document Number: NILE_E5000

REV: 1/1

Date: 12/8/21

Appendix C – Source Code

C.1 Interfacing camera with Jetson Nano [Python]

```
import pyrealsense2 as rs
import cv2
import numpy as np

# Define image size parameters
WIDTH = 640
HEIGHT = 480

# Configure depth and color streams
pipeline = rs.pipeline()
config = rs.config()

# Get device product line for setting supported resolutions
pipeline_wrapper = rs.pipeline_wrapper(pipeline)
pipeline_profile = config.resolve(pipeline_wrapper)
device = pipeline_profile.get_device()
device_product_line = str(device.get_info(rs.camera_info.product_line))

found_rgb = False
for s in device.sensors:
    if s.get_info(rs.camera_info.name) == 'RGB Camera':
        found_rgb = True
        break
if not found_rgb:
    print("No color sensor detected")
    exit(0)

# Set the image frame size and FPS speed
# Note rs.format.z16 implements uint16 datatype
config.enable_stream(rs.stream.depth, WIDTH, HEIGHT, rs.format.z16, 30)
# Configure RGB camera input data, rs.format.bgr8 implements 8-bit red, 8-bit green, and 8-bit blue per pixel
config.enable_stream(rs.stream.color, WIDTH, HEIGHT, rs.format.bgr8, 30)

# Start streaming with the enumerated parameters
pipeline.start(config)

try:
    while True:

        # To reference specific key press
key = cv2.waitKey(1) & 0xFF

        # Wait for a coherent pair of frames: depth and color
```

```

frames = pipeline.wait_for_frames()
color_frame = frames.get_color_frame()
if not color_frame:
    continue

# Convert images to numpy arrays and display
color_image = np.asarray(color_frame.get_data())
cv2.imshow('Live stream', color_image)

if key == ord("q"):
    cv2.imwrite("Still_image.jpg", color_image)
    print("Image saved, program terminated")
    break

finally:
    # Close all windows and stop the stream
    cv2.destroyAllWindows()
    pipeline.stop()

```

C.2 Training machine learning classification algorithm on dataset of physical textures [Python]

```

# import the necessary packages
from texture_class import LocalBinaryPatterns
from sklearn.svm import LinearSVC
from imutils import paths
import argparse
import cv2
import os
import joblib

ap = argparse.ArgumentParser()
ap.add_argument("-t", "--training", required=True,
    help="path to the training images")
ap.add_argument("-e", "--testing", required=True,
    help="path to the testing images")
args = vars(ap.parse_args())

# initialize the local binary patterns descriptor along with
# the data and label lists
desc = LocalBinaryPatterns(24, 8)
data = []
labels = []

# loop over the training images
for imagePath in paths.list_images(args["training"]):
    # load the image, convert it to grayscale, and describe it
    image = cv2.imread(imagePath)

```

```

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
hist = desc.describe(gray)
# extract the label from the image path, then update the
# label and data lists
labels.append(imagePath.split(os.path.sep)[-2])
data.append(hist)

# train a Linear SVM on the data
# create the classifier
print("[STATUS] Creating the classifier..")
model = LinearSVC(C=100.0, random_state=42)
# fit the training data and labels
print("[STATUS] Fitting data/label to model..")
model.fit(data, labels)

# Save the model to disk file
# filename = 'trained_texturemodel.sav'
# joblib.dump(model, filename)

# loop over the testing images
for imagePath in paths.list_images(args["testing"]):
    # load the image, convert it to grayscale, describe it,
    # and classify it
    image = cv2.imread(imagePath)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    hist = desc.describe(gray)
    prediction = model.predict(hist.reshape(1, -1))

    # display the image and the prediction
    cv2.putText(image, prediction[0], (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
               1.0, (0, 0, 255), 3)
    cv2.imshow("Image", image)
    cv2.waitKey(0)

```

C.3 Loading still image and applying pre-trained texture classification algorithm on detected contours [Python]

```

# Identify contours within image, and classify individual contours based on LBP texture analysis
# Import the necessary packages
from texture_class import LocalBinaryPatterns
import argparse
import cv2
import imutils
import numpy as np
import joblib

# Construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", type=str, required=False,

```

```

    help="path to input image")
args = vars(ap.parse_args())

# Define image dimensions to use when resizing
WIDTH = 600
HEIGHT = 480

if not args.get("image", False):

    # If no user-supplied path, load the a preset image
    original = cv2.imread("/home/pyimagesearch/software/python/camera_test/still_image.jpg")

else:
    # Otherwise, use the user-supplied path
    original = cv2.imread(args["image"])

# Resize image
dim = (WIDTH, HEIGHT)
resized = cv2.resize(original, dim)
cv2.imshow("Resized Original", resized)
result = resized.copy()

# Define HSV boundaries for desired color
blueLower = (50, 80, 10)
blueUpper = (100, 255, 255)

greenLower = (40, 80, 10)
greenUpper = (80, 255, 255)

# Define RGB red color for drawing
red = (0, 0, 255)

# Initialize the local binary patterns descriptor
desc = LocalBinaryPatterns(24, 8)

# Apply Gaussian Blur and convert to HSV colorspace
blurred = cv2.GaussianBlur(resized, (11, 11), 0)
hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)

# Apply color mask, with erosion-dilation sequence to eliminate tiny blotches
masked = cv2.inRange(hsv, greenLower, greenUpper)
masked = cv2.erode(masked, None, iterations=2)
masked = cv2.dilate(masked, None, iterations=2)

# Find contours in the masked image
contours = cv2.findContours(masked.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
# Parse the contours list to account for varying OpenCV handling methods
contours = imutils.grab_contours(contours)

```

```

# Initialize array for storing contour images
cropped_images = []

# Only proceed if at least one countour was found
if len(contours) > 0:

    for i in range(len(contours)):
        # Create black canvas matching dimensions of the image
        mask = np.zeros(resized.shape[:2], dtype = "uint8")
        # Fill in the detected contour in white against black background
        cv2.drawContours(mask, contours, i, color=(255,255,255),thickness = -1)
        # Perform bitwise AND to extract image regions within contours
        bitwiseAnd = cv2.bitwise_and(resized, resized, mask = mask)
        # Generate and draw a bounding box for the detected contour to be displayed on final result
        x,y,w,h = cv2.boundingRect(contours[i])
        cv2.rectangle(result, (x,y), (x+w, y+h), red, 2)
        # Store the cropped contour image and display
        # cropped_images.append(bitwiseAnd[y:y+h, x:x+w])
        cropped_images.append(bitwiseAnd)

else:
    print("No contours found!")

# Apply texture analysis to contour-bound cropped images
# First load the pre-trained texture model
loaded_model = joblib.load('trained_texturemodel.sav')
print("[STATUS] Loading trained model...")
# loop over the extracted image segments
i = 0
for image in cropped_images:
    # load the image, convert it to grayscale, describe it,
    # and classify it
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    hist = desc.describe(gray)
    prediction = loaded_model.predict(hist.reshape(1, -1))

    # display the image and the prediction
    cv2.putText(image, prediction[0], (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
               1.0, (0, 0, 255), 3)
    cv2.imshow("Image", image)
    cv2.waitKey(0)

segmented = cv2.bitwise_and(resized, resized, mask = masked)

# Display final results
cv2.imshow("Contours", result)
cv2.imshow("Color masked", masked)
cv2.imshow("Segmented Image", segmented)

```

```

cv2.imwrite("detected_contours.jpeg", result)
cv2.imwrite("mask.jpeg", masked)
cv2.imwrite("segmented_green.jpeg", segmented)
cv2.waitKey(0)

# # Display the contour-bound image segments
# for image in cropped_images:
#     cv2.imshow("Cropped", image)
#     cv2.waitKey(0)

cv2.destroyAllWindows()

```

C.4 TCN75AVOA713 Driver library [C++]

Header: "temp_i2c.h"

```

#ifndef temp_i2c
#define temp_i2c

#include <Arduino.h>

class TempI2C
{
public:
    TempI2C(int address);
    void init();
    double temp();
private:
    int adr;
};

#endif

```

Source: "temp_i2c.cpp"

```

#include "temp_i2c.h"
#include <Wire.h>

TempI2C::TempI2C(int address)
{
    adr = address;
}
void TempI2C::init()
{
    Wire.begin();
    Wire.beginTransmission(adr);
    Wire.write(0x01); //selects configuration register

```

```

Wire.write(0x60); //sets to max accuracy
Wire.endTransmission();
delay(70);
Wire.beginTransmission(adr);
Wire.write(0x00);
Wire.endTransmission();
}
double TempI2C::temp()
{
    Wire.requestFrom(adr, 2);
    int data = 0;
    if (2 <= Wire.available()) {
        data = Wire.read();
        data = data << 8;
        data |= Wire.read();
    }
    data = data >> 4;
    return double(data * 0.0625);
}

```

C.5 AD7995 Driver library [C++]

Header: "adc_i2c.h"

```
#ifndef adc_i2c
#define adc_i2c
```

```
#include <Arduino.h>
```

```
class ADCI2C
{
public:
    ADCI2C(int address);
    void init();
    int read(int channel);
private:
    int adr;
};
```

```
#endif
```

Source: "adc_i2c.cpp"

```
#include "adc_i2c.h"
#include <Wire.h>
```

```
ADCI2C::ADCI2C(int address)
{
    adr = address;
}
```

```

void ADCI2C::init()
{
    Wire.begin();
}
int ADCI2C::read(int channel)
{
    byte reg = 0;
    switch(channel) {
        case 0: reg = 0x10; break; //read from Vin0 MOISTURE
        case 1: reg = 0x20; break; //read from Vin1 EE TEMP
        case 2: reg = 0x40; break; //read from Vin2 HVEC TEMP
        default: reg = 0; break;
    }
    Wire.beginTransmission(adr);
    Wire.write(reg); //selects configuration register
    Wire.endTransmission();

    delay(50);

    Wire.requestFrom(adr, 2);
    int data = 0;
    if (2 <= Wire.available()) {
        data = Wire.read();
        data &= 0x7;
        data = data << 8;
        data |= Wire.read();
    }
    return data;
}

```

C.6 LS7184N library [C++]

Header: "quad_enc.h"

```

#ifndef quad_enc
#define quad_enc

#include <Arduino.h>

class QuadEnc
{
public:
    QuadEnc(int clk, int dir);
    void init();
    int count();
private:
    int clk_pin;
    int dir_pin;
}

```

```

};

#endif

Source: "quad_enc.ccp"
#include "temp_i2c.h"
#include <Wire.h>

TempI2C::TempI2C(int address)
{
    adr = address;
}
void TempI2C::init()
{
    Wire.begin();
    Wire.beginTransmission(adr);
    Wire.write(0x01); //selects configuration register
    Wire.write(0x60); //sets to max accuracy
    Wire.endTransmission();
    delay(70);
    Wire.beginTransmission(adr);
    Wire.write(0x00);
    Wire.endTransmission();
}
double TempI2C::temp()
{
    Wire.requestFrom(adr, 2);
    int data = 0;
    if (2 <= Wire.available()) {
        data = Wire.read();
        data = data << 8;
        data |= Wire.read();
    }
    data = data >> 4;
    return double(data * 0.0625);
}

```

C.7 Sensor Testing File [C++]

Source: "NILE_Hardware.ino"

```

#include "temp_i2c.h"
#include "adc_i2c.h"
#include "quad_enc.h"
#include <Wire.h>

TempI2C TEMP(0x48);
ADCI2C ADC1(0x28);

```

```

QuadEnc TrolleyQuad(3,23);

#define ROT_COUNT 4096

int t_count = 0;

void pciSetup(byte pin)
{
    *digitalPinToPCMSK(pin) |= bit (digitalPinToPCMSKbit(pin)); // enable pin
    PCIFR |= bit (digitalPinToPCICRbit(pin)); // clear any outstanding interrupt
    PCICR |= bit (digitalPinToPCICRbit(pin)); // enable interrupt for the group
}

ISR (PCINT0_vect) // handle pin change interrupt for D8 to D13 here
{
    t_count += TrolleyQuad.count();
}

ISR (PCINT1_vect) // handle pin change interrupt for A0 to A5 here
{

}

ISR (PCINT2_vect) // handle pin change interrupt for D0 to D7 here
{

}

void setup() {
    // put your setup code here, to run once:
    cli();
    PCICR |= 0b00000111; // turn on all ports
    PCMSK0 |= 0b00010000; // PCINT0
    sei();

    Serial.begin(9600);
    //Serial.println("Starting...");
    pinMode(6,OUTPUT);
    pinMode(5,OUTPUT);

    TrolleyQuad.init();
    TEMP.init();
    ADC1.init();
}

int desired[4] = {1, 3, -2, 0};
int i = 0;

```

```

void loop() {
    double val = TEMP.temp();
    int err = t_count - desired[i]*ROT_COUNT;
    int adc0 = ADC1.read(0);
    int adc1 = ADC1.read(1);
    int adc2 = ADC1.read(2);
    //Serial.print("Temp: ");
    //Serial.println(val);

    //Serial.print(" ");
    //Serial.print(adc0);
    //Serial.print(" ");
    //Serial.print(adc1);
    //Serial.print(" ");
    //Serial.println(adc2);

    Serial.println(t_count);
    if (abs(err) < 50) {
        i++;
        if (i > 3) {i = 0;}
    }

    uint8_t control = min((abs(err))/15,255);
    if (err >= 0) {
        analogWrite(5,control);
        analogWrite(6,0);
    } else {
        analogWrite(5,0);
        analogWrite(6,control);
    }

    delay(1);
}

```

C.8 Dynamic Simulation and Controller [MATLAB]

```
clear all
close all

clc

h=0.001; %simulation time step
t=0:h:240; %time array
Ts = 1/100;
T_last = -Ts;

b=zeros(6,length(t)); %initialize state vector
V=zeros(3,length(t)); %initialize generalized force vector
b(:,1)=[0.2;... %initial d2
0;...%initial d3
0;...%initial theta4
0;... %initial dotd2
0;... %initial dotd3
0]; %initial dottheta4

%define gearing ratios of wheel sprocket
N2 = 40;
N1 = 11;
%define radius of wheel and length of gantry
r_wheel = 82.5e-3/2;
l_gantry = 1095.9432e-3;

%sets target positions
r_d = [-0.5 -0.5 0 0.5;
        0 0.5 0.5 0.5;
        0.45*ones(1,4)];
target = zeros(3,length(t));
target_g = zeros(3,length(t));

%determine IK of targets
gamma_t = zeros(3,4);
for n = 1:length(gamma_t)
[theta, d, v] = NILE_IK_HVEC(r_d(:,n));
gamma_t(:,n) = [d; v; theta/(r_wheel/l_gantry*N1/N2)];
end

eps = 50e-3;
mode = 0;

gamma_d = gamma_t(:,1);
e_d = zeros(3,1); %set to zeroed
edot = zeros(3,1);
```

```

eint = zeros(3,1);
e_last = b(1:3,1);
%Controller gains
Kp = diag([5 40 40]);
Kd = diag([7 5 1]);
Ki = 0*diag([1 1 1]);

V_max = [24; 24; 12];
V_c = zeros(3,1);

k = 1;
%---Solve Equations of Motion---
for i=1:length(t)-1
    %samples at Ts
    if t(i)-T_last >= Ts
        gamma = b(1:3,i);
        e = e_d - gamma; %error signal
        edot = (e_last - e)./Ts; %derivative error
        eint = (eint + e_last.*Ts); %integral error
        if (norm(e) <= eps) %switches control once error is low (close to target)
            switch mode
                case 0
                    %initialize system with original gains
                    mode = 1; %switch mode
                    e_d = gamma_d.*[0; 0; 1]; %only
                    Kp = diag([5 40 40]);
                    Kd = diag([7 5 1]);
                    Ki = diag([1 5 1]);
                case 1
                    %
                    mode = 2
                    e_d = gamma_d.*[1; 0; 1];
                    e = e_d - gamma;
                    edot = (e_last - e)./Ts;
                    eint = eint.*[0; 1; 0];
                case 2
                    mode = 3
                    e_d = gamma_d-[0;0.005;0];
                    e = e_d - gamma;
                    edot = (e_last - e)./Ts;
                    eint = eint.*[0; 0; 0];
                case 3
                    mode = 4
                    e_d = gamma_d;
                    e = e_d - gamma;
                    edot = (e_last - e)./Ts;
                    eint = eint.*[0; 0; 0];
                    Kp = diag([5 10 40]);
                    Kd = diag([7 5 1]);
            end
        end
    end
end

```

```

Ki = diag([1 20 1]);
case 4
    mode = 5
    e_d = gamma_d.*[1; 0; 1];
case 5
    mode = 0
    k = k+1;
    if (k > length(gamma_t))
        k = 1;
    end
    gamma_d = gamma_t(:,k);
    e_d = gamma_d.*[0; 0; 1];
end
end
V_c = Kp * e - Kd*edot + Ki*eint;
V_c = bound(V_c,-V_max,V_max);
T_last = t(i);
e_last = e;
end

target(:,i) = r_d(:,k);
target_g(:,i) = gamma_t(:,k);
V(:,i) = V_c;
k1=NILE_Forces(b(:,i),V_c);
k2=NILE_Forces(b(:,i)+k1*h/2,V_c);
k3=NILE_Forces(b(:,i)+k2*h/2,V_c);
k4=NILE_Forces(b(:,i)+k3*h,V_c);
b(:,i+1)=b(:,i)+h*(k1/6+k2/3+k3/3+k4/6);
end

theta1 = b(3,:)*r_wheel/l_gantry*N1/N2;
gamma_t1 = target_g(3,:)*r_wheel/l_gantry*N1/N2;

%---Data Plotting---
figure(1)
subplot(4,1,1)
plot(t,theta1(1,:),t,gamma_t1(1,:),'r--')
xlabel('t (s)')
ylabel('theta_1 (rad)')
subplot(4,1,2)
plot(t,b(1,:),t,target_g(1,:),'r--')
xlabel('t (s)')
ylabel('d_2 (m)')
subplot(4,1,3)
plot(t,b(2,:),t,target_g(2,:),'r--')
xlabel('t (s)')
ylabel('d_3 (m)')
subplot(4,1,4)

```

```

plot(t,b(3,:),t,target_g(3,:),'r-')

xlabel('t (s)')
ylabel('\theta_4 (rad)')

trolley_fpwm = bound(V(1,:)/V_max(1),0,25);
trolley_bpwm = -bound(V(1,:)/V_max(1),-25,0);

step_angle = deg2rad(0.9);
step_displacement = step_angle/(100*40/11);
step_pwm = abs(V(2,:)/V_max(2));
step_dir = sign(V(2,:));

wheel_fpwm = bound(V(3,:)/V_max(3),0,25);
wheel_bpwm = -bound(V(3,:)/V_max(3),-25,0);

%% Control Plotting

minimum = 0;
maximum = 60;

zoom_center = (maximum+minimum)/2;
zoom_range = (maximum-minimum);

figure(3)
subplot(3,1,1);
title('Trolley Motor Driver Input');
plot(t,trolley_fpwm*100,'r',t,trolley_bpwm*100,'b');
grid on;
xlabel('t (s)');
ylabel('PWM Duty Cycle (%)');
legend('Forward PWM','Backward PWM','Location','southeast');
axis([zoom_center-zoom_range/2,zoom_center+zoom_range/2,0,110]);

subplot(3,1,2);
title('Vertical Stepper Motor Driver Input');
plot(t,step_pwm*100,'r',t,step_dir*100,'b');
grid on;
xlabel('t (s)');
ylabel('PWM Duty Cycle (%)');
legend('Step PWM','Direction','Location','southeast');
axis([zoom_center-zoom_range/2,zoom_center+zoom_range/2,0,110]);

subplot(3,1,3);
title('Wheel Motor Driver Input');
plot(t,wheel_fpwm*100,'r',t,wheel_bpwm*100,'b');
grid on;
xlabel('t (s)');
ylabel('PWM Duty Cycle (%)');

```

```

legend('Forward PWM','Backward PWM','Location','southeast');
axis([zoom_center-zoom_range/2,zoom_center+zoom_range/2,0,110]);

%---Animation---

figure(2)
v=VideoWriter('NILE.mp4','MPEG-4'); %create a video object – this will be stored as
dri_planar.avi
set(v,'FrameRate',20); %set the frame rate to 20 FPS
open(v); %open the video
for i=1:round(10/(20*h)):length(t) %20 frames per secondE
hold on;
NILE_draw([theta1(i); b(1:3,i)]);
plot3(target(1,i),target(2,i),target(3,i),'r*');
hold off;
drawnow
frame=getframe(gcf); %store the current figure window as a frame
writeVideo(v,frame); %write that frame to the video
%display(i)
end
close(v); %close the video

```

```

function y = bound(x,bl,bu)
% return bounded value clipped between bl and bu
y=min(max(x,bl),bu);
end

```

Appendix D – Mechanical Drawings