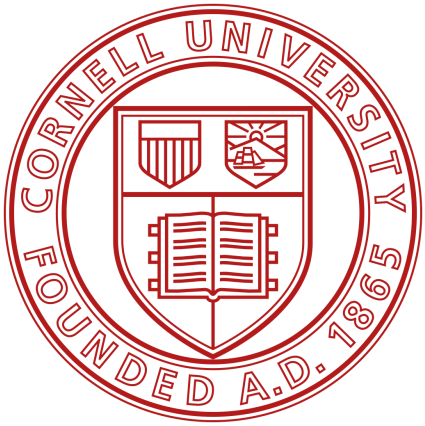
HARDWARE ACCELERATED AUTONOMOUS MOBILE ROBOT

A Design Project Report
Presented to the School of Electrical and Computer Engineering of Cornell University
In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering, Electrical and Computer Engineering



Submitted by:
Boxuan Ai
Peng Peng
Tian Qiu
Kowin Shi
Tiange Zhao

MEng Field Advisor: Bruce Robert Land
MEng Outside Advisor: Matt Ulinski
Degree Date: May 2020

Abstract

Master of Engineering Program
School of Electrical and Computer Engineering
Cornell University
Design Project Report

Project Title: Hardware Accelerated Autonomous Mobile Robot

Authors: Boxuan Ai, Peng Peng, Tian Qiu, Kowin Shi, Tiange Zhao

Abstract: This project aims to create a mobile robot manipulator platform from scratch, and to develop all custom hardware and software designs. To achieve this goal, a team of ECE and MAE students have been assembled to accomplish all of the hardware and software engineering, as well as manufacturing. The final deliverable is a low-cost mobile robot that has a 6 DOF arm, capable of high speed SLAM, as well as multi-sensor navigation and pick-and-place. This is significant because there has not been open-sourced designs of similar form factor, and robotics enthusiasts can build from our design to have their own manipulator platform (or vacuum robot) for a fraction of the cost of the Kuka Youbot ($30k). The hardware acceleration aspect comes with the main high level computer being the Nvidia Jetson Nano that has a 128-core Maxwell GPU, as well as the integration of FPGAs for various tasks.

Executive Summary:

For our MEng Design Project for the School of Electrical and Computer Engineering, we worked on developing the hardware and software for a small but powerful open source autonomous mobile robot platform. The requirements are that it has to be relatively cheap to build, have enough computation power for simultaneous localization and mapping (SLAM) and computer vision, as well as carry enough stored energy for powering a ~200W electrical device. There are a few applications that this robot is aimed for. The first is a fully custom robot vacuum to perform cleaning tasks like that of the Irobot Roomba. This was inspired by our personal experiences with cleaning robots, which was less than satisfactory due to the simple sensors used and the unsophisticated algorithms that cause them to crash into furniture frequently. Therefore, we aimed to solve this problem by using advanced sensors such as a Lidar, with a customizable suite of navigation and path planning algorithms written under the Robot Operating System (ROS) framework. The second is an affordable manipulator platform carrying a 6 degree-of-freedom (DOF) robot arm. The arm was designed by one of the project members previously, and is powered by cheap but accurate serial servos. This platform enables learning and research at a significantly lower budget than before, compared to the Kuka Youbot.

To achieve these goals, we designed the hardware from the ground up. The mobile robot chassis was designed to be completely 3D printed in one piece, and the drivetrain relies on affordable NEMA 17 stepper motors tied to a belt reduction transmission for extra torque. The battery was custom designed and built out of 18650 cells salvaged from a Tesla Model S, and configured in a 4s5p arrangement for a total of 200Wh capacity. The battery management system was purchased, but the electrical drive system as well as the onboard power delivery circuitry were custom designed, and packaged onto a 4-layer printed circuit board (PCB) atop an Arduino Mega 2560 microcontroller. The embedded code is developed in Arduino IDE for accessibility, but is powerful enough to realize a >20hz control loop, hardware health monitoring, and system safety controls. The MCU communicates to the high level computer, the Nvidia Jetson Nano, via serial, both receiving high frequency commands and sending back diagnostic information. On the Jetson Nano, we are able to run full scale Ubuntu and ROS, to enable easy integration of the Rplidar A1M8 and cameras for computer vision. In addition, we have built a manual remote control mode for debugging and development purposes.

Special Note: During the Spring 2020 semester, the COVID-19 pandemic required significant changes to project work.

Table of Contents

Abstract……1
Executive Summary……2
Introduction……4
Alternative Solutions……6
Mechanical Hardware……6
Mobile Robot Base……6
Miniature Vacuum……7
Robot Manipulator……8
Electrical Hardware……9
Battery System……9
Motor and Drivers……10
Onboard Power……11
Charging……11
Software……11
Computer System……11
Sensors……12
Applications……12
Design and Implementation……13
Initial Designs……13
Hardware - Initial Component Selection……13
Mechanical Hardware - Mobile Robot Base……17
Electrical Hardware……20
Motor Controller Embedded Software……33
Attachment - Mini 6 DOF Robot Arm……38
Final Design……40
Mechanical Hardware - Mobile Robot Base……40
Electrical Hardware……42
Motor Controller Embedded Software……43
High Level Software - Manual Remote Control……47
Results……50
Hardware……50
Software……56
Problems……58
Hardware……58
Software……59
Future Plans……60
Conclusion and Acknowledgement……61
Citations……62
Appendix……63
Peng Peng Mechanical Report……69

Introduction:

With the rapid increase of automation, it is now not uncommon to see robots in our daily lives. From service to cleaning robots, there exist many successful commercialization examples of this technology. In industry, robots have taken over traditionally dangerous and mundane tasks, such as assembling screws and welding vehicles. With an estimated global market of over 80 billion dollars in 2019 [11], the field of robotics has become very valuable to learn in this era. However, although many engineering programs now offer robotics courses, most lack hands-on work with actual robots due to the cost and complexity of setting up such devices. On the hobbyist market, there exist either affordable, but poorly designed robot platforms with limited functionality, or costly systems that are neither space-efficient nor user-friendly. This project aims to address these issues by presenting a completely open-source, expandable mobile robot platform that fits in the home as well as a research laboratory. With a 3D printer, the robot base costs as little as $400, and is designed for easy-deployment and attachments in mind. By using stepper motors and belts for the drive system, we were able to leverage the existing 3D printing market for cheap components in both motors and the controllers. By using the Mega 2560 platform as the low-level controller, we enable more people to develop and change our base design, many of whom may have used Arduinos before but are daunted by proprietary microcontroller development environments. By using the Nvidia Jetson Nano, we further extend the accessibility goal by introducing full-scale Ubuntu in a single board computer (SBC) package, capable of running most software packages available on common desktop computers. Moreover, leveraging the built in 128-core Maxwell graphics processor unit (GPU), the Jetson Nano is able to accelerate neural network inference tasks by over 10 times compared to similar SBC solutions. Beyond these design implementations, we have also come up with custom attachments to address some specific use cases. For the robot vacuum, we have developed a computational fluid dynamics (CFD) optimized, 3D printed vacuum system powered by an electric ducted fan (EDF) commonly used in radio control aircraft. The large onboard battery is able to satisfy its power requirements, and it is analyzed in simulation. For the manipulator, we integrated a 6 DOF mini robot arm onboard. In the latest version of the prototype, we have successfully completed a working mobile robot base. Due to time and resource constraints, we were not able to make as much headway on the software front as desired, but with a solid hardware foundation, development on that front can be advanced with ease. This document is a record of the research and development process, discussing all the

hardware design, manufacturing, testing, as well as software development and modification guidelines.

System Architecture and Goals:

Before the alternative solutions are proposed, the overall systems structure of the project must be established based on goals and available resources.

Some goals of this project are:

- Create a mobile robot base that allows for multiple modular attachments, and is compact as possible to fit into small indoor spaces
- Fit a large battery onboard the robot base, to allow for high power device operation such as vacuum and robot arm to run for an extended period of time
- Devise an intelligent charging system, to allow for easy maintenance
- Have enough compute power onboard for real time navigation and image processing, as well as hardware health monitoring
- Design everything for accessibility and manufacturability in mind, and develop software stack for redistribution and modification in mind

Fundamentally, the robot consists of a mobile robot base, essentially a drivable chassis onto which the power source, computers, and additional hardware can be mounted. It should be robust and easily manufacturable. There was no quantifiable size limit at the beginning of the project, the idea was to make it as compact as possible while satisfying the power and thermal constraints.

Second, the power source. It has to store enough electrical power for the onboard computers, sensors, drivetrain, and a high power attachment. Power budget at the beginning of the project was set at 30W for onboard computers, 10W for sensors, 100W for drivetrain, and 200W for high power attachment. **Total comes out to be 340W, and the voltages will be set based on individual component choices. Operating time should be at least 20 minutes.**

Third, the computers. Although there is currently a trend in the market toward robotics, there does not exist many affordable mobile computers with the form factor more or less on the same order of magnitude as a Raspberry Pi. Pi's may be an obvious recommendation, but their proprietary operating system and lack of robotics support make it an unideal choice. Thus, the goal here is just to get as much computing power as possible within the current affordable choices, no specific floating point operations per second (FLOPS) goals are set.

Lastly, the accessibility considerations. The hardware should be made from as few proprietary parts as possible, and allow the use of 3D printing for most of the custom components. Software wise, it should be built on common open source platforms available to most users with some technical knowledge, and allow for easy modifications even for those who do not have expertise with embedded systems or high level software.

Alternative Solutions:

Mechanical Hardware

Mobile Robot Base

The most fundamental design decision that impacts the entire robot system is the choice of drivetrain of the mobile robot base. There exist many solutions, all of which have a plethora of previous examples. At the core, they can be separated by their kinematic characteristics in terms of control strategies: holonomic, or non-holonomic.

Non-holonomic drives make up most of the more simple robot platform designs. Any robot base with two drive wheels spaced horizontally apart, and turn by differentially controlling the rotational velocity of each, is a non-holonomic drive. Well known commercial examples of this include the entire Irobot lineup as well as most robot cleaners.

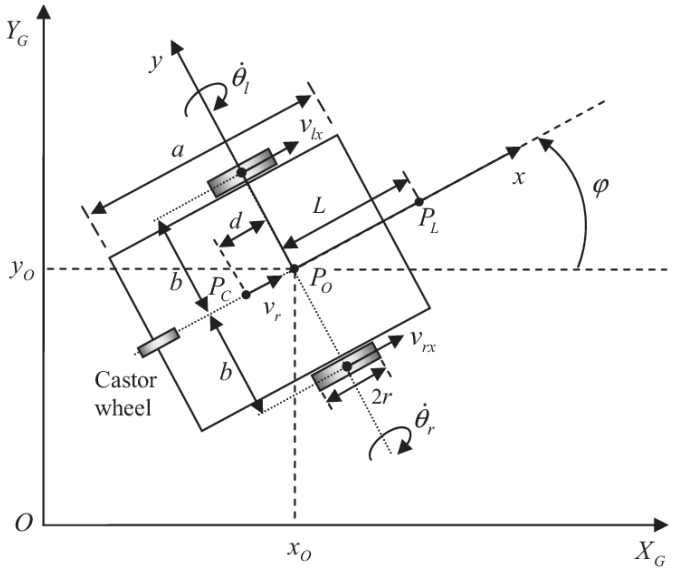


Figure 1. Non-holonomic Robot Drive [1]

In terms of dynamics, the holonomicity as it is called, depends on the degrees of freedom (DOF) of the robot in physical workspace of interest versus the degrees of freedom under control input. With a non-holonomic drive, the robot has fewer controllable degrees of

freedom than the total DOF. Assuming that the mobile base is operating in a 2D plane, it has 3 total DOF: x, y and orientation. However, It only has two controllable inputs, left and right drive wheels. Therefore, it is incapable of being fully actuated, and thus special consideration must be taken in controls design, such as feedback linearization.

With a holonomic drive, the controls do not necessarily become simpler (due to more complicated dynamics equations), but the final result is more elegant due to its nature of full actuation. Some examples include mecanum drive and omni-directional wheels, shown below.

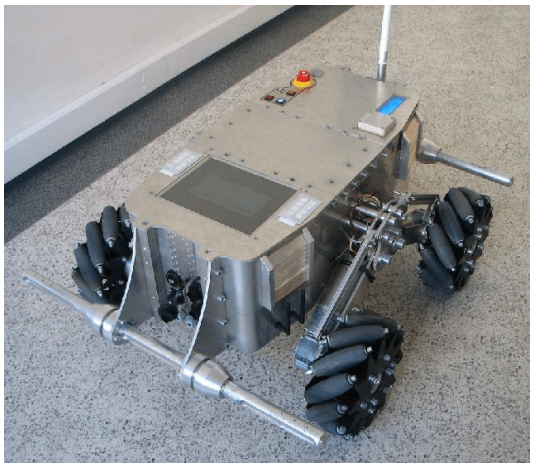


Figure 2. Mecanum Holonomic Drive [2]

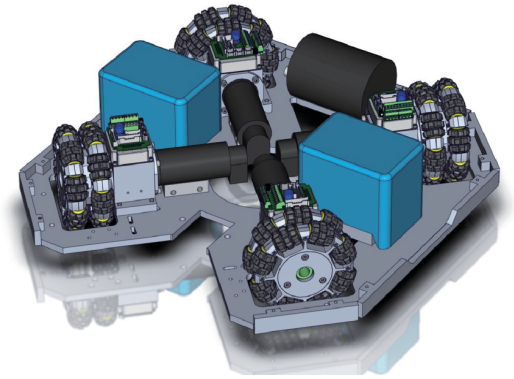


Figure 3. Omni-directional Holonomic Drive [3]

Because of the existence of opposing drive wheel rotation axes in the above designs, it allows the redirection of drive force along different directions in addition to the “tank like” differential drive of the non-holonomic design, by simply actuating pairs of wheels against each other. Therefore, the holonomic chassis are considered fully actuated, as they are able to control x y positions and orientation simultaneously and independently at all times.

Miniature Vacuum

There are many robot vacuum designs, ranging from Dyson’s advanced cyclonic system to the most basic blower fan setup. In essence, the function of a vacuum is to create a low pressure, high flow rate air stream entering an inlet near the floor, and filter out any particulates captured in the process. The most basic setup consists of a high power blower fan sucking air through a filtered inlet, with a chamber before the filter to collect any dust buildup. This is shown below in the Roomba 650 teardown. The problem with this design, as anyone who has used a cheap vacuum cleaner will know, is that the vacuum becomes less effective the longer it is used, as the filter gets clogged up with dust, reducing the airflow. Thus, the cyclonic dust

separation design popularized by Dyson proves much superior in this respect. In the diagram below of a multi-stage design, airflow is coaxed into forming strong vortices within various chambers before entering the filter. Due to the centrifugal forces generated in the “cyclones”, the dust particles tend to be flung to the outermost layer of the airflow, where it contacts the sidewalls and slows down significantly. Hence, this allows the unwanted particles to fall and accumulate, instead of taxing the filter.

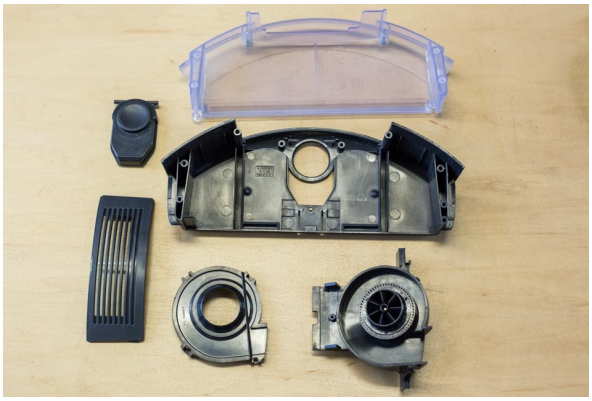


Figure 4. IRobot Roomba Vacuum Component [4]

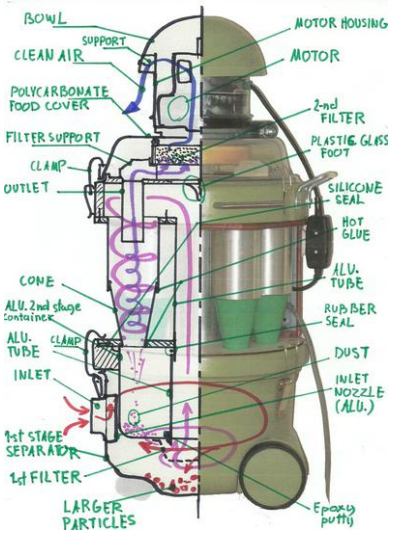


Figure 5. Custom Cyclonic Design [5]

Robot Manipulator

There are many robot arm design variations that exist, too many to list in this report. Some common design choices involve DOF, joint drive type, and feedback sensors. Depending on application, it may be desirable to have many DOF, sometimes as many as 8 or 10, even though the 3D space that it operates in only has 6 DOF (x,y,z, and 3 orientations along each axis). This is so that the robot manipulator can achieve what is called null space redundancy, that allows an arm to keep the same end effector configuration, but alter the rest of the arm positions in infinitely many ways. It allows for better obstacle avoidance and easier path planning, although often coming at the cost of rigidity, cost and controls complexity.

The joint drive design is another field in itself. A leading design is the harmonic drive, which uses a series of deforming strain wave gears to achieve incredible tolerance with no backlash. More cost efficient solutions use gear reduction trains and worm gear systems. This topic also combines with sensor selection, as angular position has to be recorded and broadcasted from each joint. Leading solutions from companies such as AMS use hall effect

and an on-axis polarized magnet to sense angles to a resolution of 14 bits and above. Similarly, cost effective and more common solutions exist in the form of potentiometers and optical encoders.

Electrical Hardware

Battery System

Besides the drive system, this is one of the other major hardware design components of this project. There are many battery chemistries and designs on the market, some of the most common include lead-acid, nickel based chemistries, and lithium. Lead-acids have a long history and have been used in automotive applications for its low costs and reliability, but its low power density makes it an unsuitable candidate for mobile robot applications. Nickel based chemistries include Ni-Mh and Ni-Cd, which have been the staple for remote controlled cars and planes before the advent of the lithium cell. They are capable of delivering bursts of power at a capacity-to-weight density between lead-acid and lithium. Lastly, lithium chemistry batteries include lithium polymer, commonly abbreviated LiPo and used in modern radio controlled systems, and a variety of other chemistries used in appliances, mobile devices and even vehicles. They are the best in every category in terms of power, density and size, but lose out in reliability due to lithium's highly reactive nature. Below shows a chart that compares density of most available battery chemistries.

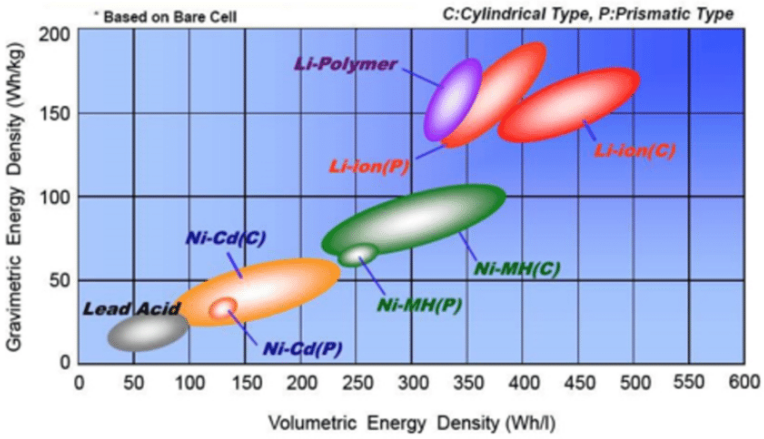


Figure 6. Battery Density Plot [6]

Form factor is another consideration in battery design. Lithium cells come in two commercially available form factors (excluding custom designs): cylindrical and pouch. Cylindrical cells typically come with a metal case having the polarities at each end, much like a

AA battery but larger. Pouch packs are what they sound like, a combination of rectangular pouches packaged together usually by heat shrink, and wires leading out for balance connectors and the polarities. Cylindrical cells typically offer better reliability and protection as they are enclosed in a hard casing, but can be less power dense and more difficult to package due to their geometry. Pouch packs are prone to swelling and physical damage, but are cheap and widely available due to the proliferation of remote controlled hobby systems.

Lastly, lithium batteries containing multiple cells in series require a balance system for charging. This is commonly called the battery management system (BMS). They help monitor the voltage of each series cell and equilibrate them if any differences are detected. This is important because the discharge characteristics of the cells vary with different states of charge, such as internal resistance. If this is not balanced across the whole system, it could result in over-discharge and damage in the long term. Most BMS systems simply bleed off excess energy through resistors, although some advanced ones boost the voltage up and use it to charge other cells. In addition, many commercially available solutions for lower power applications include built-in resettable fusing, usually with a mosfet.

Motor and Drivers

This is another subject that can have an entire book written on the various choices. The ones under considerations here are brushed DC, brushless DC, and stepper motors. Brushed direct current motors are the most common and cheapest solutions, using pairs of brushes on the commutator to mechanically accomplish switching with a constant power input. However, their efficiency suffers due to this design. Brushless DC motors (BLDC) take this design and place switching on the controller, allowing for much higher efficiency. However, the controller is expensive and complex. Both of these motor architectures are typically used for high rotational speed applications, with lower torque requirements. Thus, to use in a drivetrain they need a reduction system to increase the torque. Lastly, stepper motors are less frequently used in power transmission applications, and more in precise motion controls. However, they have become very cheap due to the proliferation of 3D printers, as have their driver systems. They do not excel in high speed ranges, but provide high torque on the low end. Their efficiency varies greatly at different speed ranges, depending on the specific winding and other specifications, and can be as good as BLDC motors in the best operating conditions, and worse than brushed

motors in the worst conditions. However, stepper motors typically generate less heat as they are lower powered for the same size, which can be beneficial for cooling.

Onboard Power

Because the battery system voltage varies over its state of charge, there must be an onboard power supply system for every sub-system that needs different stable voltages. As most demand a relatively significant current, the only choices are switching buck and boost converters. For any microprocessor level power requirements, linear converters will be able to do the job.

Charging

For mobile robot applications, most existing solutions use physical metal contacts for automatic dock charging, such as the Irobot line of products. Others use removable battery packs so that they can only be charged off the robot. Another solution is to use wireless charging, which loses energy in the process, but does not require physical contact, and allows for some misalignment. This has only become a new development in the mobile phone industry in the last few years, so there are few existing high power solutions for robotics applications.

Software

Computer System

As mentioned before, there does not exist many commercially available embedded computers capable of carrying out vision and navigation, at a price point that is acceptable for this project. Some existing solutions are Raspberry Pi, Nvidia Jetson Nano, and various lower-end field programmable gate arrays (FPGA) system on chip (SoC) solutions such as the Altera DE10 Nano and various offerings from Xilinx. The Raspberry Pi is cheap, but lacks serious parallel compute power necessary for image processing. The Jetson Nano is more expensive, is still a general purpose single board computer, but has a proprietary hardware accelerator in the form of the 128 core GPU on the same die as the CPU. The FPGA solutions, although can most likely beat the Jetson Nano in specific computation speed benchmarks, are very application specific and do not allow for rapid prototyping.

All of the systems run Linux. Raspberry Pi runs Raspbian which is a more proprietary system designed specifically for the hardware, which is based on ARM architectures. This

means it has less support for niche applications, some of which may be necessary for robotics. Jetson Nano is also based on the ARM architecture, but Nvidia has helpfully ported a full featured Ubuntu operating system. This gives it the best support for open source applications that can be leveraged to our applications. For the FPGAs, due to their custom hardware, they usually support more primitive Linux distributions, usually without a graphical user interface. This also makes their support rather lacking, besides the most common applications.

On the low level, for electrical hardware functionalities such as pulse width modulation (PWM), input/output (IO), and analog, many of the boards mentioned above are in fact capable of supporting them. However, as most contain efficient processors, the logical voltages are 3.3V instead of 5V. The other solution is to use a separate microprocessor to handle all low level controls and communications, and relay the information with a secondary link to the master computer. This also has the benefit of placing the risk away from costly components, in case electrical faults do occur.

In terms of networking and communications, some protocols that are helpful include wired and wireless local area networks, bluetooth and serial. These are mostly supported by the systems described above, as they are fairly modern designs.

Sensors

There are many types of sensors available for robotics navigation. The range that we would like to investigate in this project include Lidar and cameras. Although Lidar is traditionally a costly hardware solution, it gives far more valuable information without needing physical contact compared to bump sensors or ultrasonic distance sensors. Moreover, recent interest in robotics has made 2D lidars very affordable. Using cameras as environment sensors requires significantly more computing power as mentioned before, as the data is not natively a geometric description of the world. However, they are much cheaper in terms of hardware, and lead to many interesting software solutions, utilizing neural networks for prediction and segmentation.

Applications

For high level software, besides writing our own programs, we hope to leverage open source projects such as robot operating system (ROS). There do not exist other comparable systems on the market, as it is a fairly complex and complete ecosystem with different distributions, simulators and other tools.

On the embedded software side, there exist many solutions on the market, depending on the hardware chosen. However, a free, accessible and common development ecosystem is Arduino, and there even exists a version that can run in Ubuntu. Although there exist many quirks and most programs do not use advanced functions, Arduino is in fact capable of accessing registers and control hardware functions directly, for the AVR architecture specifically.

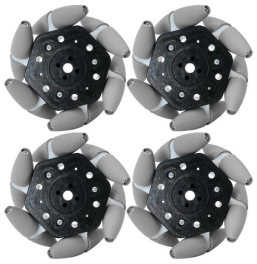
Design and Implementation:

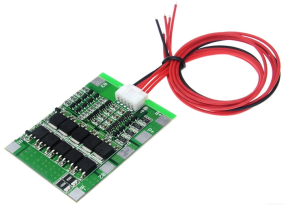
Initial Designs:

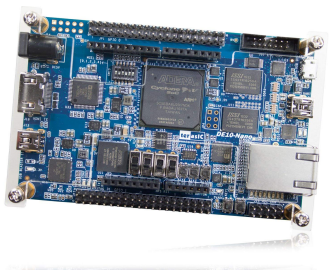
Hardware - Initial Component Selection:

Since it was not feasible to design and build every component from scratch, we selected a handful of available off-the-shelf parts that we can base our design upon. These are summarized in the table below, and the component choices also dictated some of our decisions from the list of alternative solutions above.

Table 1. Component Selection

Name	Image	Affected Design Choice
AndyMark Mecanum Wheels		Drivetrain Control Type: Holonomic
NEMA17 Stepper Motors with Built-in Encoders		Drivetrain Motor: Stepper

GT2 Timing Belt		Drivetrain Transmission Type: Belt Reduction Drive
AndyMark High Grip Wheels		Drivetrain Control Type: Non-Holonomic
Tesla 18650 Lithium-Cobalt-Aluminum Cells		Battery System Type: Lithium
VRUZEND Battery Kit		Battery System Design: Removable
4s Battery Monitoring System and Fuse		Battery System Monitor; Battery System Architecture
Nvidia Jetson Nano		Robot Main Computer Selection

Intel Altera DE10 Nano		Robot Secondary Computer Selection
RPLidar A1M8		Sensor Selection

* Images From Vendor Websites

To justify some of these design/component decisions, we can begin from the wheels. Mecanum wheels from AndyMark (a large online vendor of robotics parts, mostly for competitions such as FRC and Vex) were the first choice, due to the innate benefits of holonomic motion. They are somewhat expensive and contain many moving parts (due to having 12 rollers per wheel), hence there are some risks in choosing this component. So that is the reason for choosing to purchase the normal, non-holonomic high grip wheels as well, which are the same size and can be mounted on the same adapter system. As much of this project aims to explore previously untested robot designs, it was acceptable to get two interchangeable parts that can be tested for effectiveness.

Next are the motors and the belt drivetrain. Much of the reason for choosing stepper motors is subjective preference and novelty, as it is not a commonly used component in powering drivetrains. Therefore it would make this project unique in exploring that design choice. It does however lower the overall cost and complexity of design due to its torque and control characteristics. However, just to be safe, we opted to include a reduction system for increasing torque and lowering velocity. GT2 belts and sprockets were chosen due to its wide availability from 3D printer applications (which is also the same reason for cheap stepper motors), and also due to their low noise and low maintenance characteristics. The other option is to use gears, which are far louder and may require lubrication/cleaning if operating in a dirty

environment. The encoders were chosen because the vendor happened to give that option for the steppers, which can be useful for odometry in localization.

Moving to the battery, lithium was the obvious chemistry of choice for its superior density, but the form factor was a subjective one. Since one of our group members had access to cells from a salvaged Tesla Model S battery, it was an obvious choice to use the 18650 form factor cells. Part of this was also for the cool factor, to be able to say that the robot is powered by Tesla batteries. But fundamentally, there are many benefits to using the Tesla lithium-cobalt-aluminum chemistry. It had a capacity of roughly 10 Wh, and a continuous/max discharge of 5/20A respectively, which is on the top end of 18650 performance. At 3V discharged and 4.2V fully charged, it made sense to go with the configuration of 4 cells in series, arriving at 12V discharged and 16.8V fully charged. This voltage range works well because it is at the upper end of remote controlled hobby equipment such as electric ducted fans, allowing them to work more efficiently with less current. Moreover, this surpasses all other voltage requirements in the system, allowing simple buck converters to handle the internal power supply instead of using buck-boost. Therefore, we also selected a 4s battery management system to go with it. It monitors the cell voltages relative to each other, and keeps them equal (or balanced, as it is called) to avoid differences in discharge characteristics. Also, it runs the entire current load through the board, and fuses the connection if it is above 30A, which works with our power requirement of 340W. Lastly, it also performs low voltage cutoff at 2.7V to prevent over discharge, ensuring the safety of the battery system.

To make construction of the battery pack easier, we opted to choose VRUZEND's solution that allows for disassembly. Compared to spot-welded tabs, it was both safer and easier to maintain. Packaging wise, it did not add too much excess bulk, just some vertical height for the power connections. However, there were issues with this product, as will be explained later.

Lastly, we settled on the Nvidia Jetson Nano for the primary computer, and the Intel Altera DE10 Nano for the secondary computer. The Jetson was chosen as the primary for its good software support and general purpose hardware, while the DE10 Nano was chosen for its low power but capable compute performance. The choices were quite limited given our budget, so there were not many other options. The computer choices were important at this point for determining the packaging requirements, as they occupy a large footprint and have cooling needs. Also the RPLidar A1m8 was selected, as it was the most affordable choice with open source software packages available.

For more detail on the chosen hardware, a detailed bill of material (BOM) is given at the end of this report with links to vendors and manuals.

Mechanical Hardware - Mobile Robot Base:

The first system that needed to be designed was the robot base. As discussed before, mechanically its packaging would define constraints for everything else in the system. And the most difficult component of that is the drivetrain system, as it required careful placement of moving parts. Therefore, it was the first assembly put together:

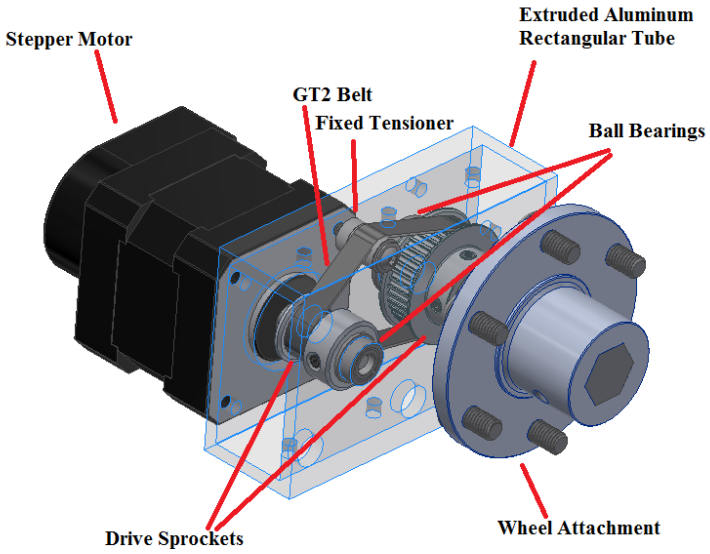


Figure 7. Drive Unit

The idea was to have 4 identical units of these drivetrain units, mounted at opposite corners of the robot. This way, it allowed the use of repeated manufacturing of parts, which reduced the time needed. There are 3 custom machined metal components in here, which are the wheel axles, the fixed tensioner, and the extruded aluminum frame.

Next were the batteries. As mentioned before, we chose to use the Tesla 18650 cells, and the modular kit for putting them together. Also, the configuration was decided to be 4s, but the parallel configuration was not decided yet. Given that 340W needed to be supplied at maximum, at minimum 4 cells in parallel would be needed for stable, long term operation. This was arrived at by:

340W / 17V = 20A (Fully Charged)

And this is the restricting requirement because the cells are current limited. With 4 cells, it would equal 5A continuous discharge per cell, which is about the recommended operating limit. Now in

terms of mechanical packaging, using the drive units designed above, there is actually enough space to fit in 5 cells in parallel, although the 4 series pack would have to be split in half and bridged by a wire in the middle. Therefore, the final design was 4s5p, see diagrams below.

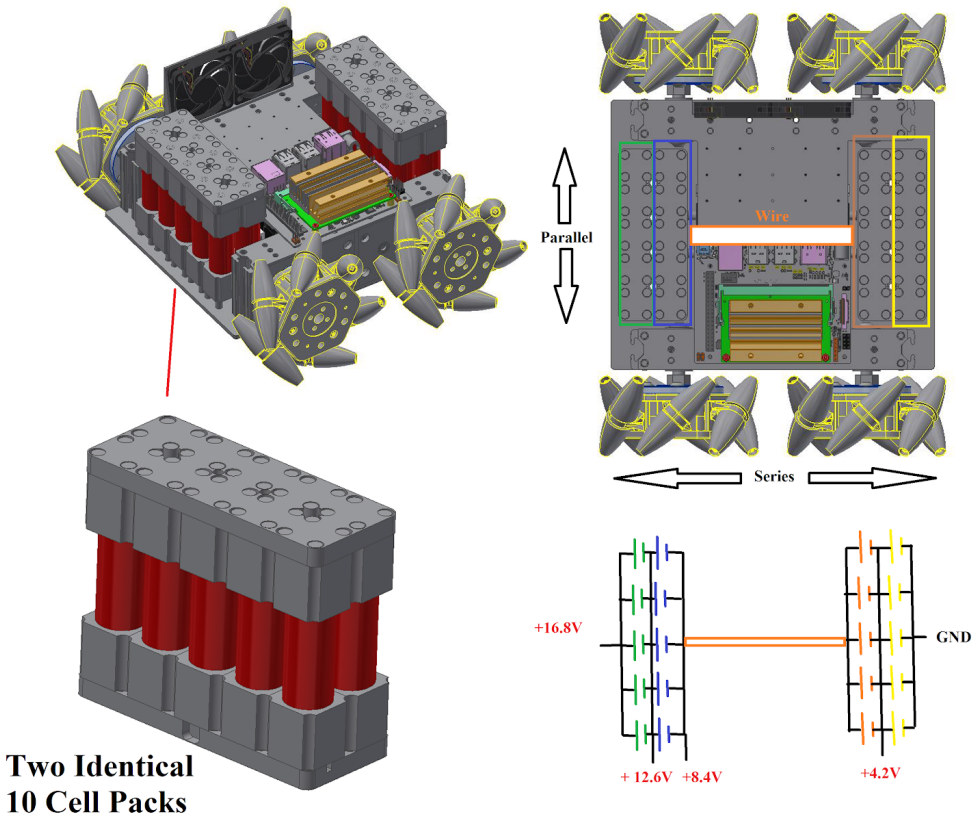


Figure 8. Battery Layout

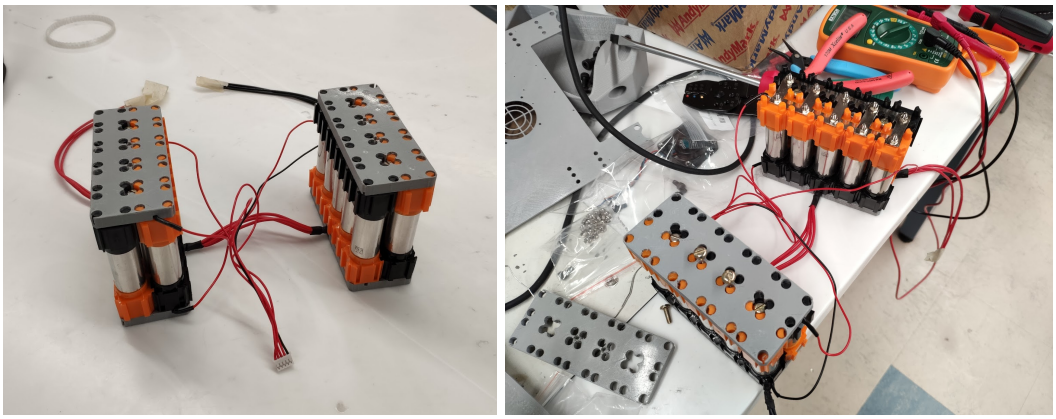


Figure 9. Battery Construction

With the major components in place, it was ready for a bottom plate to mount everything together. For manufacturability, it was designed to be a flat piece of steel that can be waterjet cut easily.

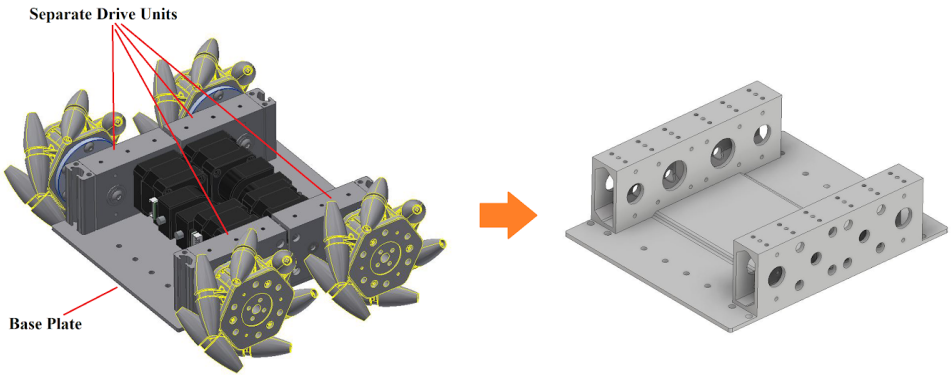


Figure 10. Combined Chassis Changes

However, after examining the assembly and reviewing its manufacturability, the metal bottom plate and the drive unit frames could simply be combined into a single mega chassis, and 3D printed easily (although it would be a long print). This change in design exhibits some of the original goals of simplifying manufacturing, moving towards 3D printing, and increasing accessibility.

Lastly, the onboard computers were fitted along with the lidar and side cooling fans on a top mounting plate to increase rigidity. The overall assembly looked like this.

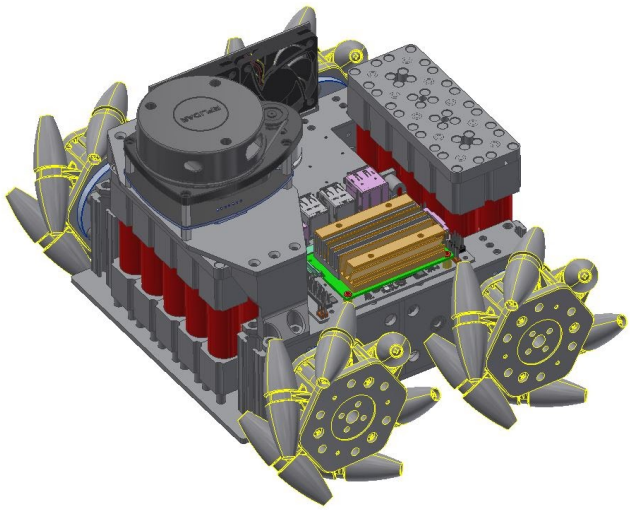


Figure 11. Preliminary Design Assembly

And after some time spent in the machine shop on the lathe to make the two metal parts, as well as many 3D printer hours later:

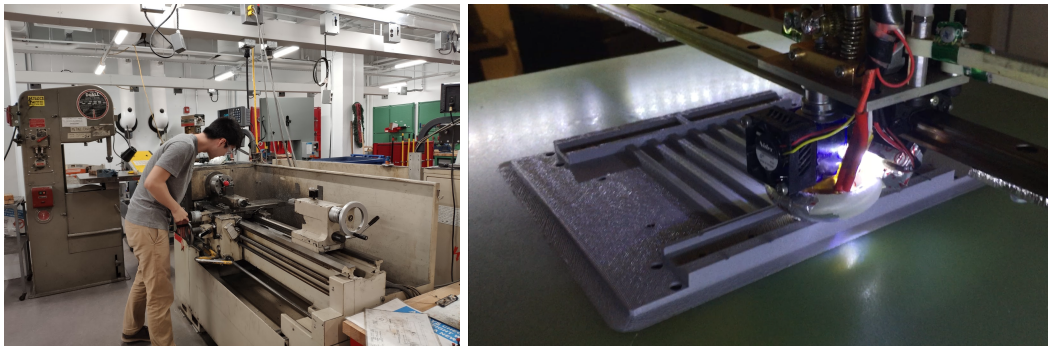


Figure 12. Manufacturing Progress

The following show the first revision of the design assembled, along with the mini 6 DOF arm:

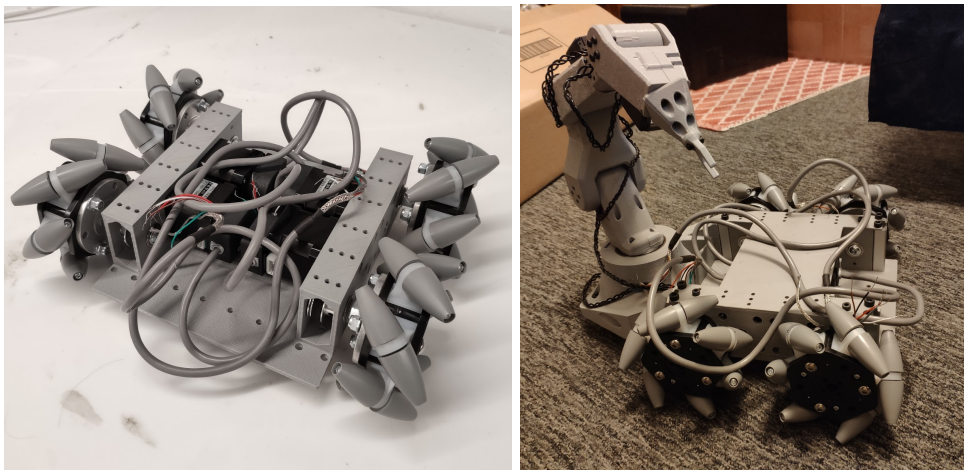


Figure 13. First Prototype Assembled

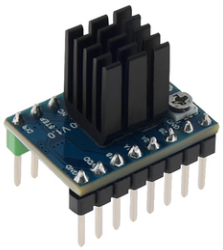
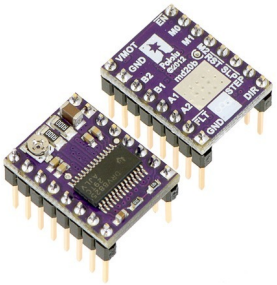
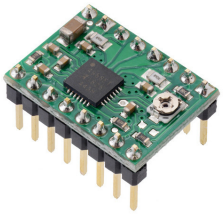
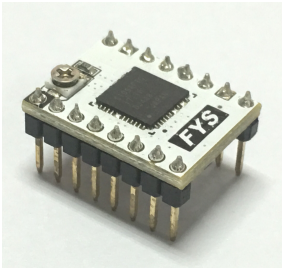
Note that the robot arm is from a separate personal project, and more details can be found here: https://github.com/SkookumAsFrig/Mini_6DOF_Arm. The mechanical design will not be explained in detail here, as that would make the report too lengthy. The important facts are that it has 6 DOF and uses serial servos for all its joints. A simple adapter was created to slot easily into the 8020 style extrusion at both ends of the robot, onto which the arm is mounted.

Electrical Hardware - Motor Controller, Power Supply and Hardware Health Monitor:

The final goal was to combine the motor controller, power supplies and various hardware health monitoring sensors onto a single printed circuit board (PCB), with everything controlled by an intermediary microcontroller that relayed the information to the high level computers. But starting from scratch, we built and tested each component separately.

Motor Controller:

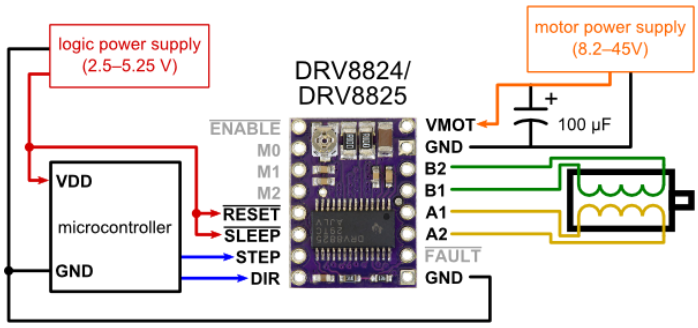
One of the benefits of choosing stepper motors is that the motor controllers are very much standardized and modularized, and there are a variety of options from different integrated circuit chips. The table below shows the list of stepper drivers that have been tested.

Manufacturer/Name	Image	Key Metrics
Trinamic TMC2130		4.75-46VDC, 2.0A max coil current (2.5A peak), 256 microsteps max resolution, *stealChop*, *stallGuard2*
Texas Instruments DRV8825		8.2-45VDC, 2.2A max coil current, 32 microsteps max resolution
Allegro A4988		8-35VDC, 2.0A max coil current, 16 microsteps max resolution
Toshiba TB67S109		10-47VDC, 4.0A max coil current, 32 microsteps max resolution

* Images From Vendor Websites

This list of stepper drivers are all modularized packages with DIP header pins for 3D printer applications, hence their standardization of input/output (IO) and dimensions. This makes them very easily interchangeable, in order to test them for performance. The important metrics to note are the microstep divisions, as well as the max coil current. The microstep division dictates how fine the driver can control the angular position of a motor. While the positional resolution is not of supreme importance to us, This characteristic greatly impacts the driving torque and control properties as will be discussed later. The current in this case also relates directly to the motor torque, as well as the heat needed to be dissipated through the driver (which is why some images above have heatsinks). Being such a small size, the driver thermal performance can be a large factor in its stability and longevity. Lastly, the Trinamic brand TMC2130 stepper driver has advanced features that the others do not, such as the *stealChop* and *stallGuard2* modes. The first one allows super fine microstepping and thus very quiet operation, and the second one allows the driver to detect stalling just by measuring back electromotive-force. Finally, the voltage ranges of these are all relatively similar, and the choice of the 12-16.8VDC 4s5p battery pack will work fine with any of them.

The control of all of them are all relatively similar, at least operating at the most basic modes. Take the diagram of DRV8825 as an example.



MODE0	MODE1	MODE2	Microstep Resolution
Low	Low	Low	Full step
High	Low	Low	Half step
Low	High	Low	1/4 step
High	High	Low	1/8 step
Low	Low	High	1/16 step
High	Low	High	1/32 step
Low	High	High	1/32 step
High	High	High	1/32 step

Figure 14. Stepper Pinout, Connections, and Microstep Settings [12]

After connecting the requisite power and ground pins, as well as the 4 phases of the stepper motor, the rest of the pins are for configuration and control. M0, M1 and M2 define the microstep divisions, and are typically configured in a binary format like the one listed for DRV8825 above. The Reset and Sleep pins are normally held to a set state to let the Enable pin control the state of the motor controller, which toggles its power on/off. One of the specialties of the stepper motors is that even if it is not in motion, it is capable of holding its position with a large torque, which the enable pin controls. Lastly, the Direction pin is a binary forward/backwards signal, while the Step pin takes a squarewave signal. On each pulse, the

driver will advance the motor one microstep. A step of the motor is normally defined by the motor manufacturer, and the motors typically come in 200 or 400 steps/revolution varieties. This corresponds to 1.8 or 0.9 degree per step, and multiplying that by the microstep division gives the corresponding control resolution.

Since the modularized “stepsticks” are fairly user friendly with many online references, it was not too difficult to get it setup and tested on a breadboard with a microcontroller. The software component is much more complicated, and will be discussed later.

Onboard Power Supply:

There are three power supply switches. Since the robot uses battery as power supply, input voltage varies from 10-17V DC (expanded from 12-16.8V to give some factor of safety). Two outputs are 5V 6A and 12V 2A, which are for the Jetson Nano, DE10 Nano and another possible 12V device. Another output is 17V with input being 5V 2A, for charging the battery. All of them are designed in Altium. In the circuit design, apart from electrical performance requirements, other important issues to consider are minimizing PCB area and capacitor cost.

We chose to use Texas Instruments' free online WEBENCH Power Designer tool [7], as we do not have extensive experience in designing power electronics of this type. From the specs above, we decided that having a buck-boost setup was too complicated, and we do not anticipate draining the battery past 12V (3V per cell) anyways. Thus, we need one 5V 6A buck, one 17V boost, and one 12V 2A buck converter. Using the online design tool and inputting our requirements, we chose the following designs and ICs, with the output being the following reference diagrams:

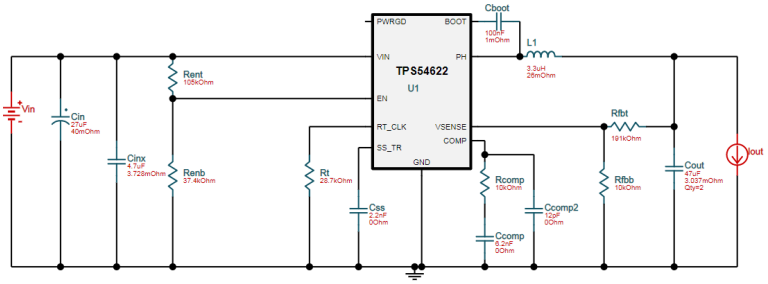


Figure 15. 5V Buck Converter Design [7]

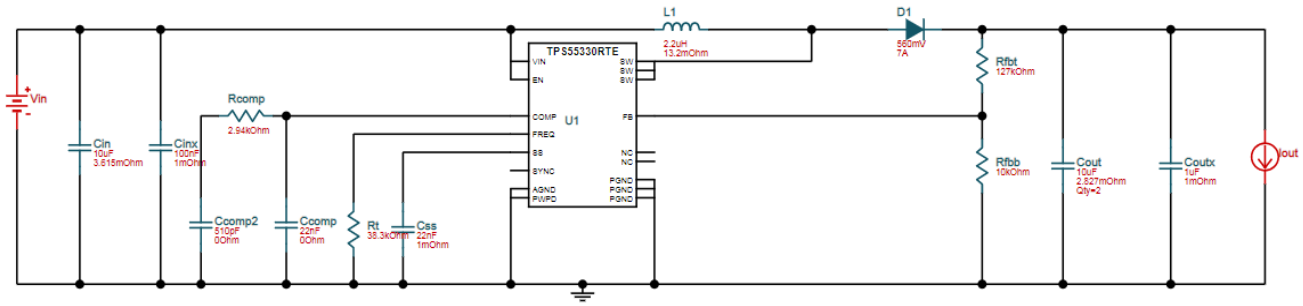


Figure 16. 17V Boost Converter Design [7]

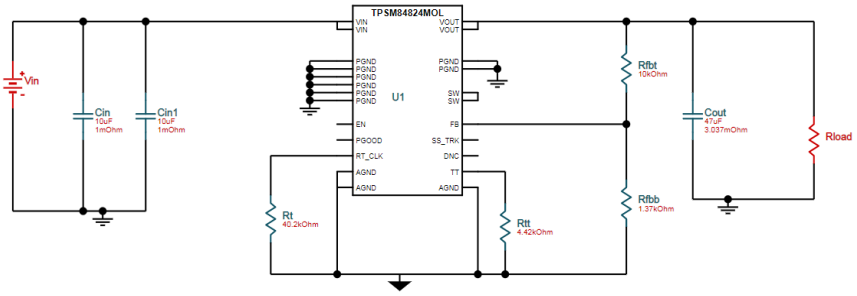


Figure 17. 12V Buck Converter Design [7]

We use the reference designs above only as a starting point. Below shows the hand calculations to validate and verify the web design, for the first 5V 6A buck converter.

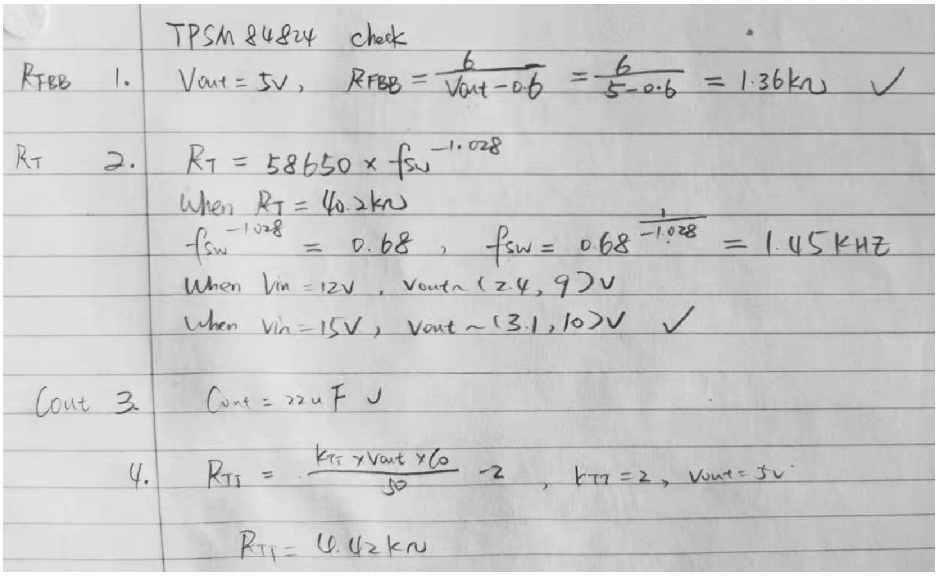


Figure 18. 5V Buck Converter Hand Calculations

With the values confirmed, we laid-out the schematics in Altium Designer using available parts from digikey, and the result is shown below.

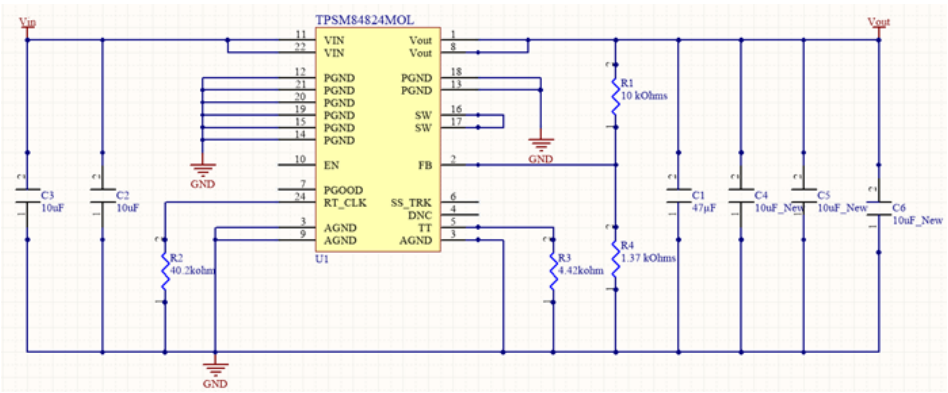


Figure 19. 5V Buck Converter Schematic

Similarly, below are the hand calculations used to verify the second 5V input, 17V output boost converter.

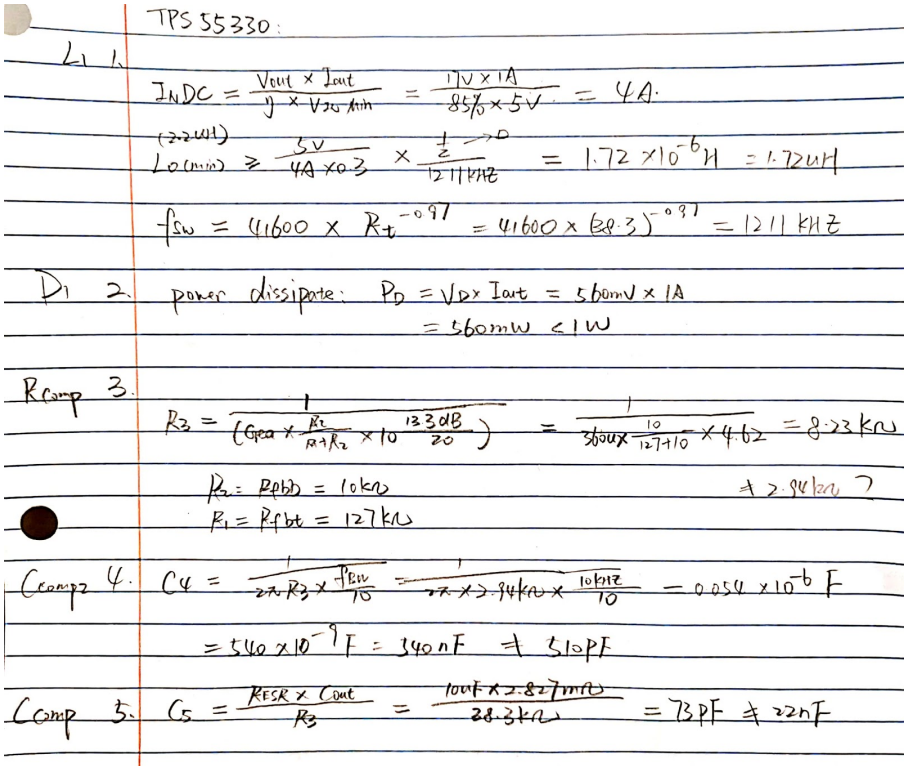


Figure 20. 17V Boost Converter Hand Calculations

And the altium schematic is shown below with actual available parts.

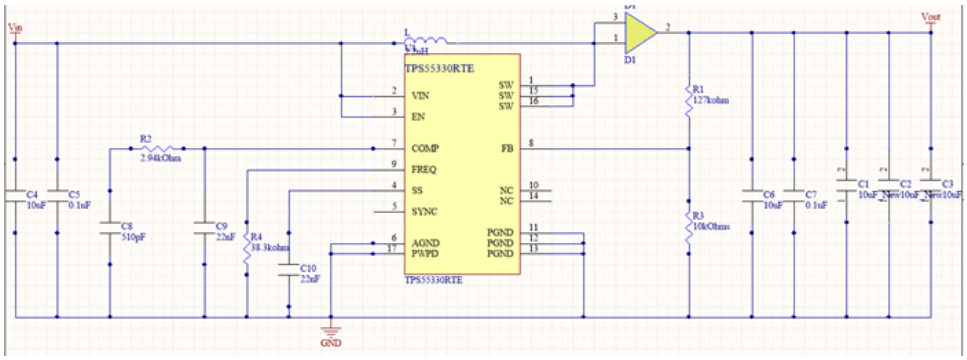


Figure 21. 17 V Boost Converter Schematic

Lastly, below are the hand calculations for verifying the 12V buck converter.

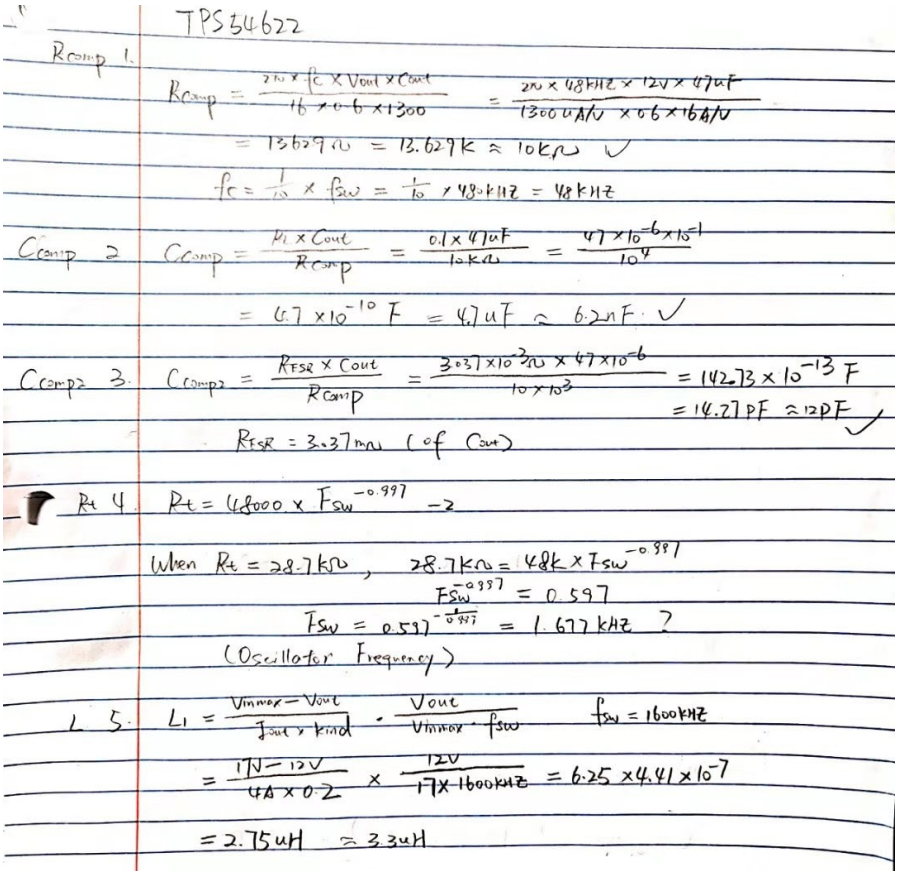


Figure 22. 12V Buck Converter Hand Calculations

And below is the Altium schematic with available parts.

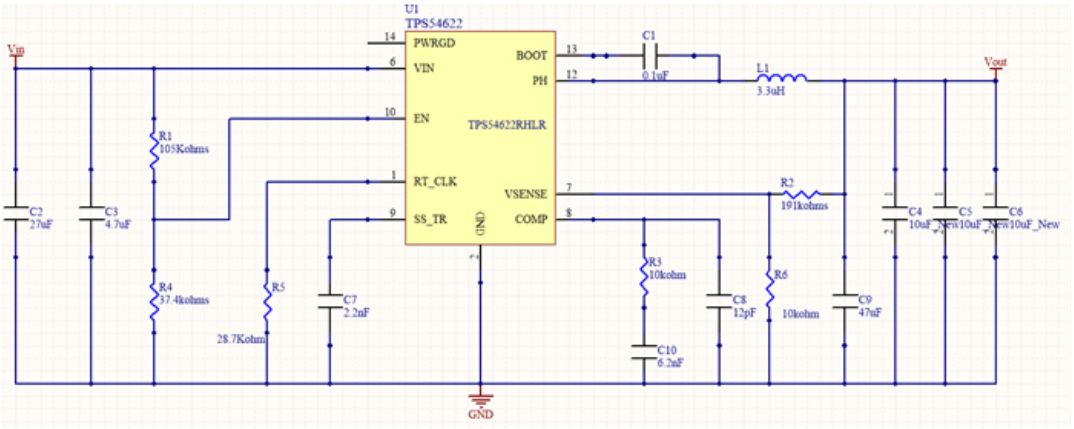


Figure 23. 12V Buck Converter Schematics

With the preliminary designs complete, an attempt to layout the boards with a single layer PCB was done, so that it can be cut on the Maker Lab PCB mill, and tested with assembled parts. Below show some screenshots.

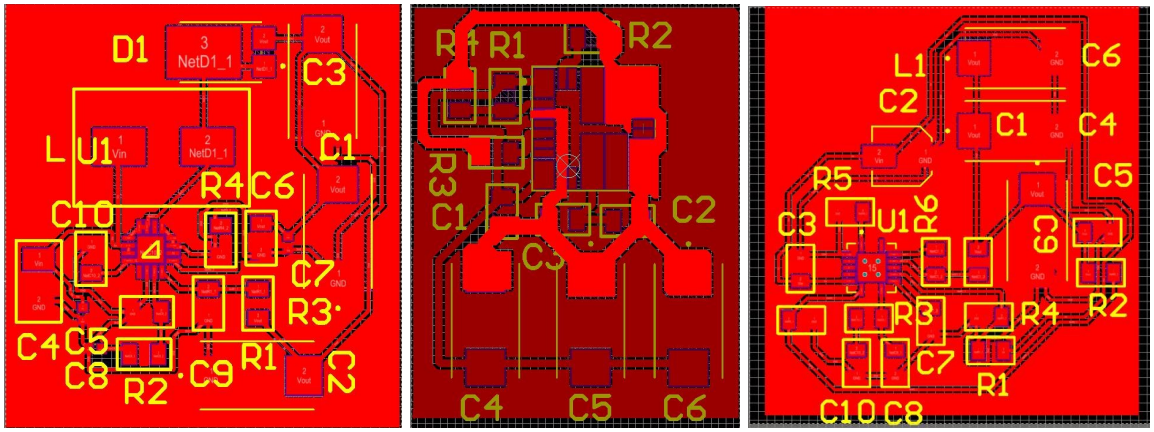


Figure 24. Preliminary Altium Designer Layouts

However, despite the work shown above, the confidence of these DC-DC converters working is not very high among the team, for several reasons. First, the architectures require many discrete components for the task compared to other integrated switching solutions, and even though most are passive components, an incorrect calculation could throw the whole design off. Second, the numerous components require a rather large footprint, which is not great for a compact robot.

Therefore, after gaining some experience with the designs above, we decided to go with a tried and true solution. From previous projects, one of the team members accrued a small number of compact buck and boost converters, which can be had cheaply from Amazon for less than 2 dollars each. They have fewer components than the designs shown above, and reach

fairly decent efficiency without overheating. After some research, we settled on the two modular designs for buck and boost converters, using the XLSemi XL4015 and XL6019 integrated switching solutions. Below show the simple and elegant reference designs in their respective datasheets.

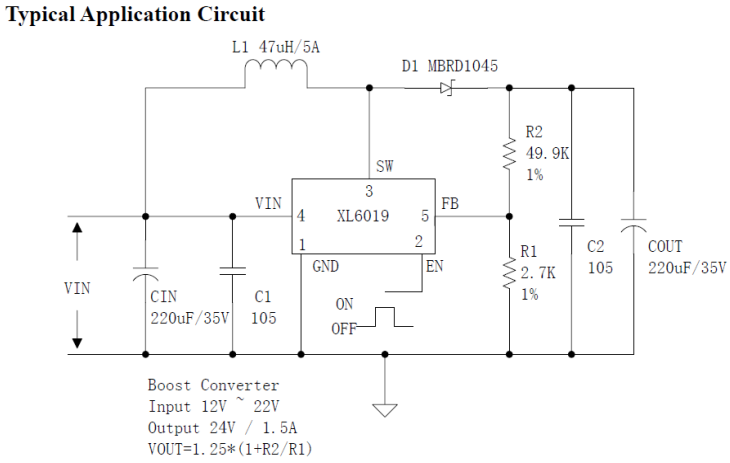


Figure4. XL6019 Typical Application Circuit (Boost Converter)

Figure 25. XL6019 Boost Converter Reference Design [8]

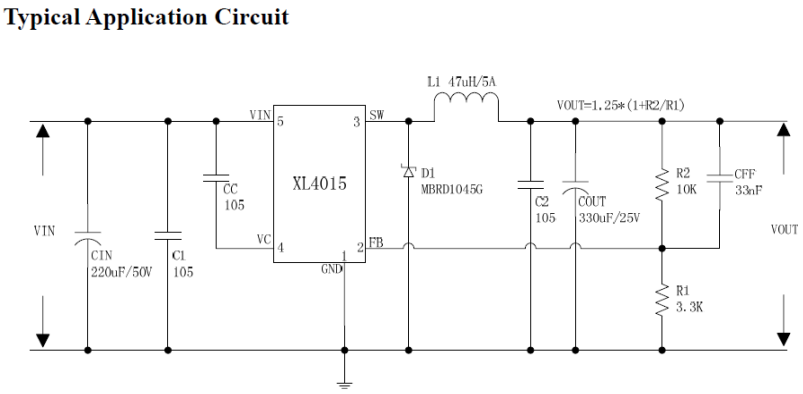


Figure4. XL4015 Typical Application Circuit (VIN=8V~36V, VOUT=5V/5A)

Figure 26. XL4015 Buck Converter Reference Design [9]

The equation for setting output voltages with resistors are given for XL4015, the buck converter, by:

$$Vout = 1.25*(1+R2/R1)$$

And similarly, for XL6019, the boost converter, by:

$$Vout = 1.25*(1+R2/R1)$$

Both are identical, which makes it easier for component selection. Since resistors are typically not made in every fractional division, we used excel to play around with the R1, R2 values, and settled on the following:

Table 2. DC/DC Converter Voltage Setting Resistors

Voltage/Type	5V Buck	17V Boost	12V Buck
R1	3.3kΩ	1.5kΩ	1.6kΩ
R2	10kΩ	13kΩ	20kΩ

And for the filtering capacitors, since they do not impact the control loop directly, will comprise of as many low equivalent series resistance (ESR) ceramic capacitors as possible for high frequency filtering, as well as a few large aluminum polymer capacitors for low frequency filtering. For the inductor choice, we decided to go with the recommended 47uH, as there were no reference materials for tuning that in the datasheet.

First Prototype:

As we did not have access to the XLSemi switch components in the US (at least not on digikey), we decided to build a prototype of the stepper motor driver combined with discrete buck converter PCBs onto a single module, so that we can work on the software at the same time. We decided to go with the Arduino MEGA 2560 based on the Atmel atmega 2560 microcontroller, for its large IO count as well as accessibility. Thus a single layer design was developed as shown below, cut on the PCB mill, and put together.

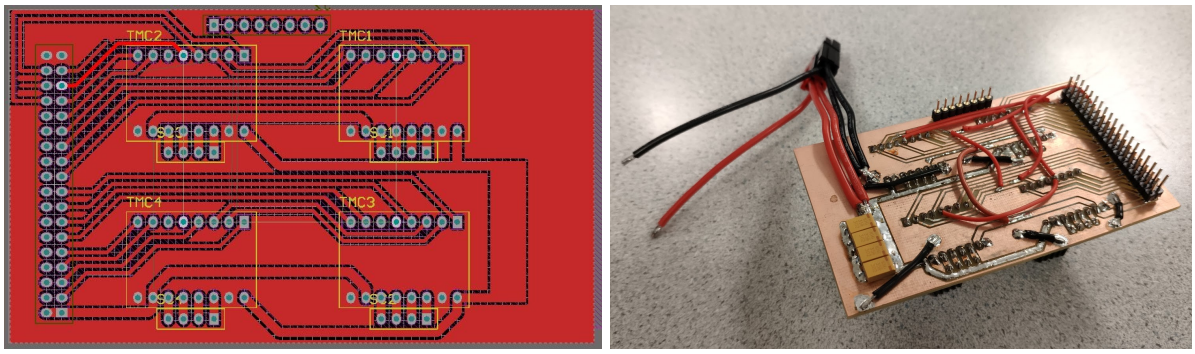


Figure 27. PCB Layout for First Prototype and Soldered Board

Note that the single layer design was not able to connect everything as needed due to the limited space, so some wires were needed to help bridge connections. Then the bottom side was epoxied for protection, and the DC-DC converters were hot-glued on top with a 3D printed bracket, shown below. Also note that the step pin was incorrectly routed in the schematics, so there are the yellow wires to fix that.

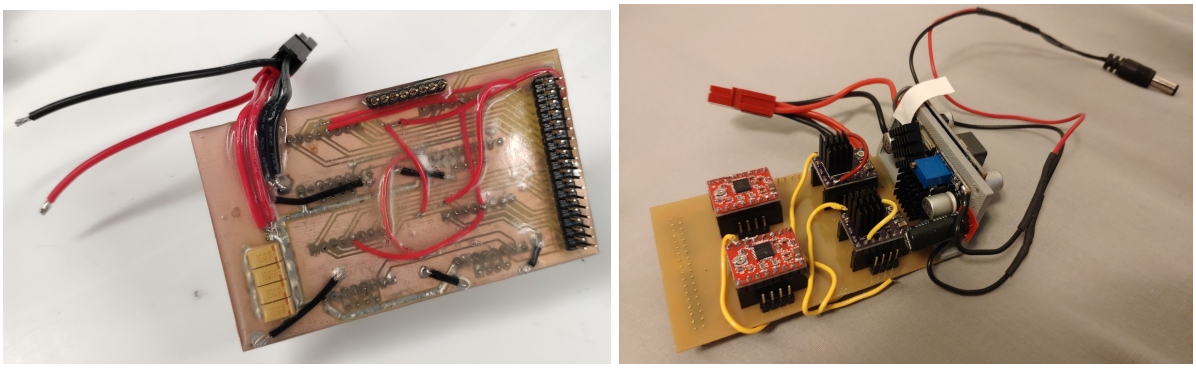


Figure 28. Epoxied Bottom Side and Fully Assembled Prototype Board

Power Monitoring:

The above prototype neglects sensing feedback, as its primary design goals were just to get something that functions. However, for a system to reliably work in the field, we believe it is necessary to provide information on its health with data such as voltages and currents. If we consider the prototype above as revision 1, then in revision 2 we added power monitoring in the form of voltage sensing of each cell in the battery pack and current output of each channel, including for charging.

Voltage monitoring is done by the microcontroller's onboard analog to digital converter (ADC), as the 10 bit resolution offered was more than enough. Accuracy is not a primary concern here, and as long as we have some feedback of the voltage levels it was considered good. However, Since there will be 3 noisey switching power supplies in addition to 4 noisey motor drivers onboard, it was a good idea to add some level of filtering in the form of RC filters. Lastly, to allow the 5VDC range of the microcontroller to read the entire battery range, it was necessary to run the higher voltage levels through a voltage divider.

Current monitoring is more difficult, as there are many ways to go about it. Voltage data did not need to capture transient behavior, so its requirements were more lax. However, it would be interesting to see more instantaneous power draw inferred from the current monitors. So it would benefit from a more robust solution. One way to do this is to run the loads by a large shunt resistor, then measuring the voltage drop across with the ADC. But this requires some power dissipation which is undesirable, especially at the relatively high currents that the board will be seeing. The solution that we settled on was Allegro's hall effect current sensors, which have a generous range, minimal inline resistance, and clean analog outputs (that do not need additional circuitry to read). They were put on the outputs of 5V and 12V bucks, as well as the

charging input for the boost converter. Additionally, ADC ports were led out to an JST surface mount connector to allow for an external, high current sensor to measure overall system power.

The appendix shows the full schematic of the board, which was designed for professional manufacturing in 4 layers to allow for the best thermal performance as well as signal integrity (as this is a mixed power/signal board). There are additional light emitting diodes (LEDs) to indicate power status. Moreover, the board is designed to accommodate wired connectors, as the robot is very compact and any power connections will need to be led out to a less crowded area. For signal integrity shielding as well as to reduce power transmission losses, as much of the board was poured as copper planes as possible. Below shows the layout of different layers.

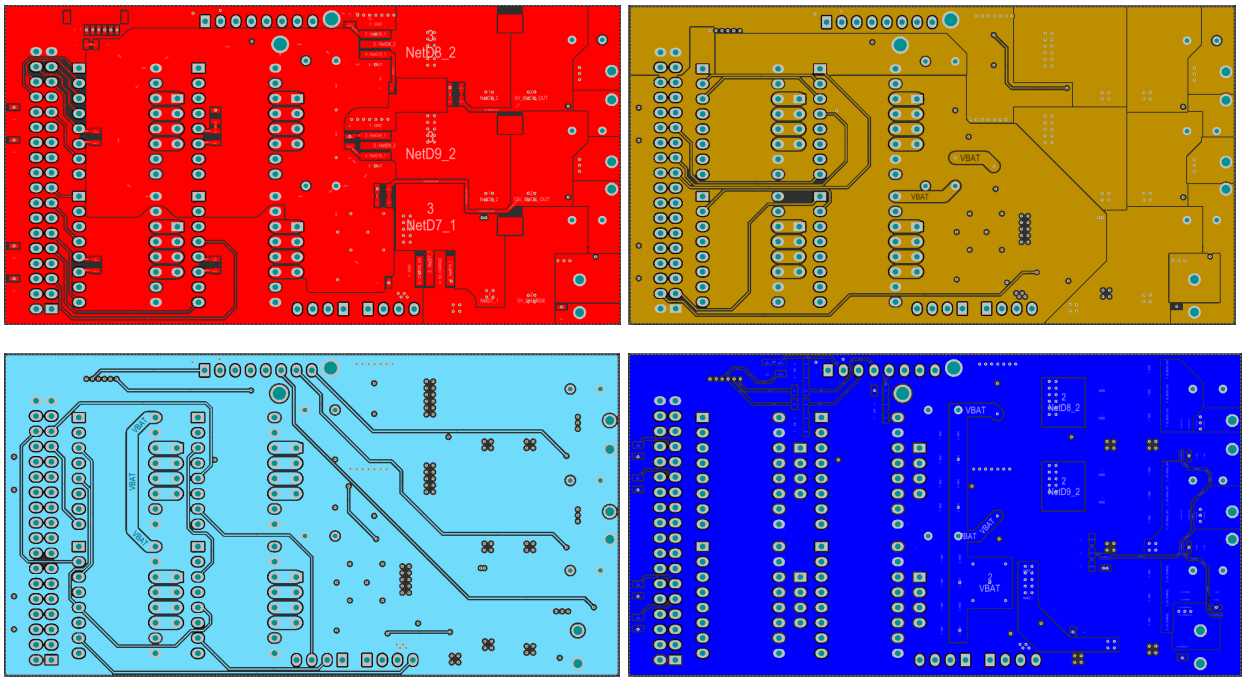


Figure 29. Four Layer Layout in Altium Designer

As mentioned, this was designed to work with an Arduino MEGA 2560, so all pin positions are matched for that board. Below show 3D renderings of the first prototype board, as well as the manufactured and assembled final product. It cost roughly 40 dollars (converted from RMB) for 10 pieces, with the specifications being 1Oz copper throughout, 1.6mm thick FR4, HASL surface treatment, and green solder mask. It was very cheap as one of the group members had it manufactured during the winter break trip to China. Although thicker copper was an option, we opted to not choose it as this was only revision 2 to save prototyping costs.

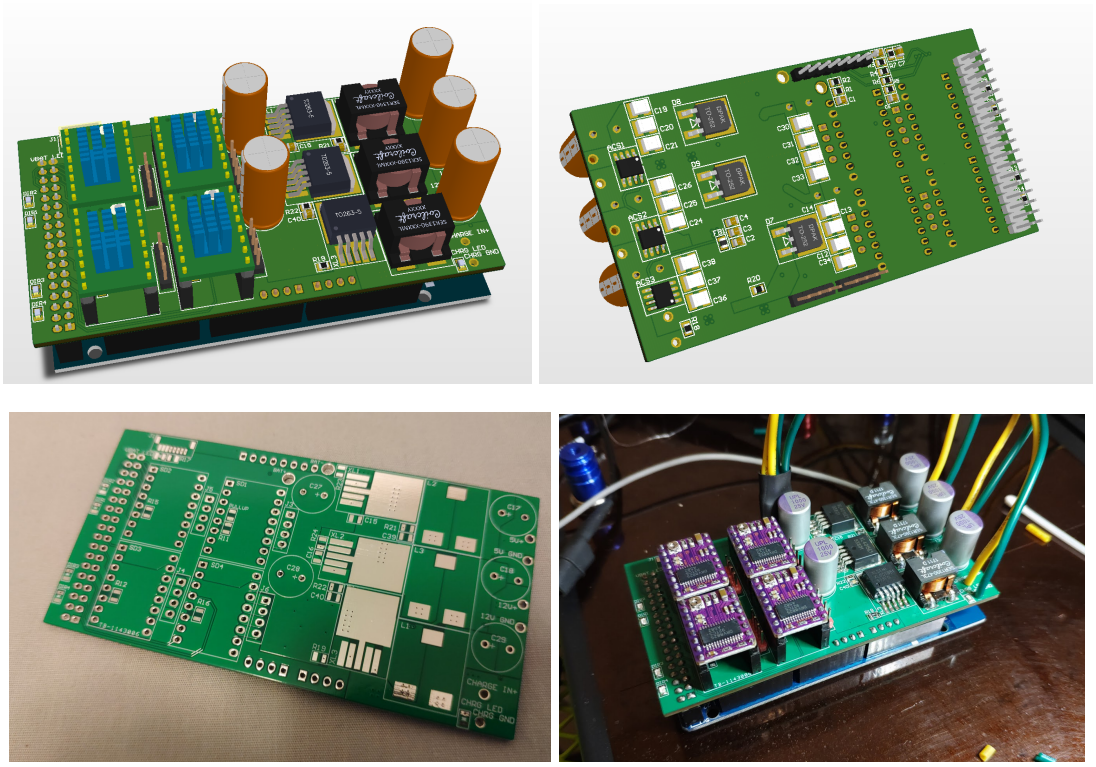


Figure 30. 3D Render of Design, Manufactured Bare Board, and Fully Assembled Board

Motor Controller Embedded Software:

Before discussing the work on the embedded software, it is important to layout the system diagram that we designed for the robot. At the top is the central computer, which is the Jetson Nano. However, from past experiences working with it, we know that it lacks robust low level IO and communication protocols. Hence, the second layer is the Arduino MEGA 2560, and its 5V logic IO will provide robustness in those areas to control the motors and handle power monitoring.

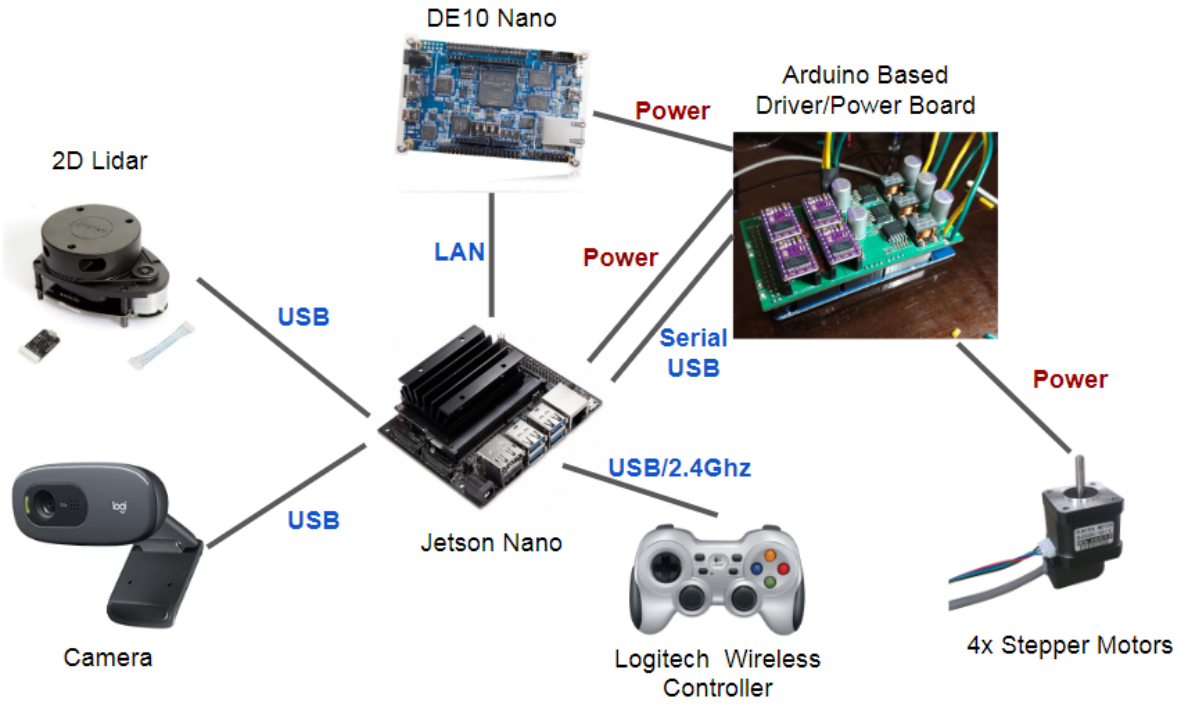


Figure 31. System Architecture Diagram

What is also very nice is that there is actually an Arduino IDE for Ubuntu, that can run on the Jetson Nano. Thus, from the robot connection alone, the central computer is able to modify and program the low level controller, which is very convenient. In essence, the robot becomes a complete development platform, a computer on wheels.

However, there needs to be a low latency, somewhat high frequency communication link between the main computer and the driver board to control the motors. Such a system does not exist, at least not one that can satisfy our needs over the serial USB link. Thus, we had to design one completely from scratch. But before that, the stepper motor control method must be discussed first to help understand the needs of the protocol.

Stepper Motor Control:

Our initial plan was just to use already existing libraries in Arduino, to control the stepper motors. However, testing with a variety of them with the initial test circuit described above yielded unsatisfactory results. Most libraries have very basic functionalities with poor documentation, and use simple IO toggling to accomplish stepping. This is not ideal especially with 4 stepper motors, as each would have to be controlled independently in terms of frequency. Thus, a custom driver control algorithm had to be created from the ground up, using hardware PWM.

The MEGA 2560's hardware PWM is fairly straight forward, although as usual the datasheet contained way more information than needed. There are 4x 16 bit timers, and 2x 8 bit timers with PWM outputs. Since we have exactly 4 motors, it was appropriate to use all 4 16 bit timers for that purpose, since the frequency of each needed to be adjusted independently. Even though we most likely could have achieved this by changing different settings of the output compare registers on the same timer, we did not need the timers elsewhere so this was the most efficient route. The timers are 1, 3, 4 and 5.

The registers for each of the timers need to be set to configure for proper PWM modes and frequencies. Below are the main setting registers, which are identical for all 4 16 bit timers. Timer 3 is chosen here for explanation. We set the WGM bits to all 1 for Fast PWM mode, as that gives all the functionality that we needed.

TCCR3A – Timer/Counter 3 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x90)	COM3A1	COM3A0	COM3B1	COM3B0	COM3C1	COM3C0	WGM31	WGM30	TCCR3A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR3B – Timer/Counter 3 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x91)	ICNC3	ICES3	–	WGM33	WGM32	CS32	CS31	CS30	TCCR3B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figure 32. 16 Bit Timer Registers [10]

Then, the COMnA1/A0 is set to 01, for toggling OCnA on compare match. This is the chosen setting because it essentially doubles the frequency capabilities, and we do not need duty cycle modulation (this forces 50% duty cycle squarewave). For all timers, only OCnA is

turned on, so that other unused pins do not output a PWM waveform (OCnB, etc.). Then lastly, the bits to set for the prescaler are CSn2/n1/n0, which follow the table below.

CSn2	CSn1	CSn0	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/1 (No prescaling
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge

Figure 33. Clock Divider Register Configuration Table [10]

We chose prescaler 1 for when the robot needs to move, because of our particular range of desired frequencies. And if the command is 0, the prescaler is set to 0 to turn off the PWM. From our testing with just a single stepper motor, we found that we need frequency in the range from 1 - 35,000 Hz. Therefore, our original hypothesis was that we needed to vary between different prescalars, in order to get the most accurate frequency number. The actual frequency derived from system parameters is the following.

$$frequency=(system\ clock)/\ [2 * prescaler * (TOP +1)]$$

Where system clock is 16Mhz, and TOP is the value assigned to OCRnA register for output compare. However, we are also interested in the reverse equation, shown below.

$$TOP = (system\_clock / (frequency * 2 * prescaler)) - 1$$

Since we command the frequency, and want to find the corresponding TOP value to put into the register. In MATLAB, we calculated the corresponding error of each choice of clock divider: 1, 8, 64, 256 and 1024, for each whole number frequency command from 1 to 35,000. This is done by first getting the TOP value from the equation above, then rounding it (since the microcontroller register only takes whole numbers in binary), then subtracting the corresponding rounded frequency from the original command frequency. Lastly, we compared the error from each, and stored the number of times from 1 to 35,000 Hz with 1 Hz increments that each divider resulted in the lowest error, shown in the table below.

Table 3. Clock Divider vs. Corresponding Lowest Error Frequencies

Clock Divider	1	8	64	256	1024
Number of Frequencies with the Lowest Error	34875	110	14	1	0

The results above are counterintuitive, if not surprising. It shows that clock divider 1 will result in the most accurate frequencies most of the time. Looking at the frequency relationship, it becomes clear; when the frequency is below 123 Hz, the higher clock dividers win, because the lowest error achievable by division 1 is capped at 123 Hz, even when TOP is 0. But after that, the higher clock dividers become quickly saturated with smaller TOP numbers with larger frequency jumps, as the equation is a 1/x relationship. Thus, clock division 1 works the best given the range of frequency that we desire.

There may be some concern with the lowest frequency achievable with clock division 1, as it seems relatively high at 123 Hz. However, with our testing and stepper motor driver microstepping divisions, it results in a very low rotational speed, which makes it acceptable as the lowest frequency command. We validated all frequency settings on oscilloscopes. However, due to the availability of hardware with the school shut down, we were unable to provide screenshots. All references above can be found in the atmega 2560 datasheet, pages 140-160.

Finally, with the correct settings, we identify the output pins using the Arduino MEGA and atmega 2560 datasheets. STP1 corresponds to OC3A which is digital pin 5, STP2 corresponds to OC5A which is digital pin 46, STP3 corresponds to OC1A which is digital pin 11, and STP4 corresponds to OC4A which is digital pin 6. The direction and enable pins are connected as shown in the PCB schematics in the appendix, they are just regular IO pins since only digital output is needed.

Controller Communications Protocol:

The communications between Jetson Nano and Arduino Mega is used to send the speed instructions of four motors to the mega, and let the mega generate the corresponding PWM signal to drive the motors. Since the high frequency motor speed control rate is required to meet up with the requirement of real-time control of the motion of the robot platform, the high communication rate is essential to this requirement. There are totally two versions of the protocol tested during the development. The final version of protocol reduced the buffer needed

to send and the delay time of the communication dramatically, which is the final version we used in the robot.

1. Original version: Intuitive communications protocol

The first version tested on the robot is an intuitive protocol. The message transfer between Jetson and Mega is the combination of desired direction value (0, 1), PWM frequency value and acceleration value split by some special characters. The basic format of the message is:

@	DIR_1, PWM_1	#	DIR_2, PWM_2	!	DIR_3, PWM_3	&	DIR_4, PWM_4	%	ACC

Since the PWW and ACC messages are long integers (represented in string form), separate different values in the string is the key problem in this protocol. We used special characters to signal the next integer begins which can avoid the conflict between the signal character and the message itself. Another advantage of this protocol is we can only send the value we need to change at one time rather than send the whole five value every time because the special characters can help us to identify which value it is. For example, we can only send PWM_2 and PWM_4 by sending in the following format.

#	DIR_2, PWM_2	&	DIR_4, PWM_4

However, the downside of this protocol is that the length of the message is apparently too long which increases the sending and receiving time, and also the chances of error happens during the transformation. If the serial connection drops some characters, the message will not be parsed correctly and will impact the robot motion directly.

During the test, we found that using this protocol, while it is able to function, has much room for improvement since the key value characters are not needed if we can directly encode the values by order in character variables, instead of sending number characters through the serial interface. The improved protocol will be presented in the final design.

Attachment - Mini 6 DOF Robot Arm:

Figure 34. Mini 6 DOF Robot Arm

While the design of the arm is not the focus of this report, as that could take another entire report in itself, we did make some progress that is relevant with regards to the control of the arm. The arm uses Lewansoul LX-16a serial servos for each joint, which communicate via a serial protocol through USB connection. The servos can be commanded for position control, where the servo's internal PID controller is in charge of rotating the joint to the desired angle, or it can be commanded for rotational power only. In both cases the angle readback is always available, and the second case essentially turns the joint into a heavily gear-reduced motor.

In terms of robot arm control, the second control option is much more useful for a smooth, customizable path planner, in the form of velocity control. Using a jacobian pseudo-inverse with the desired end-effector velocity command, the joints can be easily commanded to realize the desired path. The first option will make the algorithm simpler, but also result in a jagged path as each joint relies on its internal controllers.

Regardless, it was important to characterize the accuracy and precision of the internal sensors of each of the servos, as they are needed for angular readback. In addition, the control frequency was also investigated. Below show some test results, of approximately 20 runs across the entire angular range, as well as the servo tester that we designed, with an angular dial on the front face. The red points are angular precision errors (actual position - readback position), whereas the blue points are angular accuracy errors (actual position - command position).

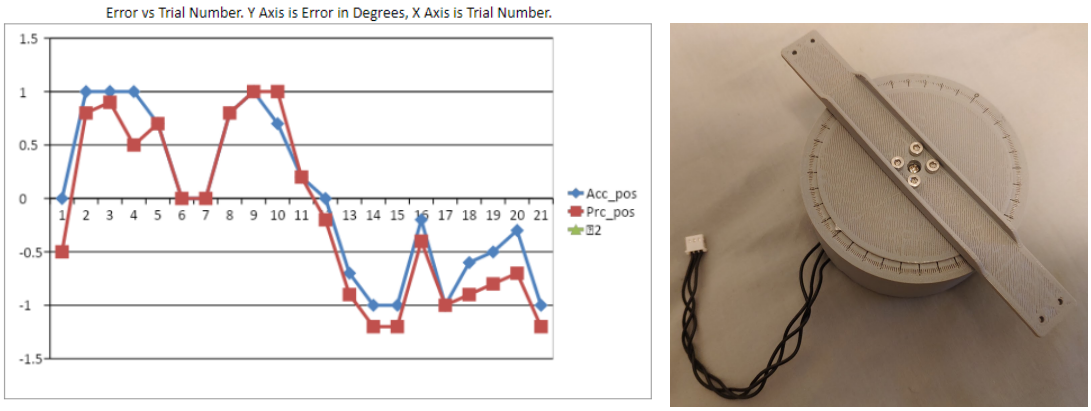


Figure 35. Error Plot, and Servo Test Setup

Lastly, the control frequency was tested. Here, a higher frequency is better. We found that the serial interface was locked at a baud rate of 11520, and below show the results of several trials of sending/receiving 1000 commands.

Table 4. Control Frequency in Different Trials

total time (ms)	920213	913338	931972	918711	923855	921617.8
control frequency (Hz)	108.67	109.49	107.30	108.85	108.24	108.5

Note that this is just with one servo. However, we found that the blocking element is a delay that is present in the serial send/receive command, meaning that there is potential for sending commands to other servos while waiting for the earlier ones to return the reading. Therefore with 6 servos, we would realistically be able to keep approximately the same control frequency, which is fairly decent for such cheap hardware, and would function well in a dynamic control loop.

Final Design:

Mechanical Hardware - Mobile Robot Base:

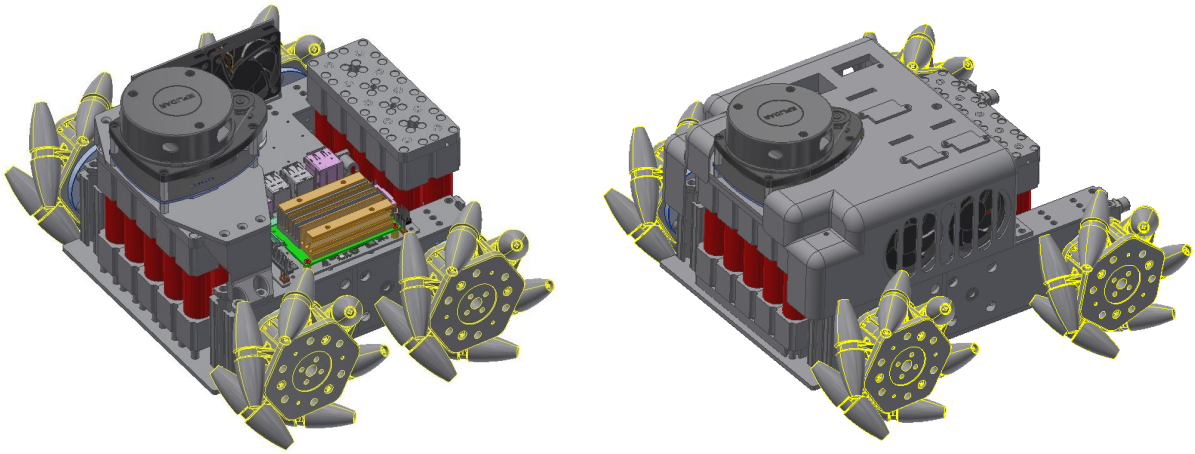


Figure 36. Prototype 1 (Left) vs. Final Robot Base Mechanical Design (Right)

Prototype 1 of the mobile robot base functioned well, but it had one fatal flaw. In the pursuit of ultimate compactness, the design was made such that the wheelbase was shorter than the track. Because the wheels were so wide, this made the robot wider than it is long. While it was not an issue without any attachment - even an advantage which can give it more maneuverability, with the robot arm it was an issue - since it created a rather hefty, cantilevered load. Physical testing showed some uneven loading, and may cause the robot to tip over. See figure 13 for example.

Therefore, the decision was made to redesign the chassis, to extend the wheelbase so that the robot is at least approximately square. The old wheelbase was about 120mm, while the track was about 230mm. The new wheelbase became 170mm, with the track unchanged. Moreover, to prevent the cantilevered load issue, the natural load of the mobile robot base was shifted towards the far side of the attachment point. See top view below, this not only created a more-or-less square overall footprint, but it also cleared extra space for the attachment. So while some empty space was wasted in the longer drive units, the overall space utilization is not affected.

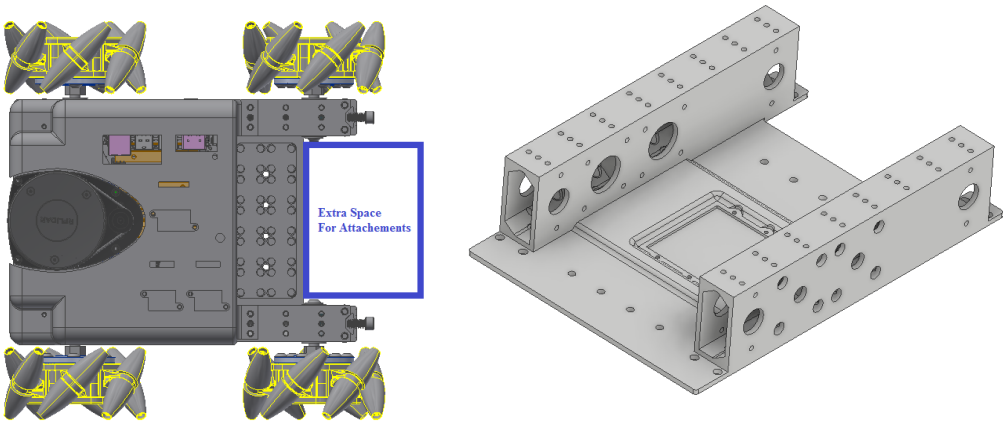


Figure 37. Robot Chassis Changes

In addition to the optimizations listed above, the single-part chassis also had some minor changes. The floor added an indent for seating the battery monitoring system, along with clearance for a heatsink. The BMS runs all current through mosfets for fusing, therefore under large loads it is expected to generate some heat.

Lastly, as shown in the renderings above, an enclosure was designed for the robot chassis. A separate report by group member Peng Peng will discuss that in more detail, along with the vacuum attachment. The assembled prototype, along with the arm attached, is shown below.

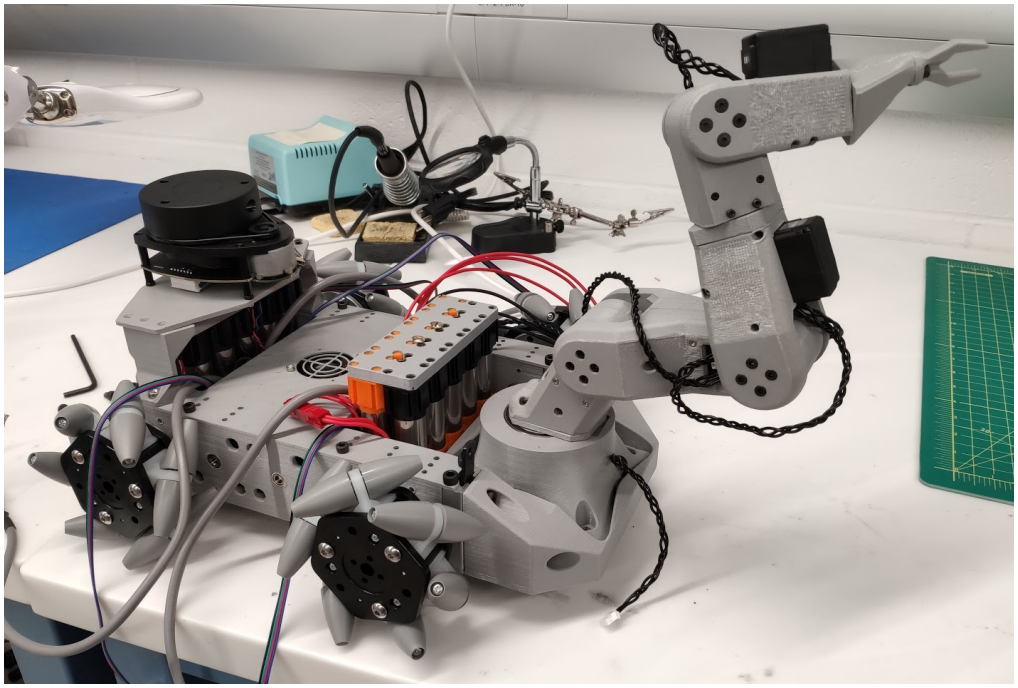


Figure 38. Assembled Chassis with Robot Arm

Electrical Hardware - Motor Controller, Power Supply and Hardware Health Monitor:

The first revision of the professionally manufactured PCB was for function check, and to test with embedded code to ensure compatibility. Thankfully, there were no issues with the board or the design. A thermal camera was used to validate the performance under load, and various scripts used during testing can be found in the github repositories listed in the appendix.

Since the board itself, or even the microcontroller, is unable to generate large electrical loads, the load testing scripts are for the high level computers to which the boards provide power. It is essentially a while loop in python computing floating point operations, and the shell script launches many copies of the same task that lasts 20 seconds in the background. It loads up the CPU only, although we can also write a CUDA enabled load script for testing the Jetson Nano's GPU as well. The results are presented in a later section.

The second revision of the board included minor changes in the power plane, and a few additional lead out connections for the connectors. But most importantly, as it was a proven design, we added good looking graphics with gold plated text and logos, shown below.

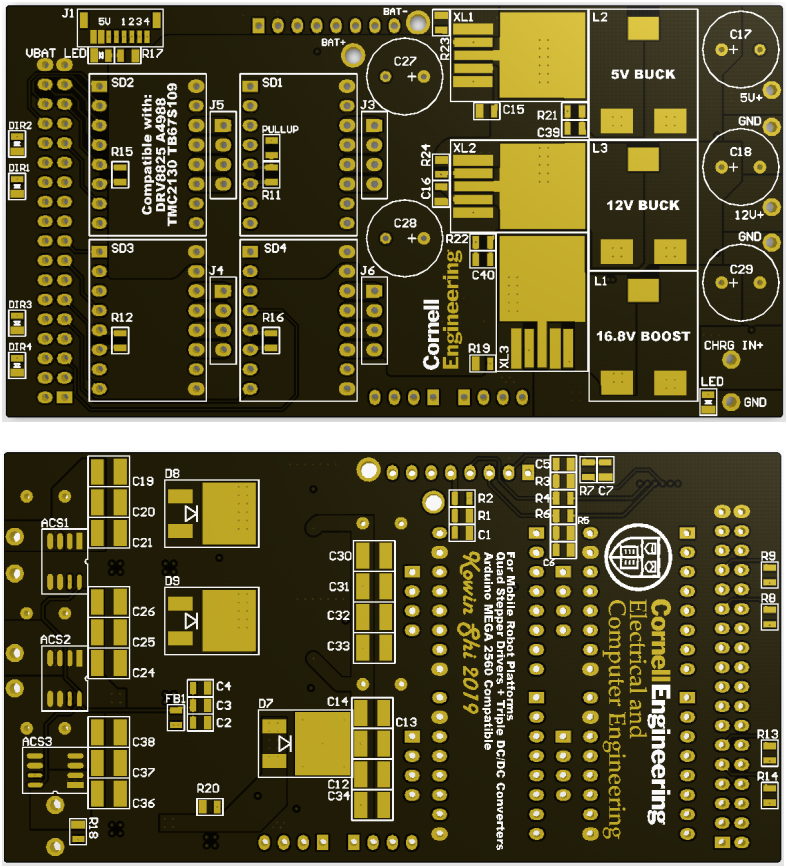


Figure 39. PCB Top and Bottom Renders

Thus a new batch of PCBs were made, still 4 layers, but this time the outer layers were changed to 2oz copper, while the inner were kept at 1oz. Black solder mask was chosen to give it a more professional look. The overall cost was 670 RMB, roughly 100 US Dollars for 10 pieces. The final assembled board looks like this, with an attached heatsink for the power supplies, as well as the stepper drivers.

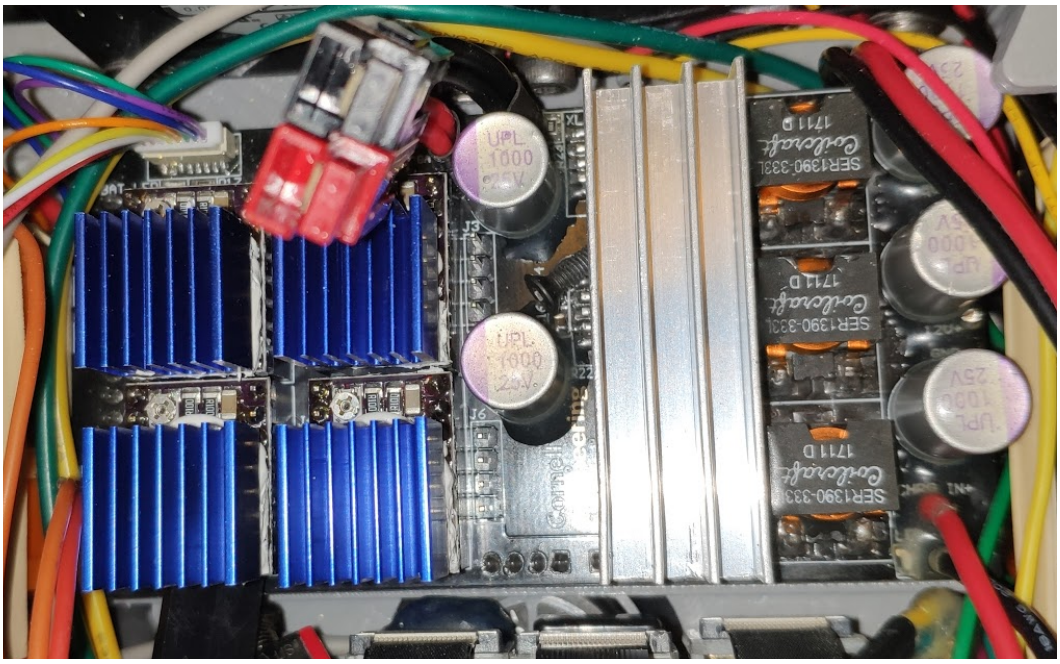


Figure 40. Final Board Assembled on Robot

Motor Controller Embedded Software:

Hardware Health Monitoring:

As shown in the schematics in the appendix, all sensors are connected to the ADC ports. Thus, work that the microcontroller has to do is to convert the sensor inputs into understandable formats. For the battery voltages, the equation is simple:

sensorValue*(R1/R2)*ref_volt/(1024)

Where sensor value is the read in integer, R1/R2 is the voltage dividing resistor multiplier for higher voltage cells, and ref_volt is the arduino analog reference voltage. Since the ADC readings are 10 bit resolution, we divide the raw reading by 1024. Then, for calculating per cell voltage, we subtract the voltage from each lower series level from the corresponding higher level, for example:

Table 5. Voltage Monitoring Calculations

Lower Level Reading	Higher Level Reading	Difference = Cell Voltage
12.5V	16.6V	4.1V

Lastly, for the Allegro ACS723 Current sensors we chose to use, the equation is the following:

(sensorValue - 100)*ref_volt/(1024*0.4)

Where the sensitivity is 400mV/A, and there is some offset voltage to subtract from the raw reading. With all numbers converted to human understandable format, they are converted once more from float numbers to strings to be sent across the serial bus.

Controller Communications Protocol:

The second revision of the communication protocol solves the low communication rate problem by using one char (8 bits) to represent 256 different numbers and the sign of the number (+ / -) to represent the direction. In that case, 2 char (16 bits) is large enough to cover the whole PWM value range (-32768 - + 32767). And instead of separate values by special characters, which will take 5 different chars, we just resend the instructions for 4 motors and acceleration value every time. Even though, the total length of the message is still much shorter than the previous protocol version. The basic format of the protocol is:

PWM_11	PWM_12	PWM_21	PWM_22	PWM_31	PWM_32	PWM_41	PWM_42	ACC

So each block in the table is one char only and the total length of the message is irrelevant to the digits number of the PWM value but is fixed length, 9 chars namely. The first value of each PWM message is a signed short integer parsed into the char and the second value is an unsigned short integer.

I. Parse PWM values in to char:

This part of code is running on the Jetson side written in Python. Since the signed numbers are stored in two’s complement, the first step is to compute the two’s complement of the desired PWM value. The function we wrote is:

```
def twos_complement(val, bits):
        return 2 ** bits - abs(val) if val < 0 else val
```

The next step is to get low 8 bits and the high 8 bits of the complement. For low 8 bits, a mask is used (0x007F), and for high 8 bits, we right shift the original value by 7 bits.

Then if we store the result above into a bytearray and send them through serial, those data will be automatically encoded into characters based on utf-8.

II. Decode the string

This part of code is running on the Mega written in C.

```
data[0]=((((int)buff[0])<<9)|(((int)buff[1]&0x7F)<<2))/2;
data[1]=((((int)buff[2])<<9)|(((int)buff[3]&0x7F)<<2))/2;
data[2]=((((int)buff[4])<<9)|(((int)buff[5]&0x7F)<<2))/2;
data[3]=((((int)buff[6])<<9)|(((int)buff[7]&0x7F)<<2))/2;
data[4] = (int)buff[8] & 0xFF;
```

The basic idea is typecasting the char into int and using logical operation to get the value back. The optimized protocol reduced the buffer needed to send dramatically. During the test, we found that using this protocol, the communication rate worked well at 20Hz and most likely can be turned up even further, which meets the requirement of real-time control. The arduino's baud rate was lowered from the standard 115200 to 57600, to be more robust against noise.

Stepper Motor Control:

On top of regular motor speed commanding, more sophisticated control was possible and needed with stepper motors. Unlike conventional motors which simply take a power command, stepper motors take a fixed speed command, and immediately run at that speed. If

the motor is not powerful enough to operate at that speed immediately under load, then a phenomenon known as stalling occurs. For stepper motors, this is when the motor begins cogging, or vibrating at high frequencies without actually rotating at the commanded speed. It is also known as skipping steps.

So to avoid this, the method implemented here, inspired by open source 3D printer firmware such as marlin, is to follow a custom acceleration profile. This was accomplished by taking the raw command received from the high level controller, then establishing a variable frequency interrupt service routine (ISR). In the ISR, the current motor frequencies are either incremented or decremented by a fixed amount, depending on if it is below or above the command threshold.But if it is within the increment away from the target, the ISR will make no changes to it. Outside the ISR, the frequencies are assigned to the motors on every loop, to provide high speed control updates. The code snippet is pasted below.

```
ISR(TIMER2_OVF_vect)          // timer compare interrupt service routine
{
   // motor frequency ++
   for (int i = 0; i < 4; i++) {
      int diff = data[i] - fqcy[i];
      if (diff > stp_sz)
         fqcy[i] += stp_sz;
      else if (diff < -stp_sz)
         fqcy[i] -= stp_sz;
      else
         fqcy[i] = data[i];
      motorFrequency(i + 1, fqcy[i]);
  }

  TCNT2 = data[4];
  TIFR2 = 0x00;
}
```

With this mechanism, the timer compare then directly controls the acceleration rate, along with the increment steps. We made the steps fixed to a small number (stp_sz) so that the acceleration is smooth. The timer compare value is then obtained as a part of the command chars described above, in particular the field ACC with 7 bit resolution (128 levels). The final values used were obtained by trial and error.

High Level Software - Manual Remote Control:

From our research, we came across three potential controllers - Logitech Gamepad F310, F510, and F710. While all three are very powerful and highly customizable with profiler software and support multiple operating systems, the F710 also provides 2.4 GHz wireless connectivity. Given the needs of remote control, we decided upon the F710 - as it would allow us to manually remote control the robot with a high connectivity rate.

We connected the controller to the high-level Nvidia Jetson Nano platform since it is able to communicate with the low-level Arduino platform that is programmed to control the motor directly. We mainly programmed the joysticks of the controller to achieve the functionality of direction control and speed control. We also set up a motor switch button in order to control the motor state efficiently. In the following section, we will describe the details in terms of the implementation of these functionalities.

Firstly, we programmed to make sure the Nvidia Jetson Nano is able to detect the controller device automatically without any manual configuration.

```
events = devices.gamepads[0].get_char_device_path()
gamepad = InputDevice(events)
```

Basically, we defined functions to detect the current joystick events on the Nvidia Jetson Nano. With detected events, we assigned corresponding values to the variables indicating the controller actions that are further converted to byte arrays sent to low-level systems.

```
def gamepad_control(gamepad, speed):
    for event in gamepad.read_loop():
        do something (joystick event detection)

if __name__ == '__main__':
    gamepad_thread = threading.Thread(target=gamepad_control, args=(gamepad, speed))
```

```
gamepad_thread.start()

while True:
    num1 = twos_complement(i1, 14)
    num2 = twos_complement(i2, 14)
    num3 = twos_complement(i3, 14)
    num4 = twos_complement(i4, 14)
    num5 = 150
    char1 = num1>>7
    char2 = num1 & lower_mask
    char3 = num2>>7
    char4 = num2 & lower_mask
    char5 = num3>>7
    char6 = num3 & lower_mask
    char7 = num4>>7
    char8 = num4 & lower_mask
    char9 = num5
    print_res = [char1,char2,char3,char4,char5,char6,char7,char8,char9,255]
    byte_res = bytearray(print_res)
    ser_obj.write(byte_res)
    linelist = ser_obj.readlines(ser_obj.in_waiting)
    for line in linelist:
        print(line.decode('utf-8'))
    time.sleep(0.05)
```

The assignment and the conversion are essentially two threads running simultaneously and constantly to make sure we would not miss any actions. To handle the blocking issue that might be incurred by the two threads, we utilized the Python threading library.

Specifically, we used the left joystick to control the left wheels and the right joystick to control the right wheels so the robot is able to achieve actions including move forward, move backward, and turn left/right.

```
# left stick operation
if stick_id == 'ABS_X':
    event_time = absevent.event.timestamp()
elif stick_id == 'ABS_Y':
    if abs(absevent.event.value) < 160:
        oneval = 0
    else:
        oneval = absevent.event.value
    i1 = i2 = (int)(oneval/4)

# right stick operation
elif stick_id == 'ABS_RX':
    event_time = absevent.event.timestamp()
elif stick_id == 'ABS_RY':
    if abs(absevent.event.value) < 160:
        two_val = 0
    else:
        two_val = absevent.event.value
    i3 = i4 = (int)(two_val/4)
```

The wheel's speed is controlled by how much we push the joysticks since the more we push the joysticks, the larger the joysticks' value is. We also restricted the minimum value to avoid the potential control glitch. Therefore, the whole process can be understood as we quantize the physical actions to the numerical format. As soon as the values are assigned, we do bit operations such as two's complement and bit shifting to convert them to the valid byte array that are acceptable by the low-level system.

Results:

Hardware:

Design:

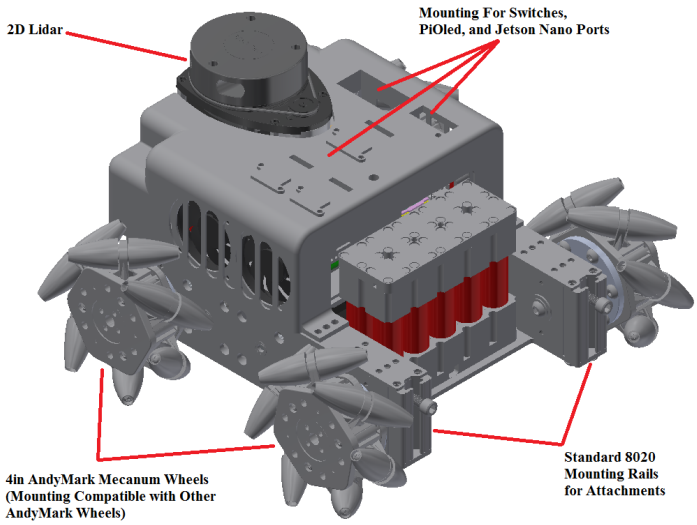


Figure 41. Labeled Robot Base Assembly in CAD

As shown earlier, this is the robot platform that was designed, analyzed, improved, and finally manufactured in its latest revision. The fairing and vacuum portions are covered by group member Peng Peng, in the separate report attached. The total dimensions of the robot, excluding any attachments, come out to be 290.5mm (W) x 272.5mm (L) x 180mm (H).

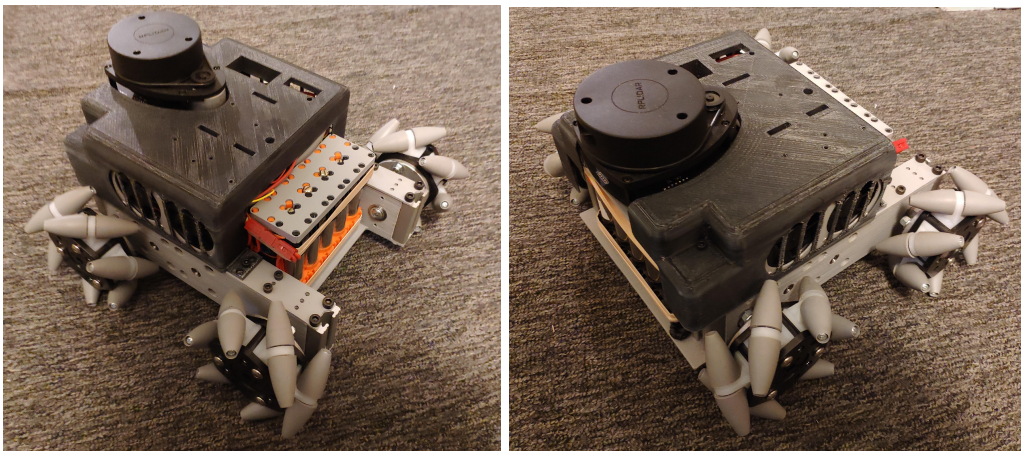


Figure 42. Assembled Robot Base

Besides the mechanical design, another main achievement of this project is the design, analysis and manufacturing of the integrated driver power board, shown below. The final stepper drivers chosen were the Pololu DRV8825 Stepsticks, as they were the best in terms of

current delivery and heat dissipation. Large heatsinks were added, and the drivers were tuned for 2A current operation.

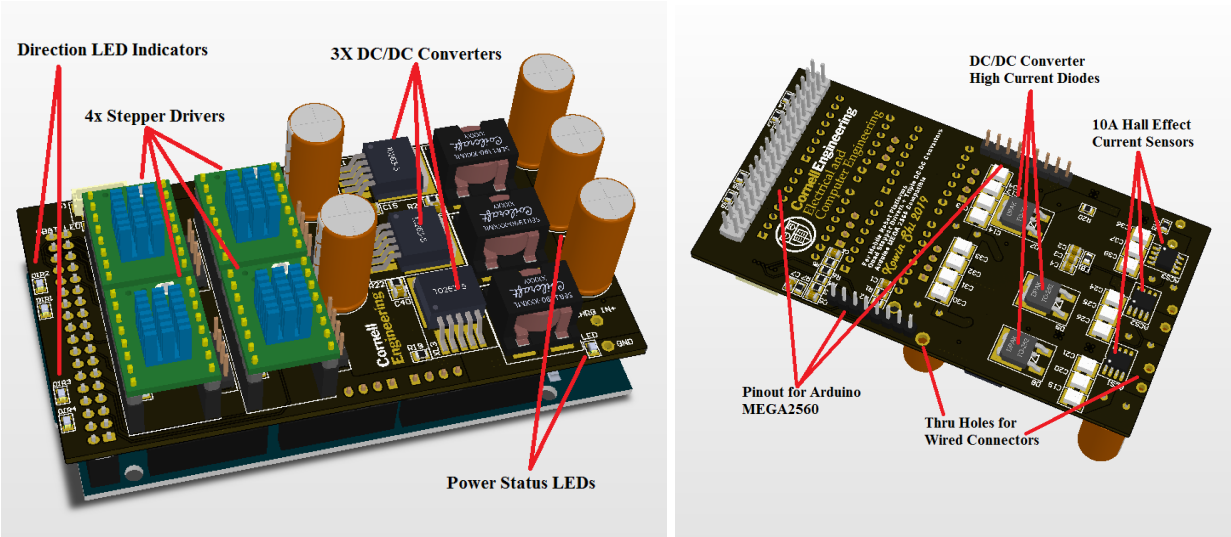


Figure 43. Labeled Driver Board

Performance:

The performance of the robot chassis can be seen in the videos linked here: https://www.youtube.com/watch?v=5nwbNZDMz04&feature=youtu.be

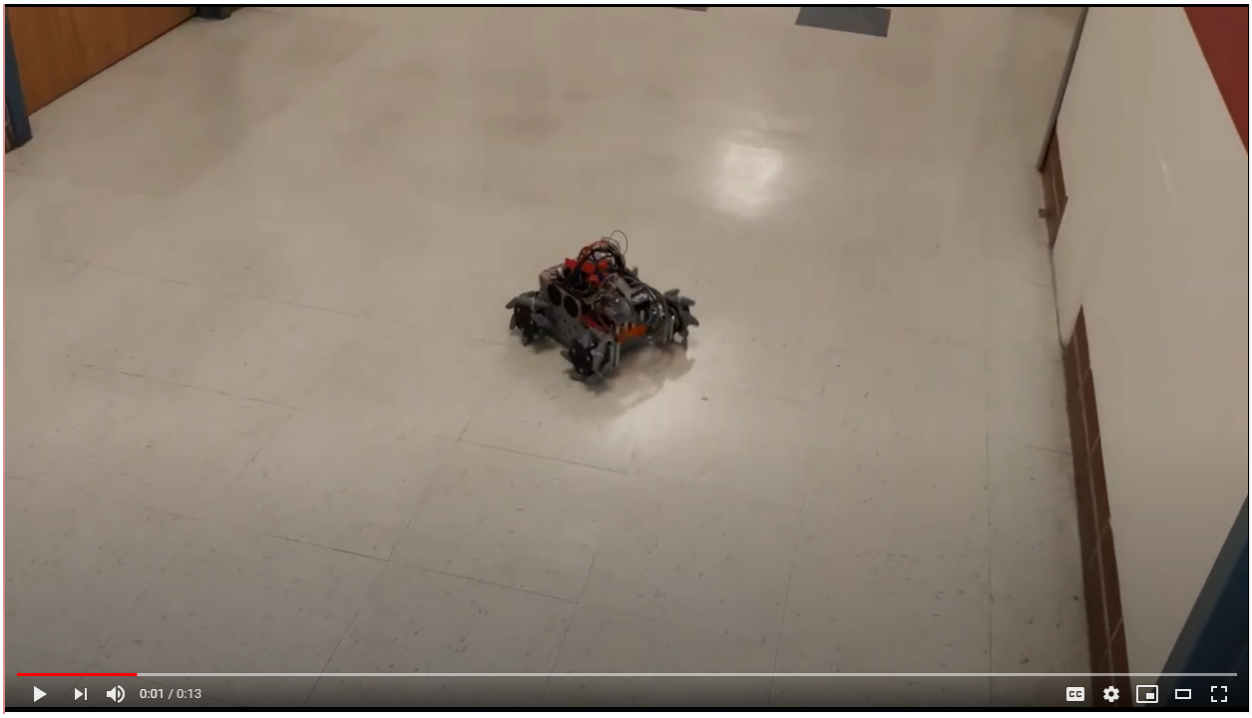


Figure 44. Robot Remote Control Video

The top speed calculated from the final tested values is shown in the table below. Note that there are two operating modes, one high speed and one low speed allowing for more controller precision. The values below are the peak values the robot is able to sustain under testing. The runtime has not been rigorously tested, but it is estimated to be more than 1 hour with just the robot platform running based on battery drainage times.

Table 6. Robot Speed Calculations

PWM Frequency	Motor rev/s	Speed m/s	Speed km/hr	Speed mph
10922	6.83	2.18	7.84	4.87
4096	2.56	0.82	2.94	1.83

The joystick remote operation of the robot can be seen here, along with a demo of the operating system user interface: https://www.youtube.com/watch?v=QqKDXBQhndI

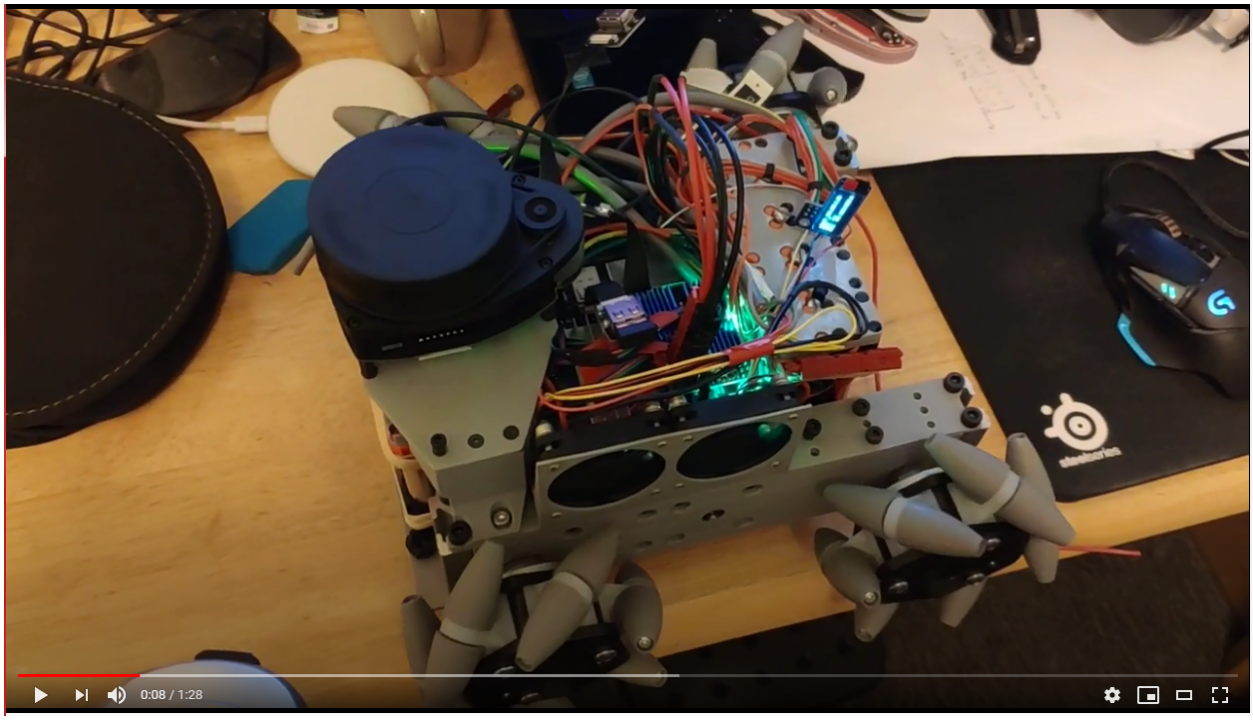


Figure 45. Remote Control Demonstration Video

Some notable features implemented in addition to the technical design explained in the earlier section of the report were a custom wired pioled screen for displaying vital statistics, as well as a hardware safety switch that disables the stepper motors on system startup. The pioled screen shows wireless IP address, CPU/GPU usage, and memory usage. The first item is especially important because when operating in headless mode (without a GUI), it is impossible

to know the address at which the system can be accessed via ssh on a general wifi network. Thus, the screen enables the user to easily access that information without needing physical connection to the robot.

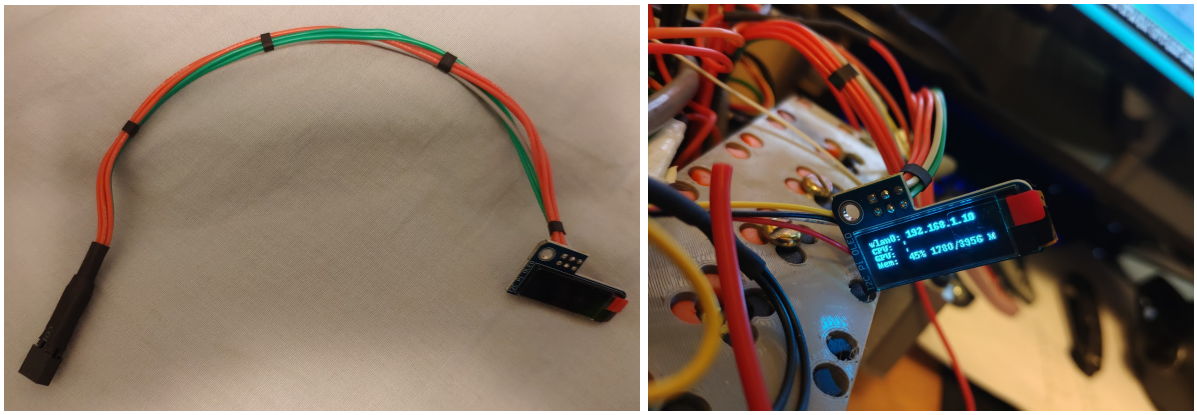


Figure 46. Custom PiOled Module

Lastly, we evaluated the thermal performance of the entire system. A demonstration video is linked here that shows the thermal camera setup we used: https://www.youtube.com/watch?v=D3uQbSVx_2w.

Figure 47. Thermal Validation Video

Thermal images were obtained of the system under load, both for the power supply section as well as the stepper motor drivers. Below show the results.

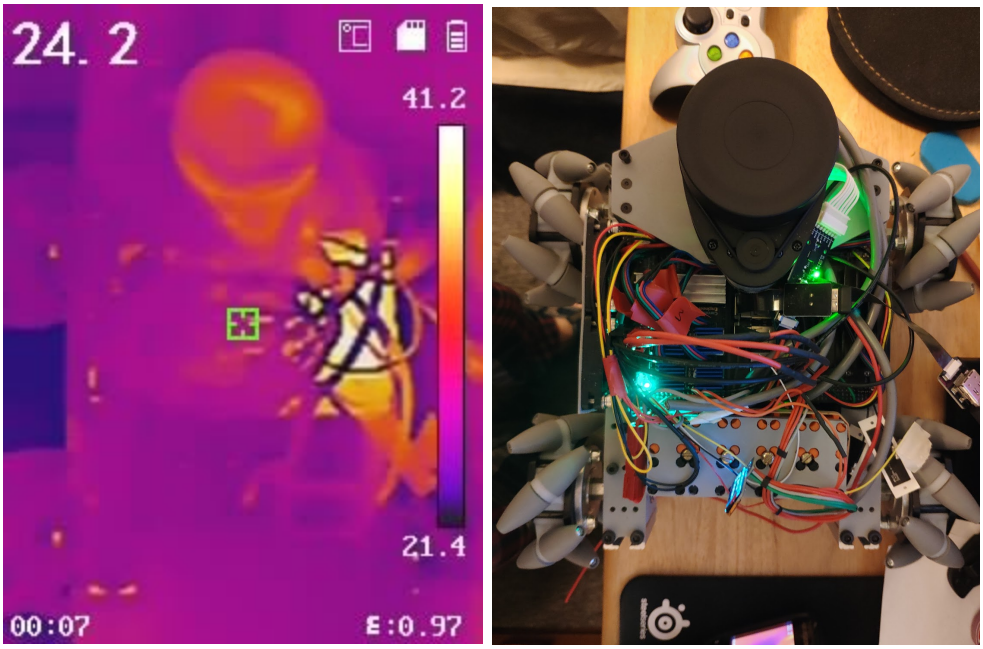


Figure 48. Thermal Evaluation, CPU Load Only

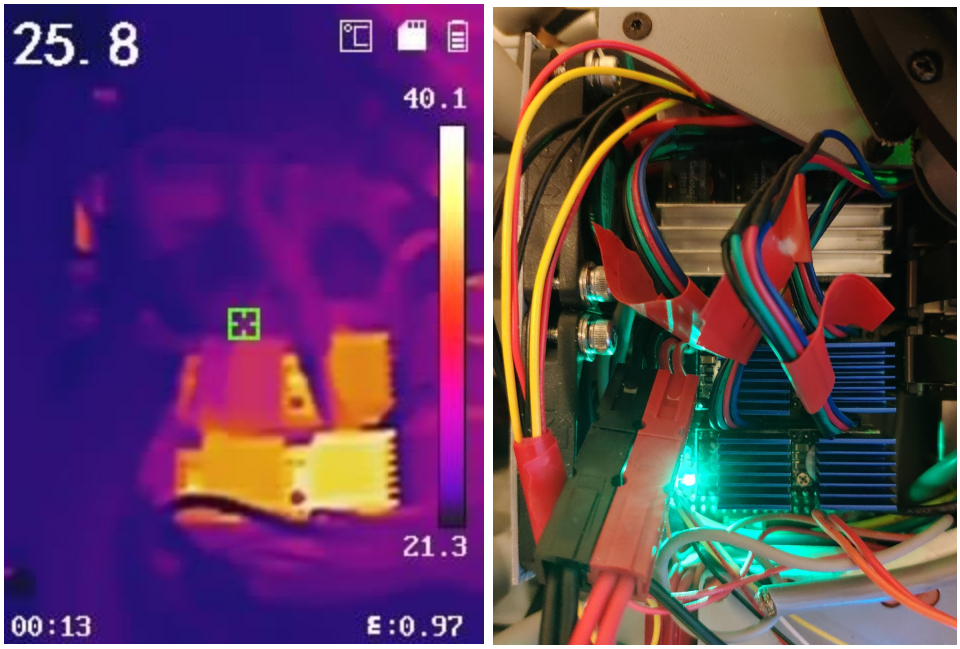


Figure 49. Thermal Evaluation, CPU Load + Stepper Motor Load

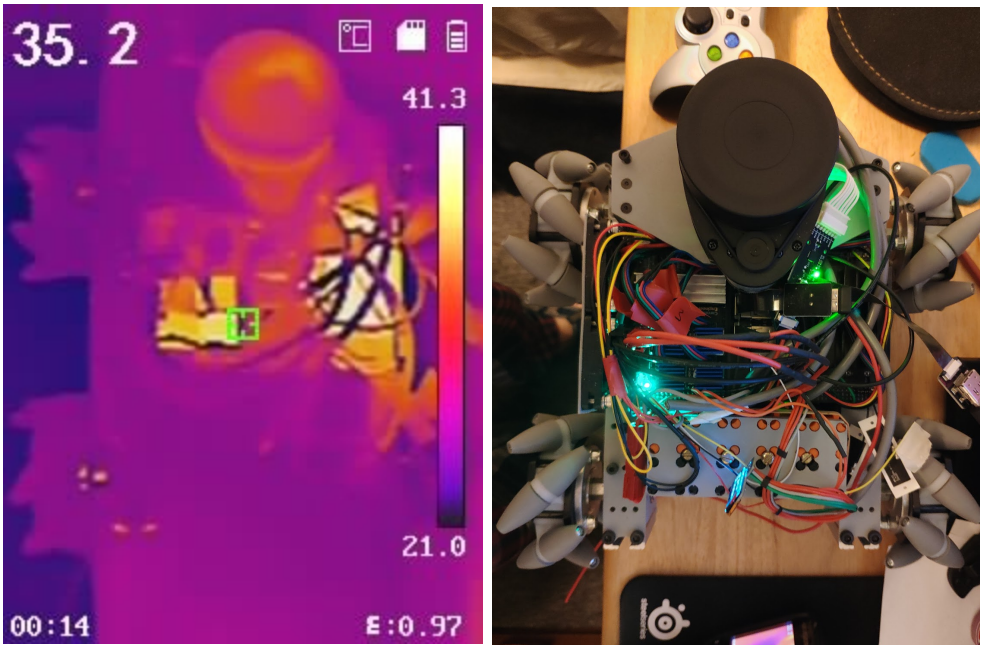


Figure 50. Thermal Evaluation, CPU Load + Stepper Motor Load

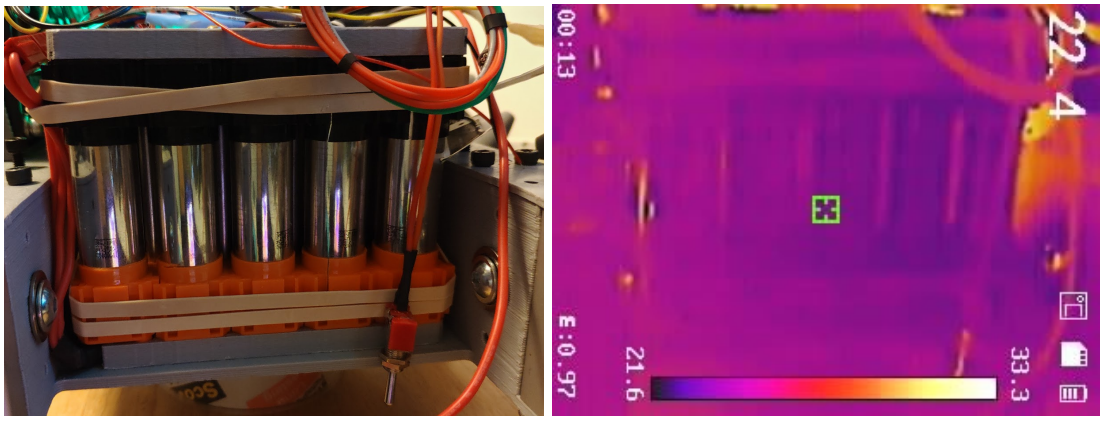


Figure 51. Front Battery Thermal View Under Load

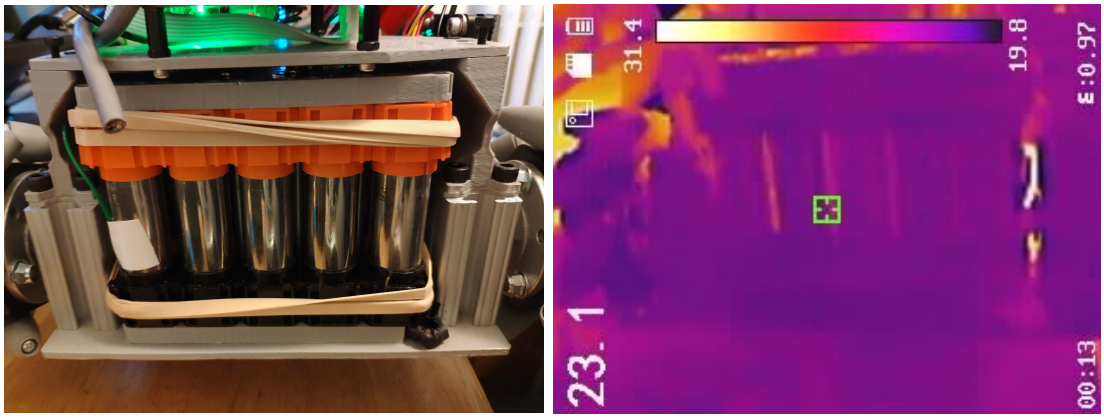


Figure 52. Rear Battery Thermal View Under Load

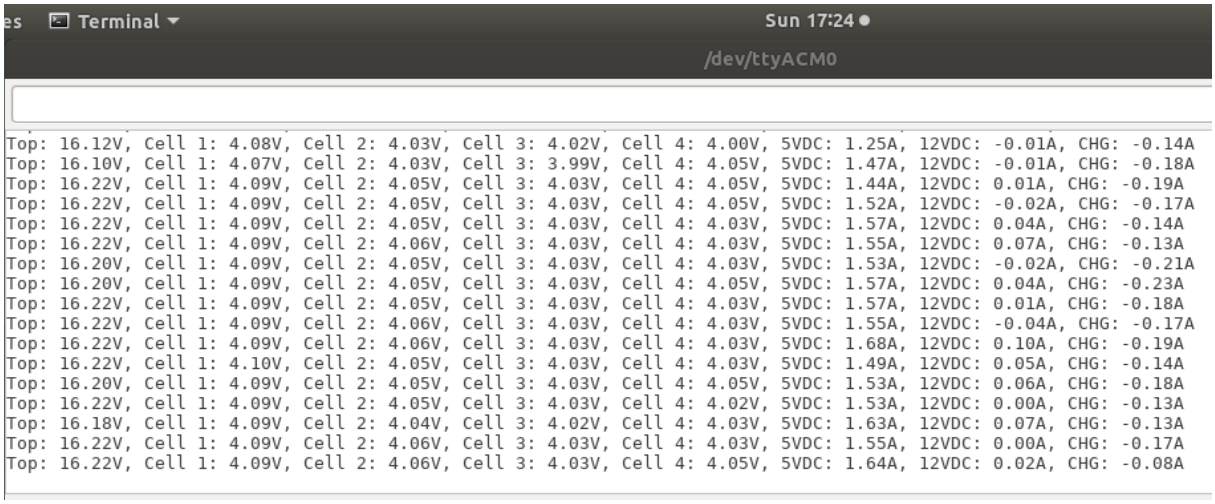


Figure 53. Power Conditions Under Testing

This load testing was not as thorough as we had liked, as the 5V rail was not taxed to its limits. However, it was a substantial current, even with just the CPU maxed out. The stepper thermal solution testing is valid though, as the motors were left powered for 20 minutes. As shown, no component in the system reached above 45 Celcius, which makes it great for longevity and safety. As shown above, the hardware monitoring works well, showing the overall battery voltage, the per cell voltage as well as the per channel current.

Software:

Design:

The final software components we developed consist of two embedded programs running on the arduino: one handles the messaging and stepper motor PWM acceleration; the other handles the hardware health monitoring. We did not have enough time to integrate the two, but combining them is not a difficult task. The files can be found under Open-AMR-Software\TMC2130_MCU\Arduino_Code_Working\Serial_Accel_PWM for the motor controller program, and Open-AMR-Software\TMC2130_MCU\DriverBoardCode\curr_volt_monitoring for the hardware health monitoring.

Then, on the high level software side, we have a successfully working joystick control Python program. It currently only supports tank drive (non-holonomic mode), but it has two speed modes that can be switched by pressing the ‘A’ button on the Logitech controller.

Performance:

The user interface of the Jetson Nano main computer is shown below, exactly identical to what would be expected of an Ubuntu interface. Shown in the videos above, the remote control functions as expected and is very responsive, with minimum CPU load on the system.

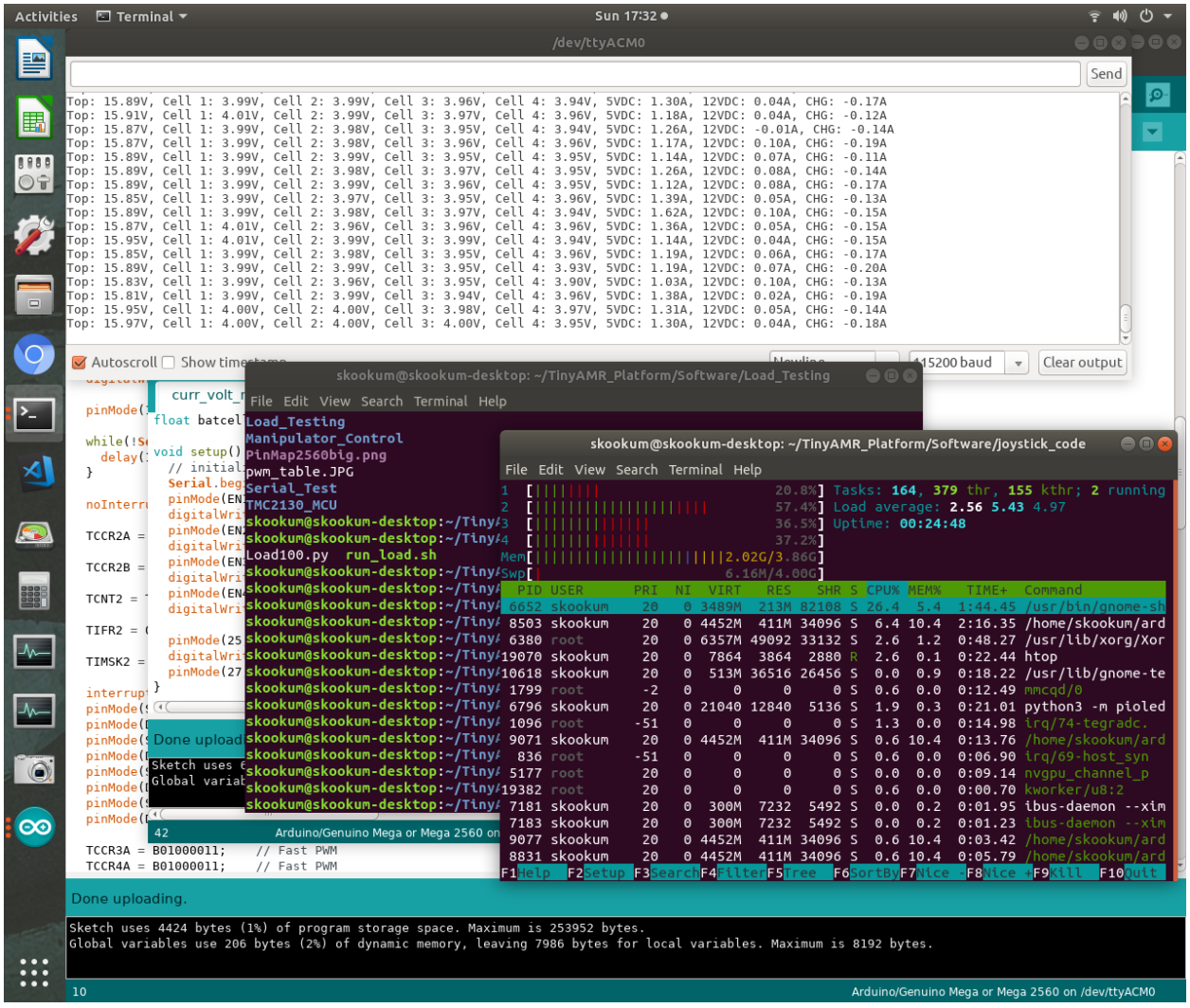


Figure 54. Jetson Nano Ubuntu User Interface

For the embedded motor control code, we decided to benchmark the interrupt service routine for handling the acceleration calculations. Because it 'interrupts' the main loop, we want to make sure it does not take so long that it impacts the messaging functionality, which is time sensitive. Using a microsecond timer, we found that with no serial input, the ISR takes 56 us, and with serial input, it takes 216 us. At 57600 baud rate, it takes (1/57600)*8 = 138.89 us to receive a byte (char). The ISR is longer than that, but it functions just as expected, most likely because the delayed character is just stored in the serial buffer.

Problems:

Hardware:

As mentioned in the component selection section, there are some issues with the VRUZEND battery kits. Due to what looks like poor design, the slots for connecting the different end caps have a stress concentration area, and splits open after a while under stress. This may be also because on top of the 18650 cells, we added a clear plastic shrink wrap to protect the cells against shorting. So this added thickness caused the cells to be a little bigger than the cells for which the caps are originally designed. This is why there are rubber bands on the outside of the battery, while it looks ugly, it does the job and keeps the pack together.

Figure 55. Battery Pack Problem

For the motors, because the drivetrain is largely an experimental design, we were unable to find any previous work to reference for using stepper motors as the main power element. Even though it works well, we found that it was lacking in torque, at least at the high speed ranges. Although as an indoor robot, there is not much of a point going above the speeds we were able to achieve. But having a torque-ier motor would definitely make the design more robust. With the current design, running the motor at higher speeds or higher acceleration rates would cause the step-skipping phenomenon described earlier. But even though it sounds bad (both literally and figuratively), it is actually not harmful for the motors or the stepper drivers. In fact, when it is skipping steps, the stepper drivers are not providing the power demanded under

normal circumstances, so it is actually much safer than stalling a normal motor. So this may be a feature instead of a bug

The above comments refer to the robot under the tank drive setup. When using the robot in the holonomic configuration, the lack of torque of the motors becomes more significant. Originally, we intended for the robot to be able to translate to any direction. However, horizontal translation requires a decent amount of torque to counteract the friction in the wheel's rollers, since the wheels are turning against each other. Thus in our testing, we were only able to achieve low speed holonomic motion.

Lastly, the wire management is an issue. Because the hardware packaging is so dense and there are so many connections for both signal and power, the resulting wire bundle is quite intimidating to handle. Even though it can be packaged under the fairing, it is not ideal and impedes airflow.

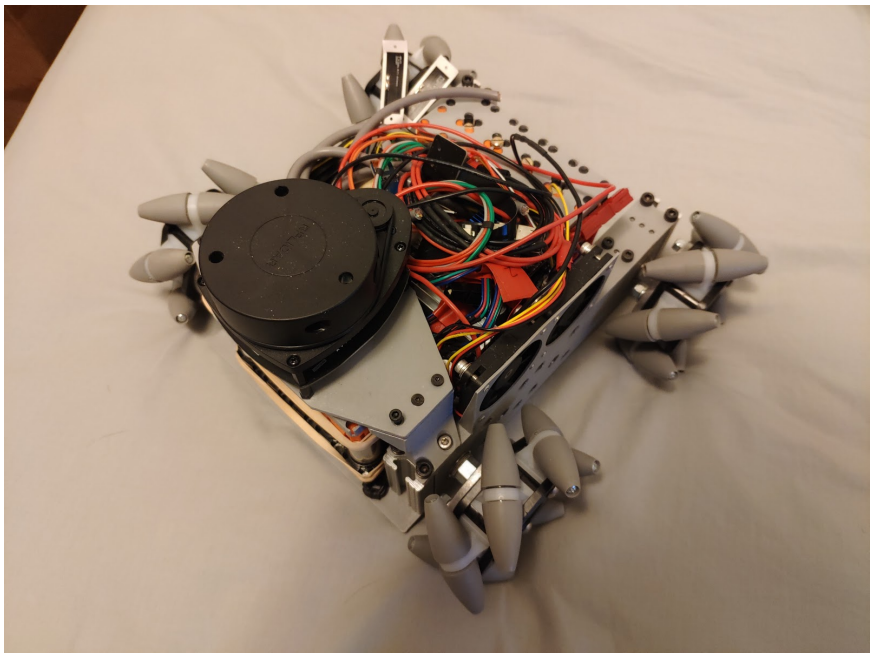


Figure 56. Wire Management Problem

Software:

There does not exist many software issues, mostly because development in that area was limited. To get the embedded code to work with the high level code took a long time, as debugging was difficult. But with that done, high level development can be accomplished easily.

Future Plans

Hardware:

The number one issue to address in hardware is the battery assembly. The pack will need to be taken apart, the battery connection disassembled, and a potentially 3D printed solution will be needed to replace the cracking plastic.

In terms of the motors, it does not seem to be a big issue. The speeds calculated earlier is more than enough for an indoor robot. If others are replicating this design and hope to gain more torque, they can simply use a longer stepper motor without the encoder attached.

Wire management is also not a complicated fix. Most of the wires are much longer than needed, since this is still only a prototype. So trimming those lengths down, and fixturing them to the robot chassis will solve most of the issues here. In addition, the power connector has triple 16 gauges wires because we did not have access to the appropriate wires, so replacing that with a single wire will certainly improve wire management.

Lastly, although we have tested the charging solution, including several wireless charging modules, we feel that it requires more work so did not include the design in the report. But our finding is that wireless charging is largely limited by thermal conditions, as AC rectification wastes a significant amount of energy. So this component warrants more detailed investigation.

Software:

There is a lot of progress to be made in the software area of this project. The first item would be to refine the hardware interfacing side, combining and optimizing the embedded programs, and modifying the interfacing python script.

The second item would be to encapsulate the high level software in ROS nodes. The robot command section would need to be separated from the remote control module in that case. This way, the RC module can be selectively switched on when autonomous mode is not in use.

Lastly, the RPLidar ROS nodes need to be integrated into the software stack, to enable SLAM.

Conclusion and Acknowledgement

The goal of the project is to create a hardware accelerated mobile robot platform. To that end, it was successful. We demonstrated a working mobile robot that can be easily autonomous, as well as possible attachments that can add functionality to it.

However, being a very ambitious project, we were not able to delve as deep into the software side as we wanted, due to both time limitations as well as circumstantial limitations due to the virus situation. That being said, we still feel like we were able to accomplish a significant amount of engineering work, as evidenced by this report. This is a project that we intend to continue, and refine the design to a polished working final product.

We would like to thank Professor Bruce Land for his tremendous support during this project. He was always ready to answer any questions, and helped out significantly with the funding. We would also like to thank Professor Matt Ulinski for his support in advising Peng Peng, which allowed us to add multidisciplinary expertise to our design project.

Citations

[1] K. Shojaei, A. M. Shahri, A. Tarakameh, and B. Tabibian, “Adaptive trajectory tracking control of a differential drive wheeled mobile robot,” Robotica, vol. 29, no. 3, pp. 391–402, May 2011, doi: 10.1017/S0263574710000202.

[2] F. Tóth, P. Krasňanský, M. Gulan and B. Rohal'-Ilkiv, "Control systems in omni-directional robotic vehicle with mecanum wheels," 2013 International Conference on Process Control (PC), Strbske Pleso, 2013, pp. 516-521, doi: 10.1109/PC.2013.6581463.

[3] J. Xiao, D. Xiong, W. Yao, Q. Yu, H. Lu, and Z. Zheng, “Building Software System and Simulation Environment for RoboCup MSL Soccer Robots Based on ROS and Gazebo,” in Robot Operating System (ROS), vol. 707, A. Koubaa, Ed. Cham: Springer International Publishing, 2017, pp. 597–631 [Online]. Available: http://link.springer.com/10.1007/978-3-319-54927-9_18. [Accessed: 11-May-2020]

[4] D. Evans, “The Great Robotic Vacuum Showdown Part 1: Roomba 650 — Mechanical System.” [Online]. Available: https://www.fictiv.com/blog/posts/the-great-robotic-vacuum-showdown-part-1-roomba-650-mechanical-system. [Accessed: 16-Apr-2020]

[5] “Maciej multi-cyclone vacuum cleaner build.” [Online]. Available: https://woodgears.ca/dust_collector/gmb/index.html. [Accessed: 16-Apr-2020]

[6] V. Beggi, L. Loisel, and Nguyen Xuan Truong, “Microgrid in USTH campus : Architecture and Power Management Strategies,” 2018, doi: 10.13140/RG.2.2.25145.42085. [Online]. Available: http://rgdoi.net/10.13140/RG.2.2.25145.42085. [Accessed: 11-May-2020]

[7] Texas Instruments WEBENCH® Power Designer, http://www.ti.com/design-resources/design-tools-simulation.html

[8] “XL6019 datasheet.” XLSemi [Online]. Available: http://www.xlsemi.com/datasheet/XL6019%20datasheet-English.pdf

[9] “XL4015 datasheet.” XLSemi [Online]. Available: http://www.xlsemi.com/datasheet/XL4015%20datasheet.pdf

[10] “Atmel ATmega Datasheet.” Microchip [Online]. Available: https://ww1.microchip.com/downloads/en/devicedoc/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf

[11] “Robotics market revenue worldwide 2018-2025,” Statista. [Online]. Available: https://www.statista.com/statistics/760190/worldwide-robotics-market-revenue/. [Accessed: 11-May-2020]

[12] “Pololu - DRV8825 Stepper Motor Driver Carrier, High Current.” [Online]. Available: https://www.pololu.com/product/2133. [Accessed: 11-May-2020]

Appendix

Fully Open-Source Github Repository:

Project Page: https://github.com/Open-AMR

Hardware Design: https://github.com/Open-AMR/Open-AMR-Hardware

Software Design: https://github.com/Open-AMR/Open-AMR-Software

Documentation: https://github.com/Open-AMR/Documentation

PiOled Installation: https://github.com/Open-AMR/installPiOLED

Setup Jetson Nano:

Install Ubuntu:

https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit#write

The Nvidia Official Image should already contain a variety of packages such as Python.

Download Code:

1. Open terminal window
2. Install git if not already installed, ‘sudo apt install git’
3. Use command ‘git clone https://github.com/Open-AMR/Open-AMR-Software.git’ to download the software repository in the current directory
4. Use command ‘git clone https://github.com/Open-AMR/installPiOLED.git’ to download the PoOled repository in the current directory
5. Follow instructions listed to install PiOled to run on startup

Run Remote Control:

1. From software repository directory, ‘cd joystick_code’
2. Run ‘python3 Remote_Control.py’
3. May need to install python serial and evdev libraries, also may need to run as sudo depending on system permissions (sometimes serial has to be run under sudo).

Setup Arduino Driver Board:

Download/Install Arduino IDE:

https://www.jetsonhacks.com/2019/10/04/install-arduino-ide-on-jetson-dev-kit/

Run Remote Control:

1. From software repository directory, ‘cd TMC2130_MCU/Arduino_Code_Working/Serial_Accel_PWM’
2. Open Serial_Accel_PWM.ino with the Arduino IDE
3. Select MEGA 2560 as the architecture from the setting list, select the correct serial port for uploading
4. Compile and upload program

Run Hardware Monitoring:

5. From software repository directory, ‘cd TMC2130_MCU/DriverBoardCode/curr_volt_monitoring’
6. Open curr_volt_monitoring.ino with the Arduino IDE
7. Select MEGA 2560 as the architecture from the setting list, select the correct serial port for uploading
8. Compile and upload program
9. Open Arduino IDE serial console, set baud rate to 115200 to view data

Access/Change Mechanical CAD Design:

Download/Install Autodesk Inventor, 2020 or newer

Access/Change Electrical CAD Design:

Download/Install Altium Designer, 17 or newer

Bill of Materials, Robot Chassis Mechanical:

Supplier	Item Name	Each Price	Quantity	Tot Price	Link/PN
AndyMark	4 in. Standard Mecanum, Wheel Set	77	1	77	Link
AndyMark	1/2 in. Hex Steel Shaft, 5in	6	8	48	Link
AndyMark	1/2 in. Hex Hub	11	6	66	Link
AndyMark	HiGrip Wheels, 4in, 50A Durometer	8	4	32	Link
	Total			223	
Mcmaster Carr	Stainless Steel Ball Bearing Flanged, Shielded, NO. 688-2Z, for 8 mm Shaft Diameter	8.54	10	85.4	7804K147

Mcmaster Carr	Stainless Steel Ball Bearing Flanged, Shielded, NO. 105-2Z, for 5 mm Shaft Diameter	8.23	6	49.38	7804K136
Mcmaster Carr	Needle-Roller Bearing Open, for 4 mm Shaft Diameter	7.47	6	44.82	5905K348
Mcmaster Carr	Alloy Steel Flanged Button Head Screws Zinc-Plated, M6 x 1 mm Thread, 12 mm Long	11.07	1	11.07	96660A215
Mcmaster Carr	Alloy Steel Thread-Locking Cup-Point Set Screw Black-Oxide, M5 x 0.8 mm Thread, 5 mm Long	13.94	1	13.94	91385A211
Mcmaster Carr	Alloy Steel Flanged Button Head Screws Black-Oxide, M3 x 0.50 mm Thread, 20mm Long	7.1	2	14.2	92137A262
Mcmaster Carr	Button Head Hex Drive Screw Passivated 18-8 Stainless Steel, M5 x 0.80 mm Thread, 40mm Long	11.02	1	11.02	92095A222
Mcmaster Carr	Medium-Strength Steel Nylon-Insert Flange Locknut Class 8, Zinc-Plated, M5 x 0.8 mm Thread	7.16	1	7.16	92461A200
	Total			236.99	
Digikey	FAN AXIAL 50X10MM 12VDC WIRE	4.69	4	18.76	Link
Digikey	FAN GUARD 50MM METAL	0.78	4	3.12	Link
	Total			21.88	

Supplier	**Item Name**	**Each Price**	**Quantity**	**Tot Price**	**Link/PN**
Amazon	BLACK+DECKER BDL220S Laser Level	14.79	1	14.79	link
Amazon	NVIDIA Jetson Nano Developer Kit	98.95	1	98.95	link
Amazon	SanDisk Ultra 64GB	11.49	2	22.98	link
Amazon	Logitech C270 USB Webcam	19.5	2	39	link
			Total	175.72	

Supplier	Item Name	Each Price	Quantity	Tot Price	Link/PN
Amazon	Logitech Gamepad F710	40.27	1	40.27	link

Supplier	Item Name	Each Price	Quantity	Tot Price	PN	Link
Digikey	SENSOR CURRENT HALL 50A AC/DC	6.3	5	31.5	620-1541-5-ND	Link
Digikey	IC HOT SWAP CTRLR GP 8SOPWR	2.04	10	20.4	296-24239-1-ND	Link

Driver Board PCB Schematics and Bill of Materials (Next 2 Pages)