Fall 12-9-2021

AUTONOMOUS CONTROL AND SIGNAL ACQUISITION SYSTEM

Rion Cadell Krampe
Murray State University

Murray State University Honors College

HONORS THESIS

Certificate of Approval

AUTONOMOUS CONTROL AND SIGNAL ACQUISITION SYSTEM

Rion Krampe

December 2021

Approved to fulfill the requirements of HON 438	Dr. Gheorghe Bunget, Associate Professor School of Engineering
Approved to fulfill the Honors Thesis requirement of the Murray State Honors Diploma	Dr. Warren Edminster, Executive Director Honors College

Examination Approval Page

Author: Rion Cadell Krampe

Project Title: AUTONOMOUS CONTROL AND SIGNAL ACQUISITION SYSTEM

Department: School of Engineering

Date of Defense: Thursday, December 2nd, 2021

Approval by Examining Committee:

(Dr. Gheorghe Bunget, Advisor)	(Date)
(Dr. James Rogers, Committee Member)	(Date)
(Dr. James Hereford, Committee Member)	(Date)

AUTONOMOUS CONTROL AND SIGNAL ACQUISITION SYSTEM

Submitted in partial fulfillment

of the requirements

for the Murray State University Honors Diploma

Rion C. Krampe

December 2021

Our goal is to continue a previous teams project to build a NDE autonomous control and signal acquisition system that is more precise, more customizable with both code and mechanical parts, and cheaper than a similar system bought by the school. This goal has two stages to it. First, to repair the system from considerable damage it received during transportation. Secondly, to continue designing and developing the system to make considerable progress towards the goal of a fully functional NDE system. Along with making progress we must consider the team after us and create an easy stepping off point for the next team to continue. For mechanical we upgraded the slotted aluminum extrusions, upgraded threaded rods to precision ball screw assemblies, replaced unsupported slide rails with supported, and redesigned and printed 3D parts to account for changes. For electrical we rewired the system, replaced, and upgraded the stepper drivers, overhauled the microcontroller to a microprocessor, and made suggestions for a digital encoder. For code we completely reworked it from the ground up with both a GUI and Utility file. We increased the stability and reduced error due to bending and shaking with changes to the aluminum extrusions and supported slide rails. We increased system precision with the precision ball screws. We increased the strength of the system by redesigning and printing 3D parts with stronger plastics on better printers. The stepper motors have been updated and improved for a higher current rating. The microprocessor streamlines code and enables the user with a more robust system to operate with. The code now controls the system in fully autonomous movement with a GUI for easy user control. All this work was done for around $1,000 making it significantly cheaper than a market version. Although there is still plenty of work to do to finish the system, considerable progress has been made towards the goal.

CONTENTS

LIST OF FIGURES & TABLES

INTRODUCTION

This document is an Honors Thesis for a singular person, me, Rion Krampe. It is still, however, over what was a team project and is a rewritten team report. To accommodate for this language is specific to who did what. If I did 100% of a task then "I" did it, however if some of the tasks were accomplished some part by another team member then "We" did it. That said, here are some things that I specifically did as electrical, coder, and team leader.

I rewrote this entire paper (no text directly copied from Team Report), worked a semester longer on the project than the rest of the team, was the project leader and a part of all design choices, researched, analyzed, and purchased ball screws, tested and rewired stepper motors, tested, replaced, and calibrated stepper drivers, was not initially in charge of microcontroller but assumed control and took it from SAMduino to Raspberry Pi, did analysis on future encoders, deciphered old code flow and intent, updated code, and developed code into current state. All the code for this project is completely mine, only inspiration was taken from previous code.

The autonomous control and signal acquisition system, otherwise known as a Non-Destructive Evaluation Device, is designed to gather measurement data autonomously and without damaging the material in question via the use of ultrasonic transducers. The motivation for this device for our customer, Dr. Bunget, is to have an NDE device for his laboratory which is less expensive and smaller than some on the industrial market. This system, initially designed by a previous team, was built to periodically scan and store data using an oscilloscope while also controlling the position of an ultrasonic transducer. The goal for this project will be to expand upon their work and continue the development of the existing system. This required that we identified the current issues with the system and repair them as well as make innovations to the system to meet customer needs. There were several improvements we considered once major repairs were completed.

Mechanical improvements focus on replacing some parts to meet our overall goal of making sure the system is stable and accurate so that future measurements have as little error as possible. The changes include redesigning and printing 3D printed parts with a better material, increasing the precision and reducing vibrations with precision ball screws, and increasing the sturdiness of the system with supported slide rails along the x-axis to reduce vertical position error and return clearer measurements. As desired by the customer, an encoder should be added to properly update the position of the system so that the position in the code can match the physical location. Being able to determine the exact location will increase measurement accuracy and keep the system aligned throughout the entire scan. Another requirement was updating the code to a newer version of Python. The conversion from Python 2 to Python 3 is necessary for keeping the system modern and up to date as the former is no longer being supported. I will also replace microcontroller with a microprocessor to increase power, speed, control, and adaptability of the system. The biggest constraint will be compatibility and the integration of all the changes with the existing system. All electrical components needed to be compatible and the mechanical components needed to be seamlessly integrated with minimal system redesign. Another major constraint is continuing the work of another team with only a limited amount of documentation on their work available. With the code specifically I had had to spend a large amount of time piecing together what they did, why they did that, and what they had planned for future teams.

DESCRIPTION OF FINAL PROTOTYPE AND ANALYTICAL METHODS OF DESIGN

Overview of the System

The framework is made up of 40 mm slotted aluminum extrusions set on top of a metal table with the y and z-axis frames being made up of 20 mm slotted aluminum. Two stepper motors with precision ball screw rods are set up for the x-axis moving the y and z-axis frame. The same stepper motor is set up on the y-axis with a threaded rod moving the z-axis frame and the same stepper motor and threaded rod is set up on the z-axis to raise and lower the ultrasonic transducer. The system can move along the x, y, and z-axes with 12.5 μm precision on the x-axis and approximately 3.2 μm precision on the y and z-axis. The x and y-axis are set upon supported slide rails that the system moves along supported by linear slide bearings. The x-axis runs parallel to the two spaced out slide rails, the y-axis runs perpendicular, and the z-axis is vertical. A model of the system's motor system from the previous team is shown in Fig. 1. A partial model of the system was recreated using updated parts shown in Fig. 2. You can see where we replaced the 20 mm extrusions for the x-axis and below as well as the unsupported slide rails for the x-axis. It also depicts precision ball screws on the y-axis, a change we have the parts for but have not done for a later discussed reason. Fig. 2 was made to help visualize our changes to the model and to see how each component would integrate. This showed us that we would have to consider much more than we originally thought regarding replacing parts. We have discovered that there would be many changes to the location of certain parts and their lengths. The addition of the ball screw assemblies in each axis as a replacement for the threaded rods will push the motors off the main frame of the system in both the x and y-axis. This will require us to design a method of connecting the y-axis ball screw fixed end to the system as well as a method of supporting the x and y-axis floating motors. An increase in thickness of the aluminum extrusions changed some lengths. The supported slide rails and open linear bearings will increase the height of the upper portion (the y and z-axis) of the system. The new part dimensions and the change in height will require a redesign of 3D printed parts attached to these components. The SolidWorks representation of our system has been critical in ensuring our ideas are feasible. With it, we were able to make our ideas real and test them, specifically regarding dimensions. We have a much more solid idea of how it will all fit together and are more certain that our changes can work.

The following sections will go in depth over each aspect of the system in this manner: A description of the current system, a description of it previously and why the previous team made those choices, improvements we that could be made from the previous system, improvements that we made, our analytical analysis of why we made those choices, our recommendations moving forward for future teams.

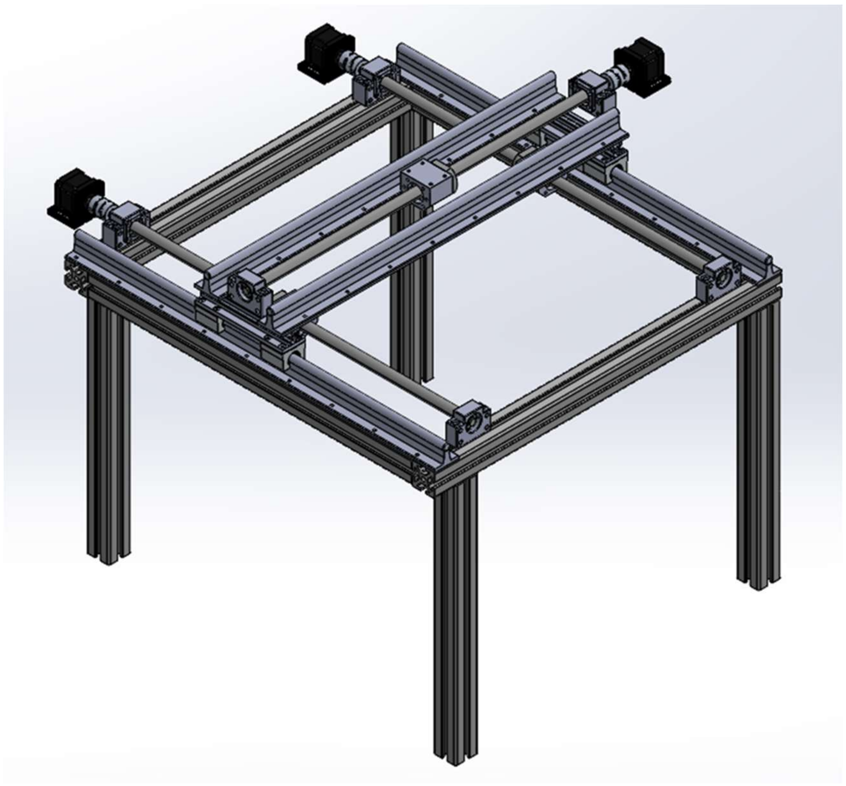


Figure 1. A partial model of the systems motor system.

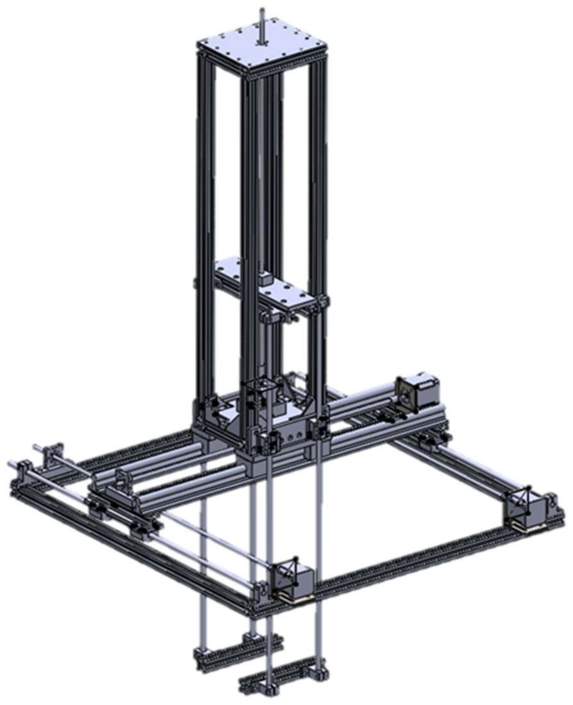


Figure 2. The partial SolidWorks model. Some parts are floating, and many 3D printed parts no longer need to be included. The z-axis is not included.

Mechanical

Slotted Aluminum Extrusions

The framework of the system was built with 20 mm aluminum slotted extrusions and has now been replaced with 40 mm extrusions. Specifically, the box frame that supports the x-axis and encloses the immersion tank has been replaced with 40 mm while the y and z-axis has been left untouched.

The previous team chose to use slotted extrusions because they could be easily pieced together with the use of brackets and have several different options of supporting hardware specifically made for them. This makes the assembly of a system with multiple supporting components straightforward. Slotted metal extrusions are, in general, an easy and common design path to use by engineers when creating a solid structure that needs to be customizable and easy to manufacture. As of why they chose aluminum, it has enough strength for the project, does not rust, is more appealing in appearance than most metal, easy to manufacture/ cut, and affordable compared to other metals.

We noted that the 20 mm was not a stable support for the system. When the motors ran, the system would shake and rattle due the vibrations that stepper motors produce at low rpms. Without the aid of external cross supports the system would lean and twist from its own weight. This would produce an error as it would change the position of the transducer in relation to the immersion tank and this would distort the recorded waveforms and stored position of said waveform. We also saw that the thin 20 mm extrusions compounded with the smaller 20 mm right-angle connectors would create a small bend in the extrusions parallel to the movement of the x-axis motors. This was an issue for similar reasons.

To fix this we replaced the noted 20 mm extrusions with 40 mm. To approximate how much the stiffness will increase by, we can calculate the moment of inertia using where b is the base and h is the height of the slotted extrusions. The calculations shown in Appendix A show that doubling the thickness results in an increase in stiffness by 2^4. This greatly increased the rigidity of the system and reduced vibrations from the motors. Now with thicker supports and more surface area per angle connector it is significantly harder to rock, does not twist under its own weight, require external cross supports, or significantly vibrate due to the motors. The table did limit how wide we can make the system but since the immersion tank was a narrower constraint, we only needed to keep the frame centered on the tank to allow for the maximum amount of scanning area. The

40 mm aluminum extrusions cost $136.50 for the necessary length of 6 meters. The portion that has yet to be replaced is the y and z-axis.

The increase in thickness of the y and z-axis would require a complete redesign of that portion, specifically you would have to find a way to make sure that the floating and fixed end of the ball screw are attached directly to the aluminum extrusions. While that can be done, it is unnecessary to do so at this time. The z-axis does not need to be strengthened like the rest of the system since it only experiences movement during setup and not during a scan. The system also moves slow enough during a scan to not cause a wobble in the z-axis. As for the y-axis we realized, after buying the ball screw assemblies, that it would be difficult to attach the floating and fixed end to the extrusions. We would have to expand the length of the y-axis and reprint all the parts for the y-axis and up. We were still assembling the base with the 40 mm and since the site we bought from did not offer a CAD model of the ball screw assemblies, we had to design it immediately after receiving the parts rather than being able to check in advance with a SolidWorks model. Our goal was to create a system that could produce autonomous movement and we were concerned that between shipping and redesigning times that trying to improvise the y-axis would leave us with an incomplete product for the next team. Our focus then became designing the system so that the next team had our hindsight as their foresight as well as recommendations for how to redesign it.

If a future team can easily integrate the change of 40 mm at that level, then that is a suggested course to keep the system consistent and increase the rigidity of the system. It will require a complete redesign of the z-axis tower which Dr. Bunget has placed as a small concern given that the z-axis is only used for initial positioning and does not influence the autonomous precision of the scan. Integrating the purchased ball screw assembly into the y-axis is a priority though and so when looking to change they extrusions or 3D parts at that level they should either ask: How can I change the spacing of the y-axis to fit the ball screw on the system? Or Should I make the system as efficiently shaped as possible and buy a custom length ball screw.

Precision Ball Screw Assembly

The x-axis is set up with two 650 mm length precision ball screw assemblies with a 5 mm lead. The y and z-axis each still use one 5/8” threaded rod with a 1.27 mm lead, a standard nut for movement, taped together nuts to create fixed end supports, and 3D printed nut brackets. I have also purchased and obtained another 650 mm ball screw assembly for the y-axis and a 450 mm assembly for the z-axis. Both are similar to the x-axis assemblies and we have intentionally not integrated either into the system.

The previous team had wanted to use precision ball screw assemblies for the system but due to funds and shipping times had to improvise with the 5/8” threaded rods. Any kind of linear slide assembly requires at least on side to be fixed so that there is no liner wobble in the screw. To accomplish this they screwed two nuts on the rod next to the 3D printed end supports and then taped them for extra measure. For the z-axis since the motor is placed on the bottom of the rods there is no need for a fixed end support since the rod rests directly upon the motor axle.

I saw that this setup is imprecise due to the large tolerance between the nut and threaded rod, weak to motor loads causing a bend in the rods, unprofessional in appearance, and observed missing steps in the stepper motors due to increased friction.

As suggested by the previous design team, we replaced the x-axis threaded rods with 650 mm precision ball screw assemblies as seen in Fig. 3. This increases the systems precision of position, the overall strength and rigidity, reduce friction which therefore decreases the chance of motor stalling, and improve the overall appearance of the system. When looking at official vendors I found the same parts on Amazon for significantly less at \$185.50 for three 650 mm and one 450 mm assemblies. The parts were of the same material, standard, and sizes and have a fixed support, free support, coupler, bracket, and nut screw on the rod.

I choose a 650 mm precision ball screw rod for the x and y-axis and 450 mm for the z-axis due to the limitations of the current frame setup. The selected ball screws chosen to have a ball nut diameter of 16 mm and a screw lead of 5 mm. The current threaded system has a lead of 1” per 20 turns and the stepper motors take exactly 400 steps for a single revolution which gives us the numbers in Table 1. I saw that the threaded rod is technically capable of a smaller index at 3.2 µm per step as opposed to the ball screws minimum index of 12.5 µm. In addition to this only being a difference of 9.3 µm, the threaded rods have an inherently larger lead error than the ball screws due to the large tolerance between rod and nut. If the tolerance error in the threaded rods is greater than 9.3 µm then the ball screw is more precise. The tolerance is so large that it can be measured with a ruler or caliper at about 1 mm. The larger margin of error found in the thread rods is due to the lack of contact and greater freedom of movement as opposed to the multi-contact ball bearings in the ball screws that reduce how much the ball nut can loosely move around. For a ball screw assembly, I needed a fixed end support, a free end support, a ball nut bracket, and a motor coupling. I chose a fixed-free end support system over a fixed-fixed partially for the cheaper price but mainly because the load of the system will be borne by the supported slide rails and the precision ball screws will not have a significant load on them, so a fixed-free system is enough to preserve the accuracy of the system. Having a free support on the motor end will also make it easier to assemble the ball screws to the motors. Specifically, it leaves room for them to be slid into place when assembling a motor mount. A design consideration for the future task of replacing the threaded rods on the y and z-axis. I bought the assemblies off amazon because when I originally found the price though, the less expensive standard supplier, automation4less, the price came out to $1,042.20 for four ball screw assemblies. This would have taken up more than two-thirds of our budget and with another team member's supported slide rails, we would have gone over our budget with just two areas. Instead, I bought them off Amazon at $185.50 for three 650 mm and one 450 mm assembly. The only real downside was that they came at set lengths in 50 mm increments. This meant that we would have to build the frame around these preset lengths.

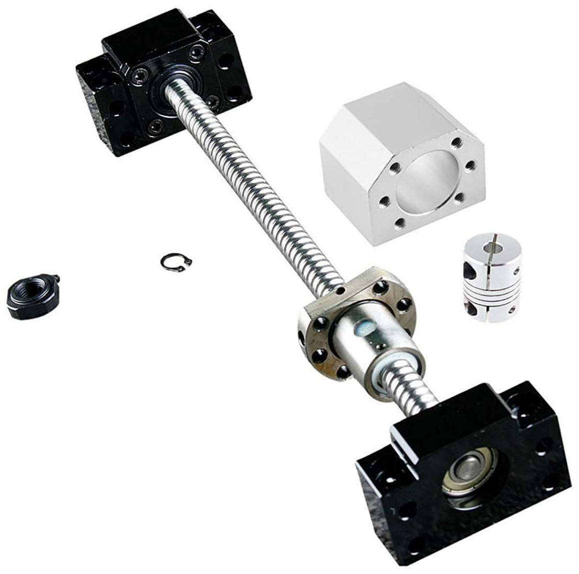


Figure 3. Precision ball screw assembly.

Table 1. Threaded rod vs Precision Ball Screw Leads

Type	Lead (mm/rev)	Step Lead (mm/step)	Step Lead (μm/step)
Threaded rod	1.27	0.0032	3.2
Ball screw	5	0.0125	12.5

For the future team they should most definitely replace the y and z-axis threaded rods with precision ball screws. For the z-axis the currently purchased 450 mm length ball screw assembly will work, it was just placed as a lowest priority since the z-axis has so little affect on the scan. As for the y-axis, the currently purchased 650 mm length ball screw assembly does not fit directly onto the frame of the y-axis. Since the end supports must fit directly onto the frame there are two choices. Increase the width space of the x-axis slide system to make the y-axis frame longer or purchase a custom length ball screw assembly. I recommend the first option as it is cheaper, faster, and they already have the assembly. They can measure the exact length of the current assembly and make a more exact model for designing the frame and motor mounts. Should they order a custom length ball screw they run the risk of not having a good CAD model, if at all, or ordering the wrong size.

Supported Slide Rails

Both the x and y-axis of the system are set on supported slide rails as seen in Fig. 4 (right). The z-axis uses only standard slide rails as seen in Fig. 4 (left) but they do not bear a load perpendicular to the length as they are set in the vertical position.

The previous team had wanted to use supported slide rails on both the x and y axis but due to budget issues was only able to do so on the y-axis where they believed a bend in the rail would be more detrimental since it directly supported the central unit of the system. The product was standard slide rails on the x and z-axis with supported slide rails on the y-axis.

It was easy to see that there was a drastic bend along the length of the x-axis standard slide rail but that the y-axis supported slide rail exhibited no such bend. We also saw that the x-axis had more room for continuous support compared to the y-axis so supported slide rails on the x-axis would hold an even heavier load before bending. We noted that the z-axis did not require replacing since they are placed vertically and never take on a load perpendicular to the center.

We bought supported slide rails of the same length and type of the y-axis on Amazon at a fraction of the price for $122.74. We already had a larger budget than the previous team but this cemented the decision of replacing the x-axis. We then replaced the x-axis standard slide rails with supported and removed the bend that was previously present. We left the z-axis as standard slide rails.

Using the equation in Appendix A we find that the supported slide rail is 5 times stiffer than the standard slide rail, not including the support itself. This is due to an increase in diameter from 8 mm for the standard slide rail to 12 mm for the supported slide rail. With the support included, the load will be continuously distributed to the frame of the system as it will have constant contact to the slotted extrusions on the x-axis. This will further reduce any bending in the supported slide rails to negligible amounts. Replacing the x-axis with supported slide rails did remove the noted bend in the system that would have created an error in both position and waveform during a scan. We did not replace the z-axis because the load is not perpendicular to the length of the slide rail but instead completely parallel. Replacing it would have been an unnecessary hassle with no visible gain. We had also considered using standard slide rails with just a larger diameter. This could have potentially worked, but the supported slide rails are guaranteed to work with minimal deflection and will hold up over time.

For future teams the only consideration is keeping the slide rails, particularly the supported, lubed and greased so that rust does not form on the rail or the linear ball bearings.

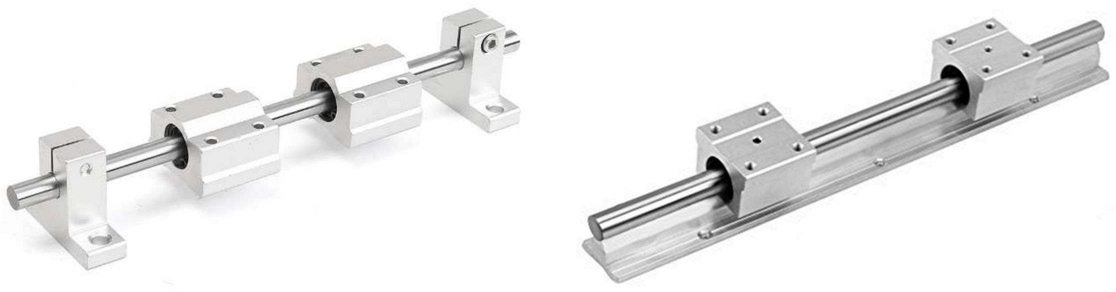


Figure 3. Standard unsupported slide rails (left) similar to those currently used on the system and supported slide rails (right) similar to what will be used on the system.

Supported Slide Rail Bearings

We use an open linear slide bearing which has a cutout for the supported slide rail as opposed to the standard slide rail bearings that fully encapsulate the rail.

The previous team used a common and cheap slide rail bearing for standard slide rails. They are a hemisphere aluminum bracket with a hole though them lined completely with small ball bearings that fully enclose the standard slide rail.

We saw that the linear slide rail bearings used on the standard slide rails cannot be used for the supported slide rails. This is because the supports run continuously down the length of the rail and the linear slide rail bearings used for the standard rail are fully circular.

We decided to purchase the pack of slide rails that came with their own supported slide rail bearings. Other that having a cut of the bottom supports they function the same as the standard slide rail bearing previously on the system.

When looking into the specific type to purchase, we decided that the linear ball bearing was sufficient. The other option considered was the plain bearing. The ball bearing is not as strong as the plain bearing, but it is strong enough for what we need for this system. Plain bearings have a smooth surface while ball bearings have small balls that contact the surface. Because these balls make little contact with the surface, they have a much lower coefficient of friction. Less friction means our motors will not have to work as hard to move the system. The goal is to try to get the system to run as smoothly as possible which can be done if there is less friction interfering with movement. However, the low contact is also what weakens them to higher loads which can lead to deformation. When looking at the specifications of each bearing, we found that the dynamic load for the ball bearing was 410 N. This translates to roughly 41.81 kg. The weight of the upper portion of the system that will rest across four of these is less than this so the ball bearings will be sufficient in terms of load bearing capacity. The linear ball bearings for the supported slide rails were chosen as they met the needs of our system. The ball bearings are also the less expensive of the two options. The 12 mm supported slide rails and the linear ball bearings will cost $184.11 total.

For future teams the only consideration is the same as the supported slide rails. Keep the slide rails, particularly the supported, lubed and greased so that rust does not form on the rail or the linear ball bearings.

Redesign of 3D Printed Parts

Multiple parts for the system have been redesigned for the current system, some have been made obsolete. This includes the linear slide rail connectors, slide rail bearing connector, end support connectors, the ball screw platform connector, and motor bracket mount.

The previous team was limited in material, printing capabilities, and what they could buy to customize their system. As such, anything that did not directly connect to the system via the aluminum extrusions was attached with simple rectangular 3D printed adapters. The designs were purely functional.

A part used to attach the upper portion of the system to the slide rail bearings needed a few changes in dimensions since different bearings were used. Other pieces used to attach the end support of the threaded rods on the x-axis and the y-axis to the system frame needed to be adjusted for the use of end supports made for the ball screw assembly. The floating support close to the motor needed a new 3D printed part to attach it to the frame. Since no floating support was use in the original design, this part had to be created. Any parts that were used to encase a nut on the threaded rods will no longer be used. We instead use a piece that connects each axis to the ball nut housing of the ball screw assembly. The 3D printing surrounding the motor of the z-axis need not be changed in terms of design.

We took measurements of the noted observation and made multiple SolidWorks models. With the use of the partial SolidWorks model, we had accurate measurements needed to redesign these parts. The measure tool has proved very useful for this.

Besides strict dimensioning, not much analysis of this needed to be done. Due to some uncertainty, we refrained from 3D printing these parts until we acquired the ball nut assembly and the supported slide rails. All the models were set to be adjusted if we found that there was an error somewhere along the way or there were differences between the parts ordered and what was used in the model. One part that is crucial to the system was remodeled and the differences can be seen in Fig. 5.

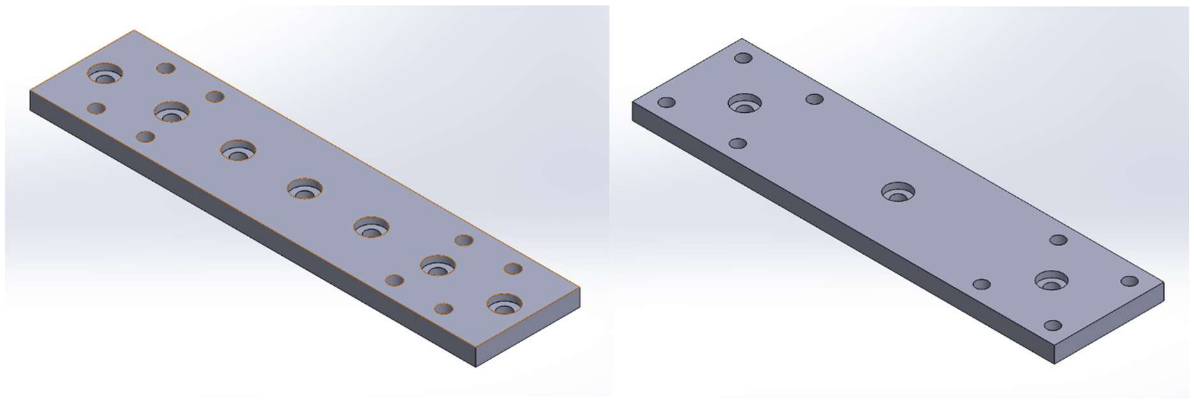


Figure 5. The 3D printed part originally used to attach the upper portion of the system to the slide rail bearings of the x-axis (left) was redesigned to fit the new slide rail bearings.

All SolidWorks models will be left on the machine for future teams to have and edit. This is in conjunction with the sketches in appendix C.

3D Printing

The redesigned 3D parts were printed on an Ultimaker 3D printer using PLA filament using a stronger infill.

The previous team printed their parts with PLA filament on an unknown printer. They used thin walls with a small infill so their parts broke easily and were deformed in many places.

With many critical components of the system broken we were initially unable to test the system and needed to fix parts that kept the structure upright. We also noted that the parts were too easily bent and broken and needed to be strengthened

Without access to the original part files from the previous group, each part had to be redrawn from scratch. They were all measured by hand and remade in SolidWorks. We decided to print only the parts necessary for the original system to function for now. They were printed on an Ultimaker 3D printer using PLA filament.

We choose to print in PLA because it is the same material used by the previous team and we knew it would be sufficient for testing. We did not print in a better material such as Onyx because it is considerably more expensive and the parts were subject to change. We instead opted to wait until the project is completely finished and it is certain that dimensions will not change before reprinting parts in Onyx. Below Fig. 6 shows a theoretical stress analysis of one the support parts using SolidWorks. This is the part that will be supporting much of the weight on the slide rails. Note that the square supports on the underside of the part were fixed pieces made to simulate the slide rail it will be sitting on, and thus will not be printed. As the exact forces are unknown at this time, an arbitrary uniform load of 5 Newtons was used in the simulation just to get an idea of the locations of high stress that will need to be reinforced. Aside from reinforcing the walls and the holes where it will be bolted to the rest of the system, there is stress concentrated at the edges of the supports. It would be prudent to reinforce the parts around those edges, as well as in the middle where there are no supports.

When the system is finished at the end of the project, a stronger material will be used. It is recommended to use a material from Markforged known as Onyx. Onyx is a unique micro carbon filled nylon that boasts high strength and toughness, with a higher flexural strength than regular nylon or ABS. Another version of Onyx called Onyx FR could be considered, but is largely similar, except for higher heat resistance and higher price.

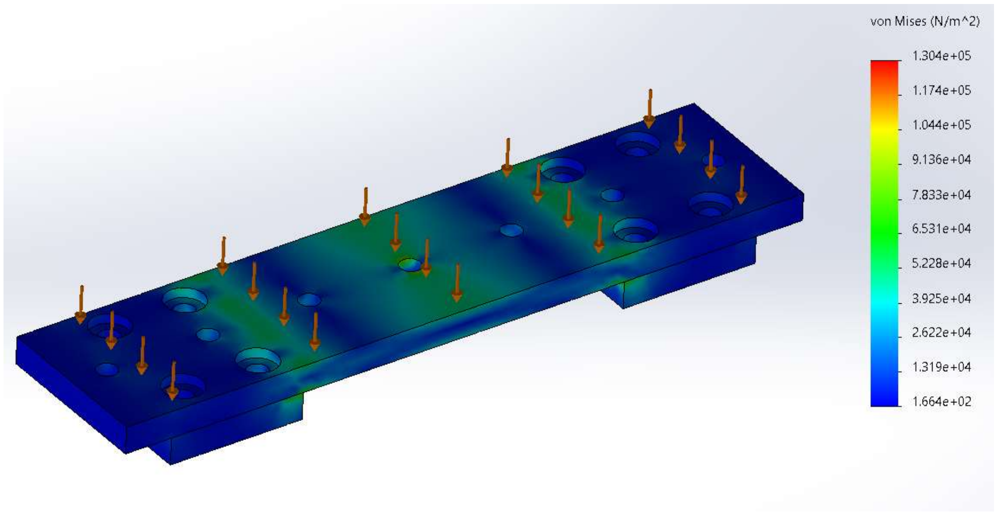


Figure 6. Stress analysis of the 3D printed part that attaches the upper portion of the system to the slide rail bearings.

Before parts are printed in Onyx, the internal design of the parts will need to be finalized. When printing with Onyx, the Markforged printer the school owns would allow one to modify the parts individual layers, as well as reinforce the part with ether concentric fiber around the walls and any holes or with isotropic fiber throughout the entire part. Some parts will not need either of these, but the base pieces that support the system will. Some analysis still needs to be done to find a good balance between part strength and price, especially since the other aspects of the project will be taking a substantial portion of the budget. This analysis is being performed through the Markforged Eiger website. The website allows one to submit the file of a part and then calculates the dimensions, mass, volume, and material cost of the part.

Electrical

Stepper Motors and Drivers

The system is powered by four bipolar NEMA 17 0.9° stepper motors each controlled by a single A4988 stepper motor driver. The drivers use 5V logic voltage and 12V motor voltage with the variable potentiometer on the driver setting the motor current at 1.6A. The drivers accept a step input and a direction input from the microprocessor.

It was the previous team who chose the stepper motors currently in use and who chose the previous set of A4988 stepper motor drivers that have since been updated. They choose these stepper motors because by the nature of how stepper motors operate, each pulse or step turns the ball screw exactly 0.9°. That’s 400 steps per revolution which is what gives us the 12.5 µm per step that we desire. Stepper motors are a common design choice for systems require precision and or torque. Inspiration could be found from observing 3D printer systems. Stepper motor drivers are commonly used as they reduce the number of wires needed from the microprocessor freeing up pin outs, they help regulate current, and allow for a secondary voltage source that actually runs the motors reducing the load of the system. The A4988 stepper motor driver is a common choice and the team used the one available to them at the time.

I tested the stepper motors by reversing their power flow to operate them as a generator and measure their output with a multimeter. They operated normally so when I had issues running the motors from the previous microcontroller, I assumed that there was an issue with the previous stepper motor drivers. Tests revealed that the drivers had been burnt out or damaged at some point and required replacement. I also saw that there were several circuit elements that had no understood purpose and with no known documentation about their purpose I assumed they had become obsolete during the previous team's design process. I also considered whether the current stepper motors would be powerful enough to turn the larger ball screw rods as well as move the system.

I kept the previous team's stepper motors, replaced the old A4988 stepper drivers with a new version rated for higher currents, did away with unnecessary electrical components, replaced the 60W PC power supply with a more manageable 12 V 2A (24W) DC power supply and regulated it with a 100 μF capacitor.

A 60W power supply was unnecessary, harder to regulate, and harder to integrate into the circuit. The new DC power supply is similar to a phone charger and easier to use. Should a new team come and find that they prefer to use a larger power supply for all components; microprocessor, motors, and Picoscope; I kept the old 60W power supply. The old stepper motors appeared to have been damaged by high current so when selecting a newer version of A4988 stepper motor drivers I selected one that had a better potentiometer and rated for higher currents. The only concern I had about keeping the same stepper motors was the amount of torque the new ball screw rods would require. They could be roughly calculated by assuming their rotational inertia and estimating the angular acceleration. This is difficult to calculate without the specific rotational inertia of the rods and acceleration of the motors, which is not at my disposal until the rods were installed. However, due to the similarities to the old rods, the torque changes are negligible because the moment created by the new coupler is assumed to be very similar. The torque necessary is also likely significantly lower for the new rods despite their weight increase because the pulse rate per second required to overcome friction in order to startup the motor will be lower with the new precision rods, due to their design and limited friction compared to the threaded rods on the existing device.

Moving forward there is foreseeable change need to these components. They only design consideration we have made was saving the old power supply incase a new team found that it would be easier to have the motors and Picoscope powered from the same source. A Picoscope was never in the scope of our project so we did not want to limit any future choices.

Microcontroller and Chip

The system is now controlled by a Raspberry Pi 4 with 32GB of upgradable storage and 8GB of RAM. A Raspberry Pi is a type of microprocessor that runs Linux and is a computer complete with a power supply, keyboard, and mouse in, display screen out, a full pin out array for controlling circuit elements, and multi core processor capable of threading for running code side-by-side real time.

The previous team used a custom SAMduino with an Atmel ATSAM4S8B chip, which had 512 KB of memory and 128 KB of RAM.

Originally it appeared that the chip and microcontroller were in good order because the very first time I plugged in the system there was some movement into motors. However upon examination the microcontroller was unfamiliar to our whole team and when we hooked it up to a PC, I was unable to access the files on it or upload new files to make changes. After some time I was able to get in contact with a previous team member who told us what software was needed to view files on the microcontroller but informed us that the specific bootloader needed to upload files was lost and the machine used to do so was beyond recovery. Their advice was to completely replace the microcontroller as a whole and start from the ground up.

At this point I had lost a lot of time and was only familiar with a standard Arduino Uno so I used one that I had in the lab. I set it up with just the pin outs for controlling the x-axis. Due to a lack processing power, start up lag, communication time, and code compatibility errors I switched to an Arduino Mega that a team member personally owned. There was an increase in processing power but still issues with start up lag, communication time, and code compatibility. I then asked advice from a peer on another project who had significantly more experience in computer integrated controls. Their advice was to completely move away from the Arduino architecture the system had always used and move to a microprocessor, not a microcontroller, specifically a Raspberry PI. I had zero experience with this system but they offered to help set it up and teach me how to run the Linux system. I placed an order for the noted Raspberry Pi but to save time I used one from the IEEE lab until ours arrived. I had some complications setting up the Linux system, not unusual for setting up an OS, but once it was running, I had none of the previous issues. It had ample processing power, dynamic memory, zero start up or communication lag since it was a microprocessor and ran the code on the same core that executed the commands, and since only one language of code was needed, I removed the issue with code compatibility. I was then able to set up the y and z-axis along with the x-axis as well as wire in the respective limit switches.

There were a lot of design choices and redesigns and each had a subsequent reason. I had to move away from the custom made SAMduino because it was wholly unfamiliar and due to the loss of equipment and software unusable. The code on it was outdated and would absolutely need to be changed and if I could not do that then it had to be replaced. I moved to a standard Arduino Uno partially out of familiarity, since I had only ever had experience with that, but mostly out of convenience. Not being able to get an output was holding up code development and testing and I could not afford to wait for shipping times. So I used one that I had on hand. For reasons to be later discussed in the code section I had multiple errors with code compatibility and the microcontroller simply was not strong enough to run the code and move the motors real time. I was missing about 20% of steps because the code was running faster than the controller could send commands. Also, due to the nature of a microcontroller the main code had to be run on a different machine and communicated to the controller through a serial port which produced a significant latency for control. Switching to an Arduino Mega seemed like an obvious choice since I believed that most of the problem lied in processing power and a Mega would have had enough. The Mega did have sufficient processing power, only about 25% of the dynamic memory was being used now as compared to the previous

100% on the Uno. I still experienced a start up lag where the motors would mysteriously run during the 3-5 seconds that the serial port was opened up for communication. This was unacceptable since it would mess up positioning for a scan. The Mega would also not run the motors at the expected speeds. The delay between steps was always longer than programed and would not scale linearly when increasing or decreasing the delay. This would make it hard to give the user control or scan speeds. I believed that this was due to the time taken for the code to be sent to the controller through the serial port and then also for the microprocessor to interpret the commands. The answer was to eliminate the communication issues by running the code on the same system that controlled the electrical components. A Raspberry Pi is a microprocessor, not just a microcontroller. It is essentially a full Linux computer that I can develop code on. There was no need for two types of code, one on the controller and one on the machine, but just one that assumed full control. The Pi also had significantly more memory at 8 GB of RAM which not only allows me to easily move all four stepper motors but also gives me plenty of room to incorporate the Picoscope control in the Pi which would make it easier to time and record scans.

For future teams I would heavily advise not changing the microprocessor. It is adaptable, has plenty of room for adaptations, easily upgradeable as it is a full Linux OS, and only requires one language of code. I thoroughly believe that the current design choice is a final one.

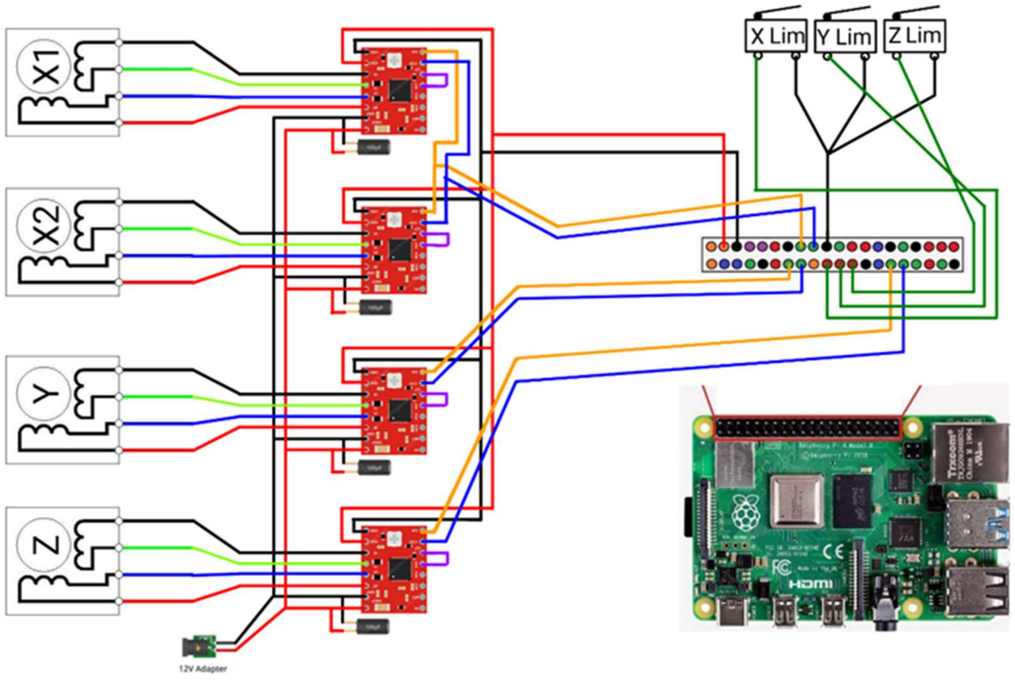


Figure 7. Wiring schematic for the system including Raspberry Pi 4, four A4988 stepper drivers, four stepper motors, four 100 μF capacitors, three limit switches, and a 12 V DC voltage source.

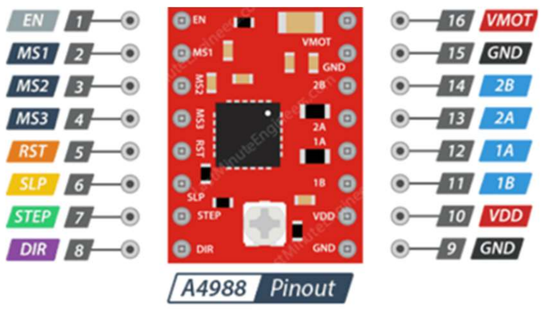


Figure 8. Pin layout of the A4988 stepper motor driver

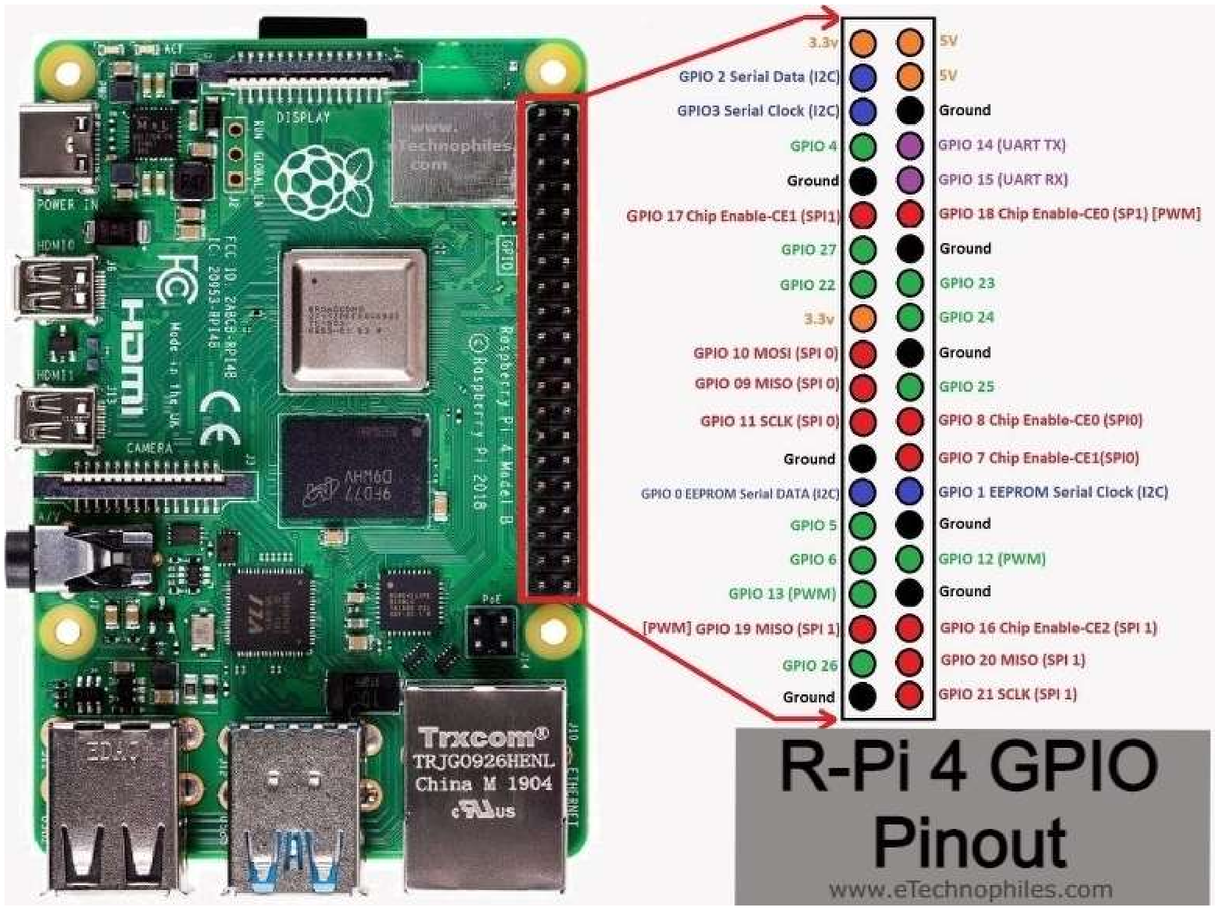


Figure 9. Pin layout of the Raspberry Pi 4 microprocessor

Encoder

There are no encoders currently present on the system nor have some been purchased and stored as future parts.

Encoders were a minor consideration from the previous team about how to make the system closed loop with locational feedback making it more precise. No analysis was done by the previous team on what type or how to integrate them.

I saw that a lack of encoders could be a major error down the line when the system is used for scans. As it is the system is simple, the code knows how many steps the motors need to take to get the transducer at the spot it needs to be and the Pi sends that many pulses to the drivers and subsequently the motors. Just because the Pi tells the motors to take 400 steps (one full turn) does not mean that the motors take 400 steps. If for some reason there is sufficient friction acting against the motors that they can not overcome then, despite the pulse, the motors will not take that step. This means that the code will think that the system has moved further than it actually has and the recorded position will be wrong. This lack of feedback is referred to as an open-loop system. What an encoder would aim to do is provide some location feedback making the system closed-loop.

Encoders do this by measuring how much the ball screw rod has turned. If it turned 0.9° then you know the motor took 1 step or the system moved 12.5 μm. There are many types with different levels of precision but since the encoder can take as little as 0.9° steps you need an encoder that can measure 0.9° of a turn at the least but 0.45° precision is optimal. I also set up the code in a fashion so that there are three ways to implement it into the code. You can either include it within the step loop having it check the position of the encoder with each step, make the measured steps from the encoder the end condition for the step loop, or you can use multi-threading to have a second line of code run continuously that keeps a real time measurement of the encoders and give it a tolerance of how "off" the encoders can be.

I did not include encoders in the current system for many reasons. The first and largest is that it was not until later in the project that we had a final method of moving the motors figured out so it would have been difficult to design a method that measuring the motors moving. The second reason is because the y and z-axis ball screws have yet to be implemented into the system so they can not even be attached yet. Rather than commit to a design choice that we would not even implement, I left that for a future team so that they do not have yet another choice from a previous team to work around.

For these future teams my only major recommendation is that you purchase an encoder with about 0.45° precision and to have that level precision or even more accurate it will almost certainly have to be a digital component.

Programming

Code

The system runs on Python 3.10 and there are only two code files it uses; the user only needs to run one. The one file that the user has to run is the GUI, it's what opens a graphical user interface (GUI) that the user uses to control the system. The GUI file imports the second file in as a library. That is the utility file, this is the core of controlling the system and contains the functions that zero the system, move it to home, and run the scan. They could be the same file but it is better practice to separate them.

Previously the system ran on a mix of Python 2, some custom divers, and Arduino code. Not much is known about how the custom drivers or Arduino code were set up but I do know that the python code only kept track of position. The python code would send a data packet containing the information of where the system needed to move to, the custom SAMduino would interpret this and move accordingly, then it would send back a packet with the updated location. This meant that the bulk of computation was being done on the SAMduino and it was probably done this way to eliminate the communication issues that we have previously noted. Fig. 10 below depicts the previous teams flow chart for their python code.

One of the major concerns I saw with this is that I could not see anything. The SAMduino was a black box to us that we could not look into or edit. Another major issue is that the system had been built around Python 2 and I was tasked to update all the code to Python 3. The two versions handle number computation differently so even if there were not several other issues with compatibility between the old SAMduino and PC, the code would still demand a change to the SAMduino.

Originally, I just updated the existing code into Python 3. Between the differences in how the two handle bit encoding, as well as number calculations, along with the untouchable code on the SAMduino being made for Python 2; this did not work. When moving to the Arduino Uno I went for the same method the original team used. Send only a byte packet through a PySerial port to the Uno and let it do the work. Not only were there issues in sending multiple string bytes to the Arduino, along with decoding the packet. There was also the issue that the Arduino language is not suited for this type of project. Python 3 was specifically chosen due its utility and convenience but if the only thing I use if for is to send numbers then its effectively a waste. My next idea was to move away from the original code design and use an Arduino library called PyFirmata. PyFirmata is an Arduino/ Python 3 library that allows one to control the inputs and outputs of the Arduino board with Python code. You type the command in Python, PyFirmata on the Python side converts it to something the Arduino understands, it gets sent to the board via PySerial, and then PyFirmata on the Arduino side reads the commands and executes it. PyFirmata was essentially the previous approach I tried but now I had the encoding/decoding issue fixed. PyFirmata caused other issues though such as: increased processing consumption, PySerial set up lag, motor movement during PySerial setup, variable motor speeds due to communication lag, shaky motors due to variable motor speeds, and missing steps due to lack of processing power. When I changed from the Uno to the Mega this fixed the error of processing consumption and missing steps due to processing consumption. The issues of PySerial set up lag, motor movement during PySerial setup, Communication lag due to PySerial, variable motor speeds due to communication lag, shaky motors due to variable motor speeds. All these issues centered around PySerial, or more specifically, having the code run on a PC and commands be executed on a different piece of equipment. That is why I switched to the Raspberry PI, the Linux based system that is a microprocessor, not just a microcontroller. Programing wise this does require know that a user know enough Linux to set up and navigate the microprocessor but that has been done. Code wise the Pi only needs one language since the Pi both runs the code and has the pins to execute the code. This does away with both the processing consumption and limitations of PyFirmata as well as the PySerial relevant errors. The code was now one language and fully Python 3. Commands were condensed and converted to Python 3. The PIGPIO Python library is used to read and write pins and the PySimpleGUI library is a wrapper for Tkinter and is used for the GUI. Respectively there are also two Python files used to run the system, a GUI, and a Utility file. The GUI is what calls the functions from the Utility file and lets the user interact with the system, and the utility file is what does the bulk of computing and pin outs.

As it is this is the furthest the system has been. The GUI is functional and allows the user to zero the system, move it to home, input a specimen size, set a scanning speed, and run a scan. It only moves the system; it does not run a Picoscope but it does include a spot to call a Picoscope scan function within the code. Future teams would need to write the code that would generate, read, and save the ultrasonic waveform to a .csv file along with its current position. They would also need to write code that reads a digital encoder, compares it with where the system should be, and adjust position as appropriate.

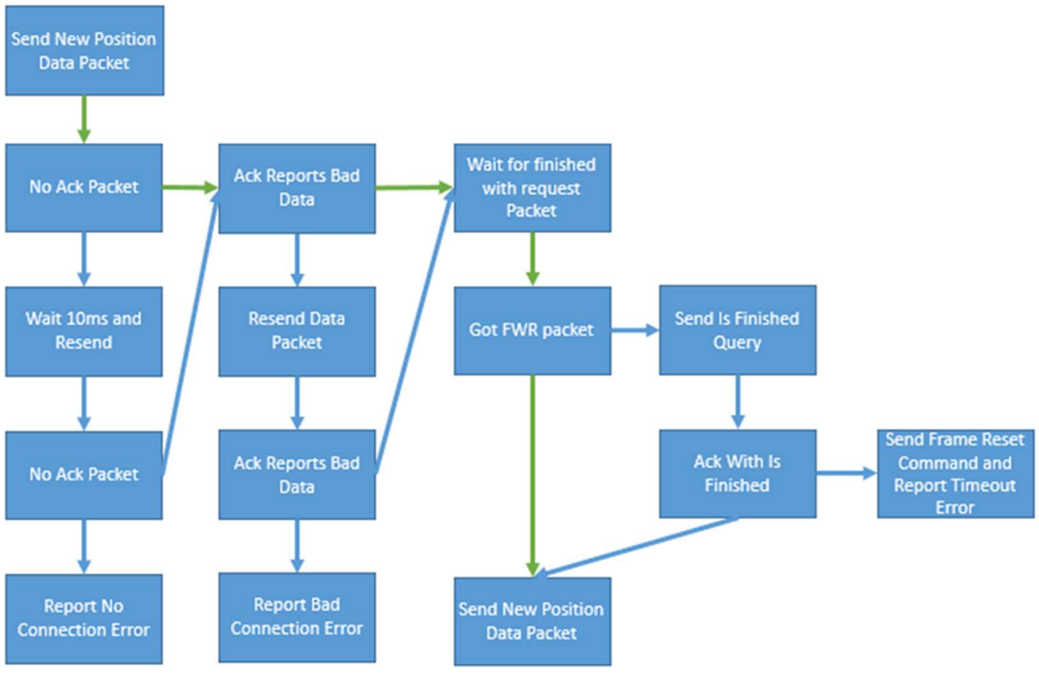

Send New Position Data Packet
No Ack Packet
Ack Reports Bad Data
Wait for finished with request Packet
Wait 10ms and Resend
Resend Data Packet
Got FWR packet
Send Is Finished Query
No Ack Packet
Ack Reports Bad Data
Ack With Is Finished
Send Frame Reset Command and Report Timeout Error
Report No Connection Error
Report Bad Connection Error
Send New Position Data Packet

Figure 10. Previous method for sending a position update with Python.

CONCLUSION

The goal for this project was to extend upon the work of the previous design team and continue the development of the existing system. In order to do this, we focused on making the system more stable and accurate so that measurements done in the future can have as little error as possible. All these additions improve the system and set the course for the next design team to finalize the project. With all these design considerations, the improved mechanical components will increase the precision of measurements made by the system and the new electrical circuit will be compatible with this setup. The 20 mm slotted aluminum extrusions for the base frame were replaced with 40 mm slotted aluminum extrusions. Ball screws were used in place of threaded rods for the x-axis and supported linear slide rails were used in place of unsupported slide rails. All of these changes contributed to an overall increase in rigidity. A change in the electrical circuit was necessary for meeting our end goal of having it run from Python 3 as well as communicate efficiently. We upgraded the system code and changed the microcontroller from a SAMduino to a Raspberry Pi 4 microprocessor. The electrical circuit was completely redone and now uses a 12 V adapter as the main power source and newer versions of the original drivers. The cost for these upgrades totals at $972.57. A detail cost report is included in Appendix B.

Acknowledgments and Thanks to:

Cindy L. Krampe for being an exception teammate, Brian Williams for helping another team set up a Raspberry Pi and Linux, Luke Kamrath for helping the current team despite being from the old team, Kaylea A. Wilson, and Spencer T. Grodi

APPENDIX A │Sample Calculations

Moment of Inertia for 20-mm Aluminum Slotted Extrusions:

$$I = \frac{1}{12}bh^3$$

b_1 = 20 mm	b_2 = 40 mm
h_1 = 20 mm	h_2 = 40 mm
$I_1 = 1.33 \text{ x } 10^{-8}$	I_2=$2.133\text{x}10^{-7}$

$$\frac{I_2}{I_1} = \frac{2.133 \times 10^{-7}}{1.133 \times 10^{-8}} = 16 = 2^4$$

Stiffness of the system increases by a factor of 2^4.

Area Moment of Inertia for Slide Rails Compared:

$$I = \frac{\pi}{4}r^4$$

r_1 = 8 mm	r_2 = 12 mm
$I_1 = 3.217 \text{ x } 10^{-9}$	$I_2 = 1.629 \text{ x } 10^{-8}$

$$\frac{I_2}{I_1} = \frac{1.629 \times 10^{-8}}{3.217 \times 10^{-9}} = 5.0625$$

The supported slide rails are 5 times stiffer based on the increased diameter alone.

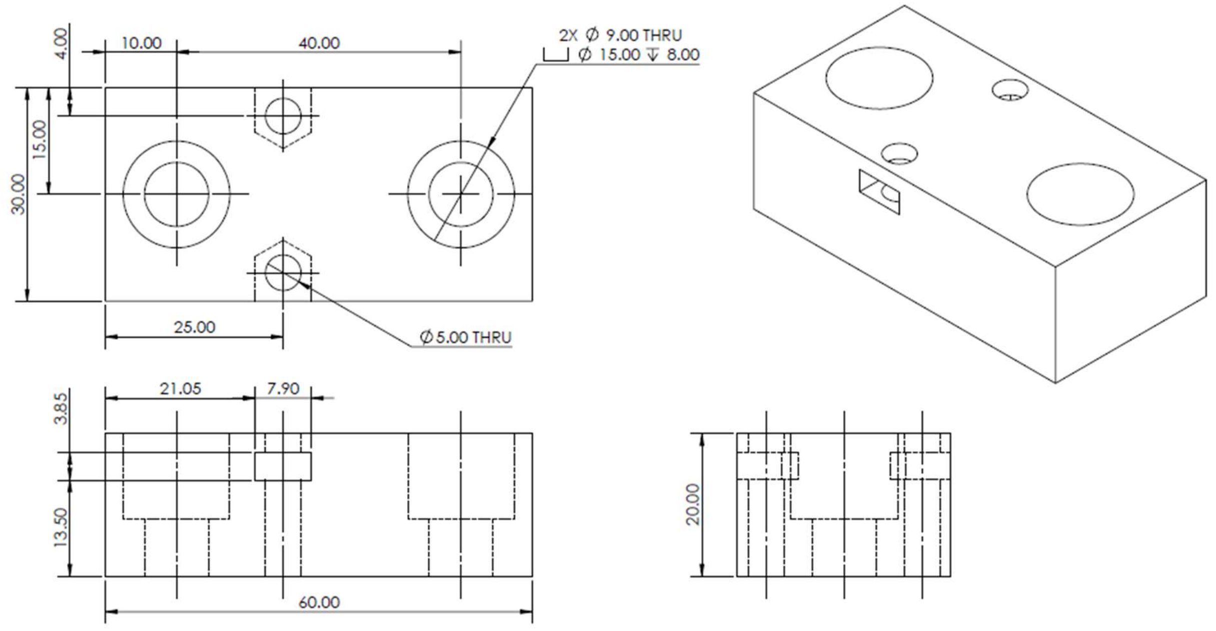
APPENDIX B | Detailed Cost Report

Part	Quantity	Price/Part	Subtotal
Ball Screw Assembly (650 mm)	3	$45.00	$135.00
Ball Screw Assembly (450 mm)	1	$40.00	$40.00
Supported Slide Rails/Bearings	2	$61.37	$122.74
40 Series 6 Hole - Wide 2x4 Inside Corner Bracket	2	$6.45	$12.90
40 Series 4 Hole - Tall Gusseted Inside Corner Bracket	10	$6.25	$62.50
M6 x 40.00mm Socket Head Cap Screw	24	$0.62	$14.88
M6 Hex Nut	24	$0.13	$3.12
M5 x 12.00mm Button Head Socket Cap Screw	32	$0.25	$8.00
M5 x 16.00mm Flat Head Socket Cap Screw	28	$0.28	$7.84
M5 Hex Nut	28	$0.12	$3.36
M8 x 16.00mm Button Head Socket Cap Screw	80	$0.28	$22.40
M8 Slide-in Economy T-Nut - Offset Thread	80	$0.32	$25.60
M8 Hex Nut	8	$0.15	$1.20
M8 x 50.00mm Flanged Hex Head Bolt	4	$1.15	$4.60
40mm X 40mm T-Slotted Profile (425 mm)	4	$16.06	$64.24
40mm X 40mm T-Slotted Profile (610 mm)	4	$16.60	$66.40
40mm X 40mm T-Slotted Profile (573 mm)	2	$15.71	$31.42
PLA 3D Printing (in grams)	488	$0.15	$73.20
M5 x 12.00mm Button Head Socket Cap Screw*	50	$0.15	$7.50
M4 x 12.00mm Button Head Socket Cap Screw*	50	$0.13	$6.50
12V AC DC Adapter*	1	$11.99	$11.99
Assorted Capacitors (40 pack)*	1	$7.28	$7.28
A4988 Stepper Motor Driver (5 pack)*	1	$10.99	$10.99
Lucas Oil White Lithium Grease*	1	$4.98	$4.98
6in x 6in x 1/16 in Rubber*	2	$1.99	$3.98
Raspberry Pi 4 Desktop Kit with Display	1	219.95	219.95
		Total:	$972.57

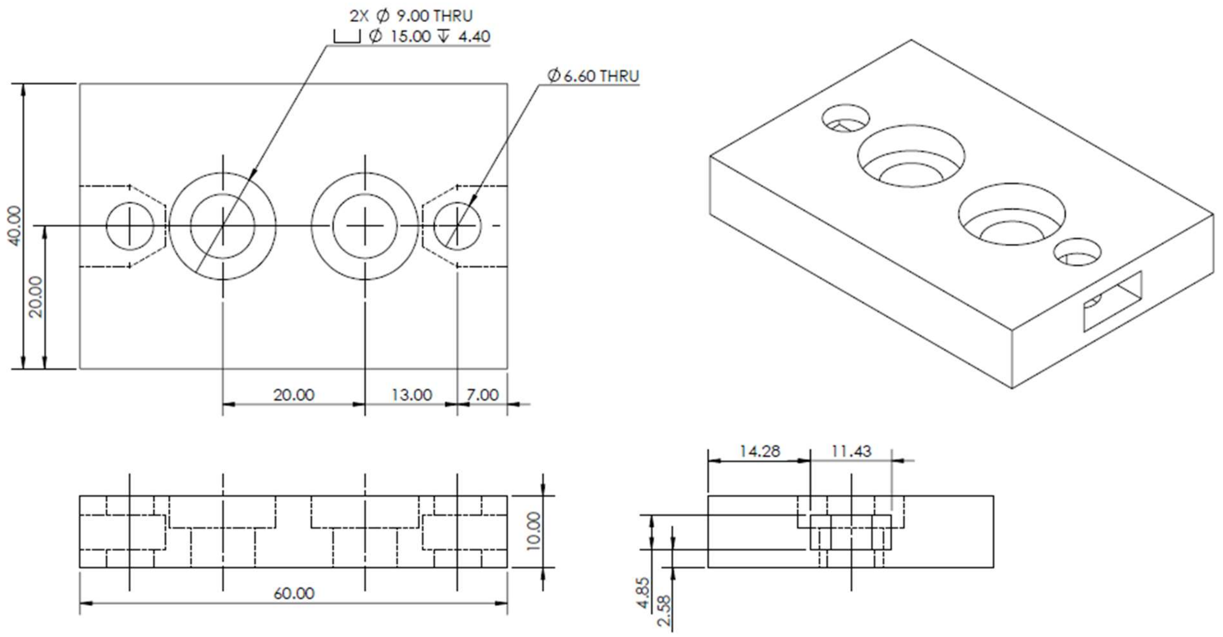
NOTE: tax and shipping not included

*Personally purchased

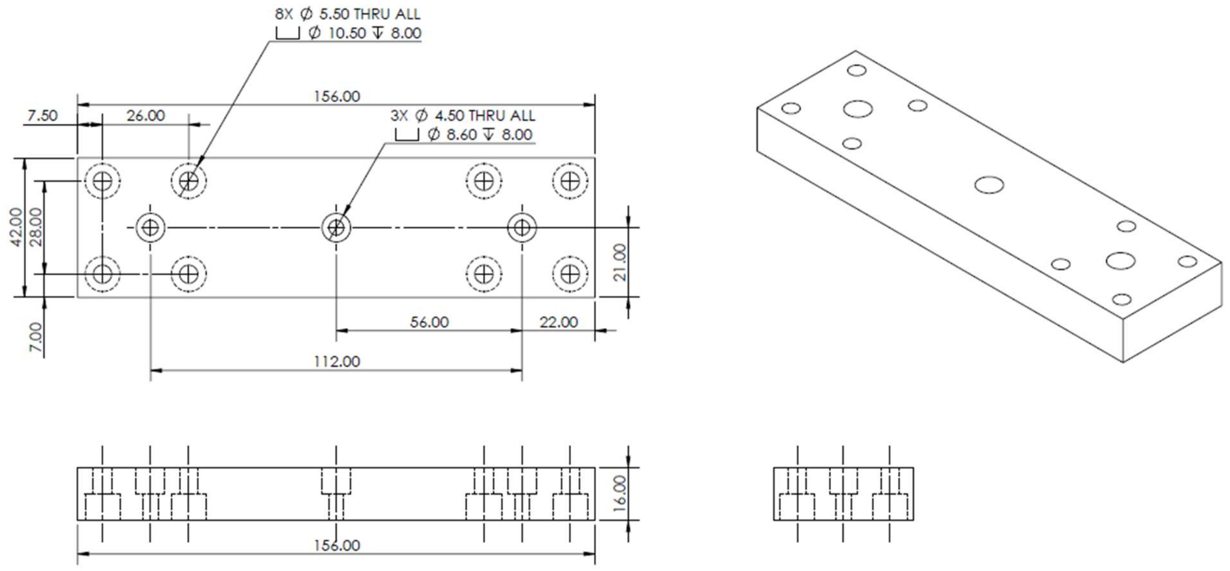
APPENDIX C │3D Printing
Linear Slide Rail Connector



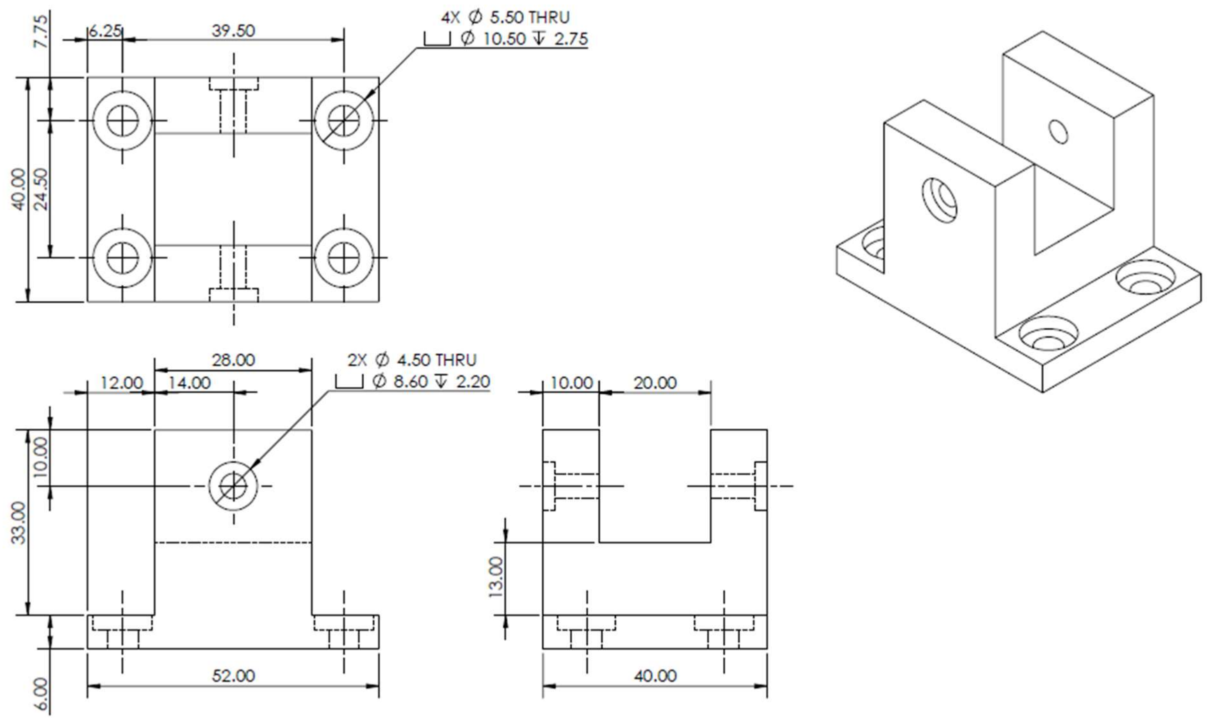
Ball Screw End Support Connector



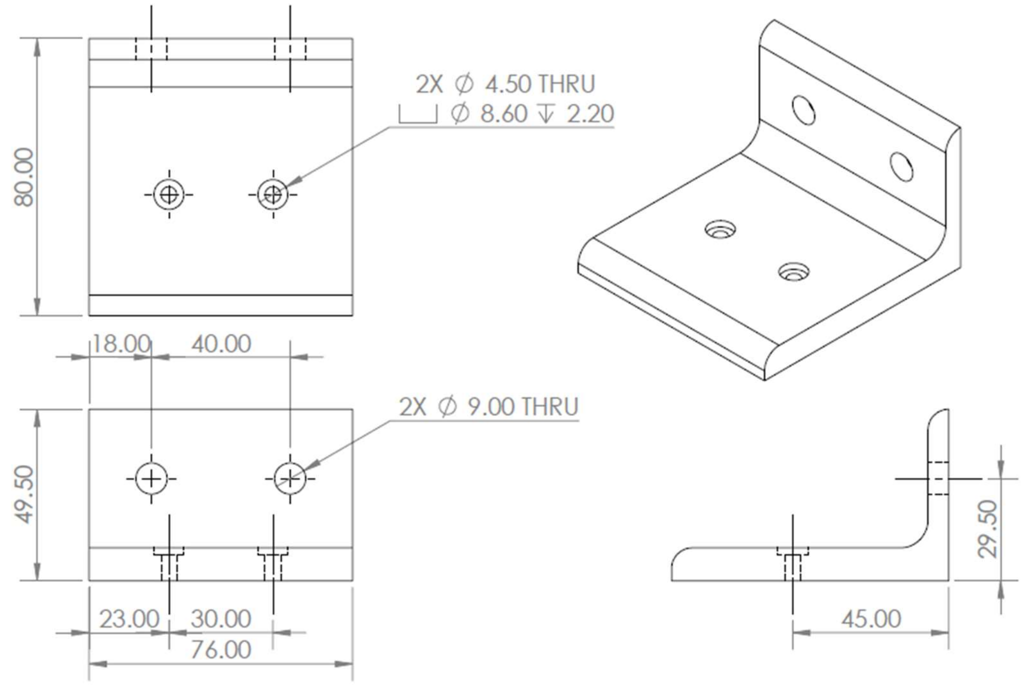
Slide Rail Bearing Connector



Ball Screw Platform Connector



Motor Shelf



APPENDIX D | Code
GUI Code

```
import PySimpleGUI as sg
from NDE_Stage_Utility import * # my custom stage library


##################################################################################
#	Before startup, in terminal, run:
#	sudo pigpiod
##################################################################################


sg.theme('BluePurple')


layout = [[sg.Text('Always zero machine on startup'), sg.Button('Zero Machine',
	pad=(10,0))],
	[sg.Text('Home Position (cm):		x		',
      'y                z')],
[sg.Canvas(size=(150,1)), sg.Input(size=(10,1), key='-xHome-',
 justification='center'), sg.Input(size=(10,1), key='-yHome-',
 justification='center'), sg.Input(size=(10,1), key='-zHome-',
 justification='center'), sg.Button('Move', pad=(10,0))],
[sg.Text('Specimin Size for Rectangular Scan (cm):   x Length',
 '        y Length')],
[sg.Canvas(size=(255,1)), sg.Input(size=(10,1), key='-xLength-',
 justification='center'), sg.Input(size=(10,1), key='-yLength-',
 justification='center')],
[sg.Text('Physical Distance between scan (mm):'),
 sg.Input(size=(10,1), key='-ScanWidth-', pad=(2,0),
 justification='center')],
[sg.Text('Scan speed (mm/s):'), sg.Canvas(size=(110,1)),
 sg.Input(size=(10,1), key='-ScanSpeed-', justification='center'),
 sg.Canvas(size=(74,1)), sg.Button('Scan', pad=(10,0))],
[sg.Canvas(size=(1,40))],
[sg.Text('File Name of Scan: '), sg.Input(size=(50,1),
 key='-FileName-')],
[sg.Text('Where to Save File:')],
[sg.Text('Your Folder', size=(15, 1), auto_size_text=False,
 justification='right'),
 sg.InputText('Default Folder'), sg.FolderBrowse()],
[sg.Button('Save')],
[sg.Canvas(size=(1,20))],
[sg.Text('System Output:')],
[sg.Text(size=(60,1), key='-OUTPUT-', background_color = 'white')],
[sg.Button('Exit')]]
```

```
window = sg.Window('NDE Ultrasonic System', layout)

# Initialize variables
MachineZeroed = False
MachineHomed = False
pos = [250, 250, 250]

while True:  # Event Loop
    event, values = window.read()
    # print(event, values)
    if event == sg.WIN_CLOSED or event == 'Exit':
        break

    if event == 'Zero Machine':
        pos = StartUpZero()
        while pos[0] != 0 or pos[1] != 0 or pos[2] != 0:
            pos = StartUpZero()     # Keep running startup zero until zeroed
        window['-OUTPUT-'].update('The stage is zeroed')    # Print to GUI that
        MachineZeroed = True                                 # stage is zeroed

    if event == 'Move':
        if MachineZeroed == False:
            window['-OUTPUT-'].update('You must zero the machine upon startup')
        elif pos[0] == 250 or pos[1] == 250 or pos[2] == 250:
            window['-OUTPUT-'].update('You must enter coordinates to move to')
        else:
            # Call function to move to Home position
        if (values['-xHome-'] == '' or values['-yHome-'] == '' or
            values['-zHome-'] == ''):
            window['-OUTPUT-'].update('You must enter coordinates to move')
        else:
            xn = float(values['-xHome-'])*0.01  # Convert string cm
            yn = float(values['-yHome-'])*0.01  # to float m home values
            zn = float(values['-zHome-'])*0.01  #
            Str = ('Moving to (x, y, z) cm:     (' + str(xn/0.01) + ', ' +
                    str(yn/0.01) + ', ' + str(zn/0.01) +')')
                    # Display moving position in cm
            window['-OUTPUT-'].update(Str)

            MoveToHome(xn, yn, zn, pos[0], pos[1], pos[2])
            pos = [xn, yn, zn]
            Str = ('Moved to (x, y, z):     (' + str(pos[0]/0.01) + ', ' +
                    str(pos[1]/0.01) + ', ' + str(pos[2]/0.01) +')')
                    # Display moved position in cm
            window['-OUTPUT-'].update(Str)
            MachineHomed = True


    if event == 'Scan':
        if MachineZeroed == False:
            window['-OUTPUT-'].update('You must zero the machine upon startup')
        elif MachineHomed == False:
            window['-OUTPUT-'].update('You must home the machine to start scan')
        else:
            xL = float(values['-xLength-'])*0.01
        yL = float(values['-yLength-'])*0.01
        ScanWidth = float(values['-ScanWidth-'])*0.001
        speed = float(values['-ScanSpeed-'])*0.001
        # Convert string cm/mm to float m home values
        RectScan(xL, yL, ScanWidth, speed)
window.close()
```

Utility Code

```
import pigpio
import time

pi = pigpio.pi()

# Enable pigpio on command line prior to running code using ~~> sudo pigpiod
# Disable after code using ~~> sudo killall pigpiod

# 23 - x Dir        27 - y Dir       5 - z Dir       Orange
# 24 - x Step       22 - y Step      6 - z Step      Blue
# 10 - x Limit      9 - y limit      11 - z limit    Grey

yzdelay = 0.00025
xdelay = 0.0025
# x Direction
#       0 ~~> Clockwise ~~> towards motor
#       1 ~~> Counter   ~~> away
# yz Direction
#       1 ~~> Clockwise ~~> towards motor
#       0 ~~> Counter   ~~> away
# Precision and Threaded leads
#       12.5 um / step        1 step = 0.0000125 m
#       3.175 um / step       1 step = 0.000003175 m
PreStepConv = 0.0000125
TheStepConv = 0.000003175

def StartUpZero():
    pi.set_mode(10, pigpio.INPUT)          # Set pin 10 as input
    pi.set_pull_up_down(10, pigpio.PUD_UP)  # Set default 10 as up/high/1/True
    pi.set_mode(9, pigpio.INPUT)           #  \
    pi.set_pull_up_down(9, pigpio.PUD_UP)   # -- y axis
    pi.set_mode(11, pigpio.INPUT)          #  \
    pi.set_pull_up_down(11, pigpio.PUD_UP)  # -- z axis

    pi.write(23, 0)     # x Dir low ~~> towards motor
    pi.write(27, 1)     # y Dir high ~~> towards motor
    pi.write(5, 0)      # z Dir low ~~> away motor/ up

    # Zero X
    input_state_x = True      # Switch not pushed
    while input_state_x == True:  # While switch not pushed
        pi.write(24, 1)         # Set local pi's GPIO BCM 10 high
        time.sleep(xdelay)
        pi.write(24, 0)         # Set local pi's GPIO BCM 10 low
        time.sleep(xdelay)
        input_state_x = pi.read(10) # reads 1/True/not pushed or 0/False/pushed
    # Zero Y
    input_state_y = True
    while input_state_y == True:
        pi.write(22, 1)
        time.sleep(yzdelay)
        pi.write(22, 0)
        time.sleep(yzdelay)
        input_state_y = pi.read(9)

    # Zero Z
    input_state_z = True
    while input_state_z == True:
        pi.write(6, 1)
        time.sleep(yzdelay)
        pi.write(6, 0)
        time.sleep(yzdelay)
        input_state_z = pi.read(11)

    # Return zeroed position
    if (input_state_x == False and input_state_y == False and
        input_state_z == False):
        return [0, 0, 0]

def MoveToHome(xn, yn, zn, Xc, Yc, Zc):
    # Moves sensor from zero to home to start scan
    # xn, yn, zn are distances in m of the point from zero that we move to
    # Xc, Yc, Zc are the current position of the sensor
    # Calc x move dist & dir
    if xn > Xc:     # Move away from zero/motor
        x = xn - Xc
        pi.write(23, 1)     # x Dir ~~> away zero/motor
    else:           # Move towards zero
        x = Xc - xn
        pi.write(23, 0)

    # Calc y move dist & dir
    if yn > Yc:     # Move away from zero/motor
        y = yn - Yc
        pi.write(27, 0)     # y Dir ~~> away motor
    else:           # Move towards zero
        y = Yc - yn
        pi.write(27, 1)

    # Calc z move dist & dir
    if zn > Zc:     # Move away from zero and towards motor
        z = zn - Zc
        pi.write(5, 1)      # z Dir ~~> towards motor and away from zero
    else:           # Move towards zero and away from motor
        z = Zc - zn
        pi.write(5, 0)

    xStep = round(x/PreStepConv)
    yStep = round(y/TheStepConv)
    zStep = round(z/TheStepConv)
```

```
    # Move to x home
    for x in range(xStep):
        pi.write(24, 1)        # Set local pi's GPIO BCM 10 high
        time.sleep(xdelay)
        pi.write(24, 0)        # Set local pi's GPIO BCM 10 low
        time.sleep(xdelay)

    # Move to y home
    for y in range(yStep):
        pi.write(22, 1)
        time.sleep(yzdelay)
        pi.write(22, 0)
        time.sleep(yzdelay)

    # Move to z home
    for z in range(zStep):
        pi.write(6, 1)
        time.sleep(yzdelay)
        pi.write(6, 0)
        time.sleep(yzdelay)

def RectScan(xLength, yLength, ScanWidthMeters, speed):
    # All values in m, m, m, m/s

    XScanWidth = int(ScanWidthMeters/PreStepConv)    # - Convert scan width
    YScanWidth = int(ScanWidthMeters/TheStepConv)    #   from m to steps

    xStep = round(xLength/PreStepConv)      # Convert x length from m to Steps
```

```
xIndent = int(round(xStep/XScanWidth))  # number of times to indent x for
yStep = round(yLength/TheStepConv)      # new line of scans
yIndent = int(round(yStep/YScanWidth))

xScandelay = (PreStepConv/speed)/2
yScandelay = (TheStepConv/speed)/2

xDir = 1
pi.write(27, 0)     # y Dir ~~> away motor
for indent in range(yIndent):
   for y in range(YScanWidth):
      pi.write(22, 1)
      time.sleep(yScandelay)
      pi.write(22, 0)
      time.sleep(yScandelay)
   pi.write(23, xDir)     # x Dir ~~> initial away motor
   for x in range(xIndent):
      for x in range(XScanWidth):
         pi.write(24, 1)
         time.sleep(xScandelay)
         pi.write(24, 0)
         time.sleep(xScandelay)
      time.sleep(1)      # This time would be taken to scan/record
                         # Insert Scan function here
   if xDir == 0:
      xDir = 1
   else:
      xDir = 0
```