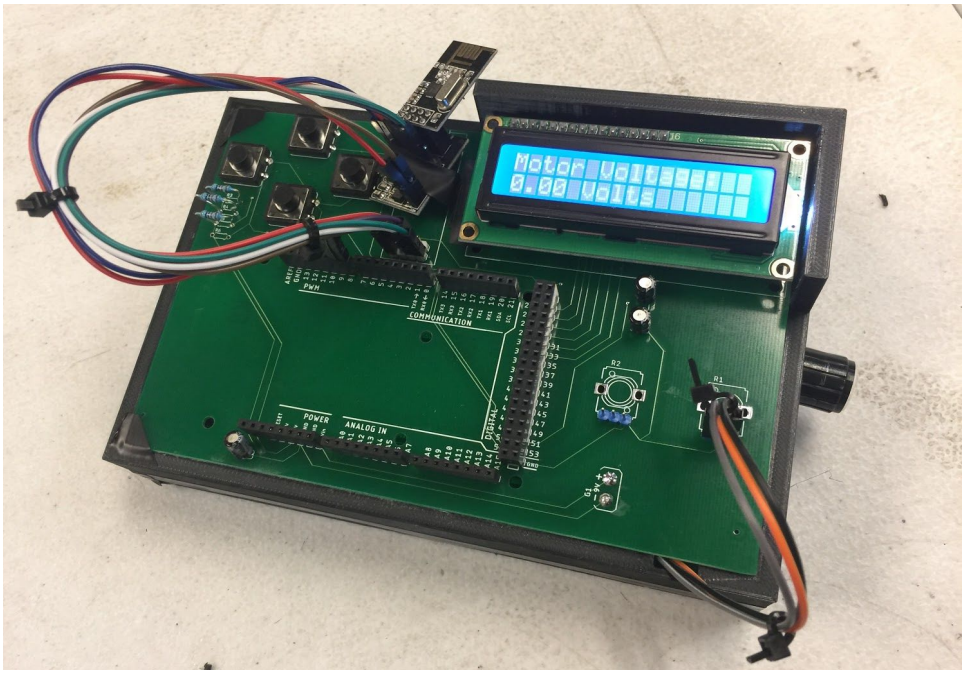
SPARTAN SUPERWAY

Spartan Superway: A Solar Powered Automated Transportation Network

ME 195B 2018-2019

Full-Scale Controls Team



May 19th, 2019

Presented By:
Alan Philip Kalarickal (Team Lead)
Alex Krause
Justin Guro

Abstract

For the 2018-2019 academic school year, the full-scale controls team built wiring schematics to operate the specified brushless DC motors and designed a PCB controller to operate the motors wirelessly to allow for better quality control testing. We also built upon Arduino programs from previous years to fix manual and automatic issues stated in previous years' reports. Lastly, we effectively implemented wireless SPI connection into the project to improve feasibility and laid down building blocks for future controls system teams.

This was all done through looking through datasheets and understanding the setup for each individual part in perspective to the project as a whole. We built the wiring diagrams to wire up the motor and their motor controllers. After successfully testing the motor setup, we built upon it by incorporating SPI communication and having the possibility of creating a wireless system where the user can be a good distance away from the bogie and maintain control. This was done through learning the basics of wiring schematics and PCB design in Autodesk Eagle. Once the PCB was designed and met manufacturer specifications, we soldered all the hardware components onto the PCB that were required for wireless communication and programmed the controller and the bogie to communicate with one another.

The outcomes were satisfactory in relation to our goals and objectives for the semester. The controls system was able to output at least a speed of 2 miles per hour to each motor, and increased safety by using wireless communication to incorporate a wireless PCB controller. Programming for autonomous driving was finished, but due to lack of time and lack of a reliable test bed, we could not debug our program to see if it failed at any of our conditions or if our functions even worked. Recommendations for the future are to obtain enough knowledge about the software and hardware to be successful as a controls team, program a successful position tracking system, and improve automation features. We also recommend to make sure to communicate with other teams effectively and regularly, as the controls team is very reliant on track development, bogie improvement, power sustainability, and motor efficiency. We hope that future controls teams can learn from our mistakes this year and continually improve upon the controls section of the project to improve the overall functionality of Spartan Superway.

Acknowledgments

Over the course of the 2018-2019 academic year, many people came to us to offer help along our busy year of creating our controls system from scratch. We would first like to thank Dr. Furman for his continued feedback into research, hardware, and software development, as well as answering all our mechatronics based questions. We would also like to thank Daniel Ornellas for providing us feedback on his time working on earlier versions of the control system back in 2015 and acting as our coach throughout the academic year. Without Daniel for guidance much of the year, we doubt we would be at the point we are at today. We would like to thank the motor and wayside power team members for working with us consistently throughout the academic year and being in constant contact with us when something went wrong. Finally, we would like to thank a fellow mechanical engineering student Mirza Baig for his introduction of Eagle software to us and showing us how the software works and telling us about the in and outs of schematic and PCB design.

Unfortunately, the controls team was not able to procure any sponsors this semester, but do be on the lookout for hardware and software companies that you work through that may be interested. Sponsors help the project financially and the project would not be where it is today without the generous donations from investors and continuous or newly acquired sponsors each year. We may not have been able to obtain sponsors this year, but we hope future teams will be able to gain traction and hopefully gain a sponsor that may help the controls team into an even better future at Spartan Superway.

Table of Contents

Table of Figures

Executive Summary

Introduction and Objectives

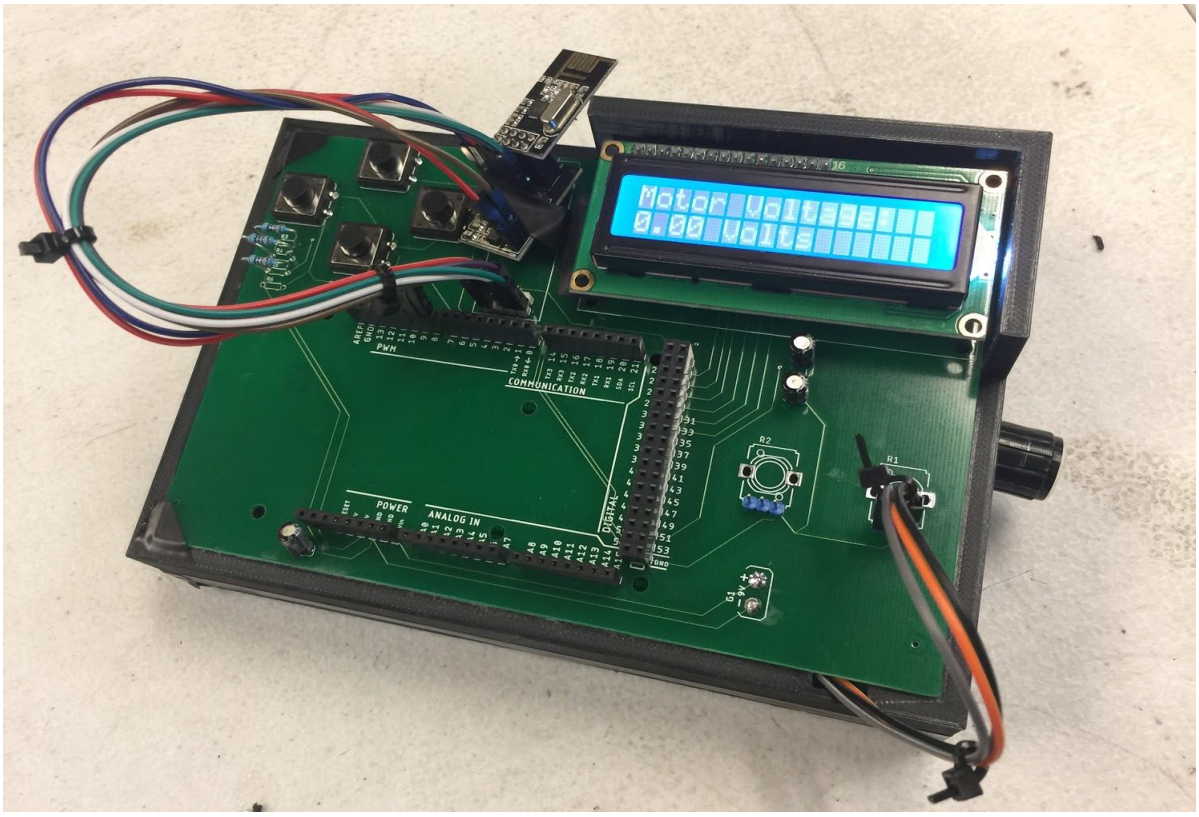
As the controls team, we were in charge of bringing the whole project to life. We were tasked with creating a controls system that would control all the electrical components of the Superway bogie system. Our goal was to integrate all of the other full-scale subteams parts and be able to drive the integrated bogie. The motors are controller through various hardware and motor controllers supplied by the motor team. In case of an emergency, the emergency brake developed by the braking subteam would be deployed to avoid danger. Power for the controls systems would provided by the wayside power team in the form of supercapacitors. Our objectives for the year were to improve the overall safety of the system, bogie position tracking, path switching mechanism, wireless communication between the bogie and a controller, and be able to control the bogie in both manual or automatic modes. Spartan Superway is an interdisciplinary project, so as the full-scale controls team, we had to learn various techniques and tools to help integrate everyone's parts together and control all of them efficiently.

Procedure and Results

As a team, the work was divided evenly in order to effectively reach all of our design requirements and goals by the end of the academic year. Justin Guro was in charge of wiring schematics, hardware evaluation, quality control of motors and PCB design. Alex Krause and Alan Kalarickal were in charge of testing various sensors, linear actuators, and the main motor driver code on Arduino IDE. The motor team supplied the controls team with the motors and their controllers along with datasheets. Following the datasheets allowed us to wire everything correctly and test the motors. For modular purposes, all the wires were attached onto molex connectors using a crimping tool. This allowed for quick switching of any faulty wires and

allowed for efficient debugging of the system. For the wireless controller, we researched SPI communication protocol and nRF24L01+ modules to understand how wireless communication works. We had to troubleshoot how to properly use and wire the wireless transceiver modules, but quickly overcame this issue and we were able to proceed with wireless communication. For the controller, the PCB was manufactured by JLC PCB and was fitted into a 3D printed casing made using the 3D printer available at Superway, as seen in Figure 1.

Figure 1. Finalized wireless controller with 3D casing



While the controller and motor driver programs of the system were being worked on, the motor was being setup and tested. The motor team provided our team with the datasheets necessary to complete this task, and completed the wiring schematic for the motors seen in Figure 2.

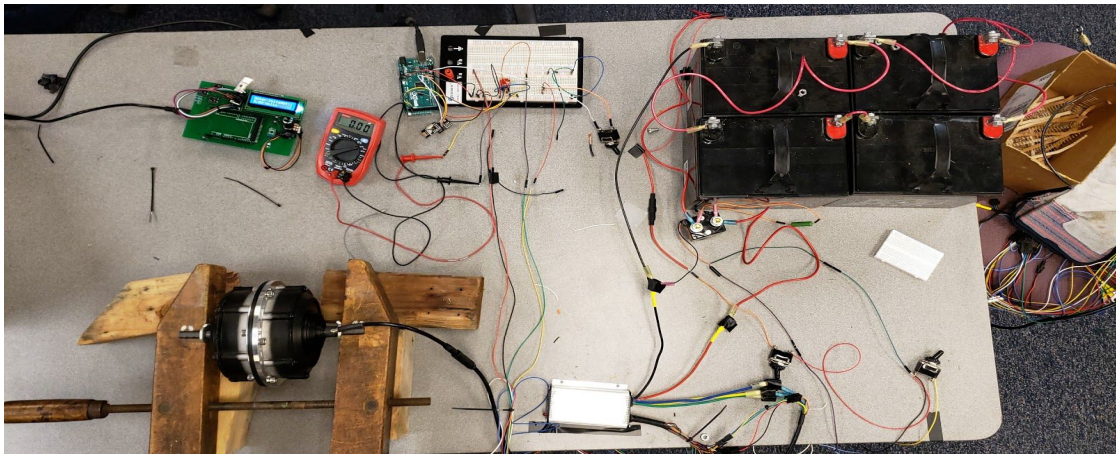


Figure 2. Motor and motor controller setup

Conclusion and Recommendations

As we finalized the work done on the control system for this academic year, we were able to accomplish most of our objectives and believe to have given a good baseline for future controls team. Wireless communication and the motors were fully operational, researched hall effect sensors for position tracking, and introduced better safety features in case of emergency. Modularity allowed for quick and efficient debugging and fixes if anything broke. The automatic and manual modes can be accessed using a combination of button presses, for more complex routing. Recommendations for future teams are to work on implementing a rotary encoder to improve position tracking capabilities and work with the bogie integration team to get a bogie switching mechanism implemented. Research everything that we have worked on and replicate what we have done to ensure understanding of the whole system has been achieved. When writing the drive bogie code, break up the code into functions for better access and troubleshooting. PCB design can be implemented further by iterating our board to house the proper sized components as well as future designs to reduce wiring in the system. General PCBs can also be used if time is an issue and components can be quickly soldered on. Finally, work closely with all the subteams and keep a track of the gantt chart. Ensure to list all parts in the bill of materials and consistently ask questions to coaches and advisors to ensure a nothing goes wrong.

CHAPTER 1

Introduction and Project Description

Introduction

Current Problems with Transportation

The health of our planet is exponentially deteriorating, and will continue to to get worse unless significant changes are successfully implemented soon. Most people in the United States push the blame onto the industry sector, and believe that they are the major cause for the disastrous environmental shift. Instead, society needs to point their fingers at the transportation industry, since according to the Environmental Protection Agency (2018), the sector that contributes the most greenhouse gas emissions in the United States is the transportation sector at 29% of total emissions. Greenhouse gas emissions from the transportation sector primarily come from the burning of fossil fuels to power the most common methods of transportation, such as cars and planes. While changes in the transportation industry have helped slow down its dependence on fossil fuels, there still has not been a significant push towards changing the way that humans travel from one place to another.

One of the best places that exemplifies the current problems in mass transportation is the Bay Area. With its ever growing tech industry and promise for opportunity, people from all over the country are flooding to the area, and it is exposing the Bay Area's weak public mass transportation systems, such as BART and Light Rail. The Bay Area also already has to deal with massive traffic congestion and commuting problems, as seen in Figure 3. According to a report by INRIX Global Congestion Ranking, the Bay Area had "the 5th worst traffic congestion in the world, and the 3rd worst congestion in the United States in 2017" ("San Francisco Bay Area 5th Worst Traffic Congestion In The World, Study Finds, 2018). It is obvious that there are

serious problems in current transportation systems in dense urban areas, yet there are still no presentable solutions.

Figure 3. Traffic congestion trying to get into San Francisco. Retrieved from https://www.janekim.org/2018/03/12/break-the-gridlock-better-transit-and-transportation-for-everyone/

The main goal of the transportation sector needs to be getting as many cars and trucks off the road as possible. The problem is with a continuous growth in population comes a significant influx of vehicles in dense urban areas. The most effective way for urban areas to solve the problems that come from an increase in vehicles is creating an efficient mass transportation system, but at their current state, these systems are unreliable and are plagued with problems. Crowdedness at peak hours leaves riders uncomfortable and dissuaded from continuing to use mass transportation methods, and most systems are unable to make a profit due to high maintenance costs (Rodrigue, 2018). With no solution in place, major complications arise very quickly. Traffic congestion is one of the most prevalent problems, and according to Dr. Jean-Paul Rodrigue (2018), in the 21st century, “drivers would spend about 3 times more time in congestion as they did in the later part of the 20th century.” Residents in dense urban areas are also subjected to increased pollution and noise, and are at a higher risk of getting into an accident

with drivers (National Express Transit, 2017). Drivers are even at an increased risk of getting into an accident, considering about "94% of transportation fatalities occur on highways and mostly involve passenger vehicle crashes (Transportation Research Board of the National Academies, 2013). Mass transportation systems try to solve these problems, but no system has been able to show that it is a reliable solution.

Automated Transit Networks (ATN)

Instead of trying to solve the problems with current mass transportation methods, an entirely new method of transportation needs to be implemented. This would prevent having to use old and outdated infrastructure to create a temporary fix to current transportation problems. It would also allow cities to invest money into a permanent fix instead of investing in a broken system. The most promising new transportation idea is the Automated Transit Network (ATN). Automated Transit Networks use an automated system to transport people from station to station in a quick and efficient manner (Ellis, Fabian, Furman, Muller, & Swenson, 2014, p. 25). The idea of building Automated Transit Networks has only recently gained popularity, but the concept is promising and is a very attainable solution to mass transportation problems.

In order for a transportation system to be classified as an ATN, it needs to meet certain standards and contain certain features. Dr. Burford Furman, Lawrence Fabian, Sam Ellis, Peter Muller, and Ron Swenson wrote a report on Automated Transit Networks and laid out the groundwork for what determines an ATN. According to the report, Automated Transit Networks need to feature direct origin-to-origin service, small vehicles available for the exclusive use of an individual or small group, service available on demand, fully automated vehicles that are available for use 24 hours a day, 7 days a week, vehicles captive to a guideway that is reserved for their exclusive use, small guideways that are usually elevated or can be underground, and vehicles that are able to use all guideways and stations on a fully connected network (Ellis, Fabian, Furman, Muller, & Swenson, 2014, p. 1). Having the vehicles be able to go from one destination to another with no stops and being able to be used by a single person or small group is very similar to the services that Uber and Lyft provide to the public. These rideshare services do help take cars off the road, but the fact that Automated Transit Networks are elevated and

available 24/7 helps tackle the major problems of traffic congestion and commute times. This, plus the fact that these systems can be powered solely on solar energy equals a transportation solution that is realistic, scalable, and promising.

Spartan Superway Project

The Spartan Superway project is a serious attempt at creating an example of what an Automated Transit Network can be. The main goal of the project is to create a transportation system that meets all the criteria of an Automated Transit Network and shows how an ATN can solve the current problems in the transportation sector, such as pollution, safety, and congestion. Spartan Superway takes the form of a podcar system that hangs from an elevated guideway above the streets, as seen in Figure 4. The guideway follows the natural flow of traffic, can be implemented above any street or freeway, and is completely solar powered. The podcars can be called to a certain station, pick people up, and drop them off at their designated stop. This personal rapid transport system can greatly reduce the amount of cars on the road, decreasing traffic congestion, commuting time, and can reduce greenhouse gas emissions since it runs on clean energy.

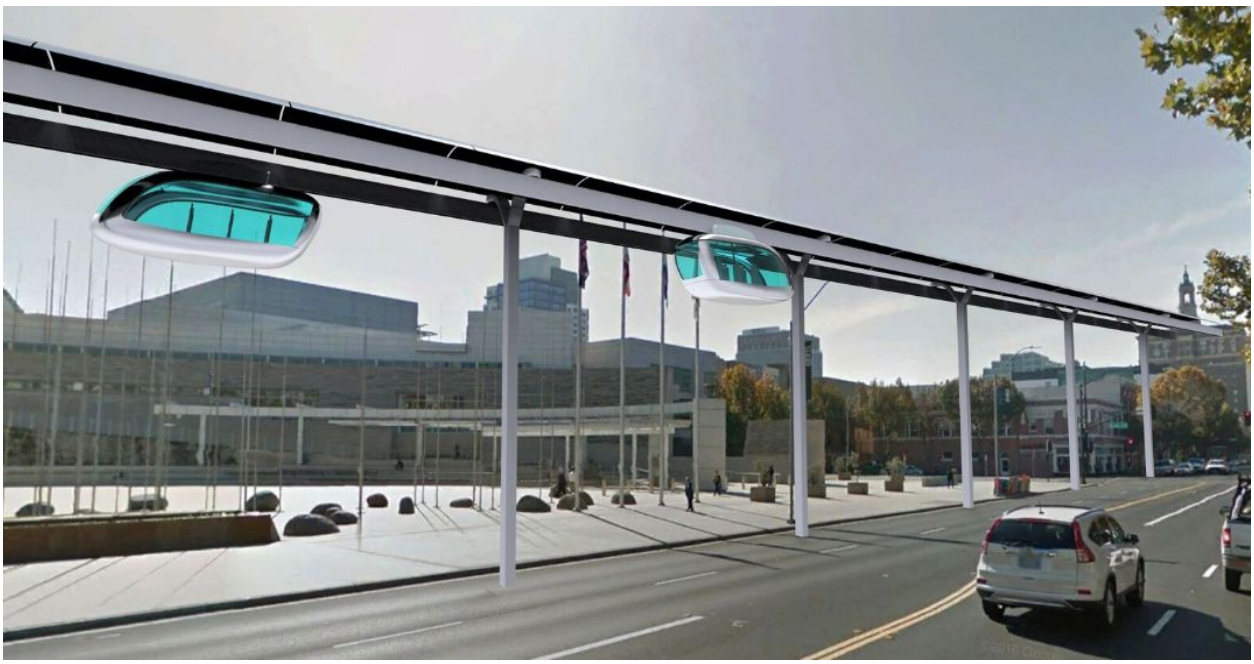


Figure 4. Conceptual model of Spartan Superway. Retrieved from http://www.solarskyways.net/2017/04/summer-2017-research-experience-for.html

Societal Impact of Spartan Superway

The Spartan Superway team has an ambitious goal: change the way humans travel from one point to another. While the motives behind the project have great merit, reaching that goal could prove difficult. The transition period where old mass transportation infrastructure would be replaced with the Superway system would have a negative impact on society. Whenever commuting routes are altered, it creates a great inconvenience to commuters and could turn society away from adopting the system.

While setting up the system would take years to complete and would need the patience of the community, once complete, Superway would yield beneficial results to all who were affected by the transition. Providing society with a simple, fast, and clean new method of transportation would bring about great praise and improve the quality of life of residents. Less cars on the road would lead to less traffic congestion and shorter commute times, but more importantly, society would have more space on the ground to use. With Spartan Superway being an elevated system using pods, cities could reduce the amount of parking garages and parking spots, freeing up space for parks, walkways, or businesses. The rest of the world could also greatly benefit from a system like Spartan Superway being implemented. A successful showing of how ATNs can improve transportation would hopefully convince other parts of the world to invest in new, clean methods of transportation to reduce its dangerous impact on our environment.

Not only will Spartan Superway improve transportation methods, but it would also increase accessibility to transportation. Most current mass methods of transportation only reserve small spaces to the disabled. Spartan Superway pods and stations would be designed to accommodate all the needs of the disabled, especially people in wheelchairs, and provide the disabled enough space since the pods are designed to only carry a couple of people. The controls system for Superway would also be designed in a way to make it very easy to use for the public, allowing the elderly, who would have to adjust greatly to a new method of transportation, to easily and safely use the system. Lastly, the Superway system would be run autonomously and be solar-powered, saving the city on operating and energy costs and would make it very cheap for the public to use the system.

Spartan Superway History of Work at SJSU

The Spartan Superway project has been in the workings for the past 7 years, and with each year comes innovations and new features that bring us closer to a true Automated Transit Network. The 2012-2013 school year was the first year that the Spartan Superway project was started, and the team that year created an amazing 1/12th-scale model that provided a basis for how future half-scale and full-scale models should be designed. The 2013-2014 team was able to successfully build a full-scale bogie and pod car that could be used to demonstrate to the public and future students what the project was all about. Significant progress then followed during the 2014-2015 year where a full-scale track with two pathways was built to mount the full-scale bogie and podcar to. A half-scale model was then introduced during the 2015-2016 school year, as well as improvements towards solar power harvesting and a suspension system on the full-scale model. The biggest thing that happened during the 2016-2017 school year was that the 1/12-scale, half-scale, and full-scale models all included more features that made them closer to true Automated Transit Networks. Finally, last years group made significant improvements to the solar teams mounting brackets, the full-scale model's bogie design, and the integration of wireless controls to the full-scale model.

What to Expect in This Report

The main objectives of this report are to explain the current status of the full-scale controls team's design and to setup future controls teams for success by laying the groundwork that can be improved upon. The report will first go into more detail about what Automated Transit Networks are and provide examples of successful implementations of Automated Transit Networks around the world. Next, we will go over our final design for our controls system, how we controlled the motor, and go over our programs for our wireless controller and our bogie. Lastly, we will go over the budget our team had this year, summarize our work, and provide recommendations for next years controls team.

Background of Full-Scale Controls Team

Context of Work

The Superway Project has depended not only on its strong design work in previous years, but has been built upon solid electrical and control systems knowledge to ensure the automation and movement of the bogie. This pivotal step allows teams to see their designs in action and allow for design feedback if changes need to be made. As the full-scale controls team, we must integrate all electrical components and ensure safe and secure electrical connections and motor control. The problem the controls team faces is to ensure that all electronics on the bogie work for their intended design. The ability to integrate all components and ensure no faults occur, such as failure of speed control, position tracking, current and voltage regulation, and manual and autonomous movement, will allow the project to continue further into its design improvements and possible integration into a small sub area of the local community for real life testing.

Full-Scale Controls Team Objectives

When the project began, many problems were to be accounted for. Given the state of the project from the previous team, we decided to focus on three major objectives that would affect the success of the project as a whole this year. The three objectives were:

- The control and wiring of the brushless DC motors(BLDC)
- Position Tracking using encoders and hall effect sensors
- Manual and Automation of the bogie

By focusing and prioritizing these objectives the team will achieve a baseline for controlling the bogie system and ensuring main design requirements and specifications are met before diving deeper into smaller objectives such as optimizing feedback, calculating more efficient setups, and diving into deeper research on complex control systems as to portray a real life integration.

Design Requirements and Specifications

The Spartan Superway system design requirements for the 2018-2019 Full-Scale Controls team are based upon plausible specifications for what is it be produced by the end of the semester. These design requirements are not to portray a real life scenario and have been based upon the mock-model that will be produced at the end of the academic year. In the Appendix will

be listed real life design requirements and specifications that can be used to help with deciding the design requirements and specifications for future teams. For the following 2018-2019 academic year, the controls team stated the following design requirements and specifications that are to be met by the end of the year. The bogie control system shall do all of the following:

- The bogie shall be able to reach a speed of 2 miles per hour and cruise until a turn or stop is approaching
- The bogie shall keep track of its position to ensure it does not make a wrong turn or fall of the track. This also ensures the bogie will not run past a station and will be used as markers to break into or accelerate out of an oncoming station.
- The bogie can be manually operated to allow for debugging, design verification, and quality control.
- The bogie can be autonomously operated under normal load conditions
- The third rail actuators will be control so they may switch positions within a reasonable time to ensure the bogie does not make a wrong turn
- Ensure all components from Wayside Power, Motor team, and Brake team are integrated and can be controlled manually and autonomously such as voltage and current control, speed control, and emergency brake integration.

These design specifications and requirements uphold the quality and safety of the project and ensure that safety measures as well as efficiency have been accounted for in the final design of our control system.

CHAPTER 2

Literature Review and Current Studies

ATN Technology

Automated Transit Networks (ATN) are a unique form of transportation which have special features that traditional forms of transportation lack. The first concept came around the 1950's, and since then there are five main installments around that world of Automated Transit Networks that shows promise. These five are located in Morgantown in West Virginia, Heathrow Airport in London, Masdar City in Abu Dhabi, Rotterdam in the Netherlands, and Suncheon Bay in South Korea. According to a study conducted by Mineta Transportation Institute, there are seven unique features ATNs possess, and they are:

1. "Direct origin-to-destination service with no need to transfer or stop at intermediate stations
2. Small vehicles available for the exclusive use of an individual or small group traveling together by choice
3. Service available on demand by the user rather than on fixed schedules
4. Fully automated vehicles (no human drivers) that can be available for use 24 hours a day, 7 days a week
5. Vehicles captive to a guideway that is reserved for their exclusive use
6. Small guideways (narrow and light relative to light rail transit or LRT and bus rapid transit or BRT) that are usually elevated but that also can be at or near ground level or underground
7. Vehicles able to use all guideways and stations on a fully connected network" (Ellis, Fabian, Furman, Muller, & Swenson, 2014, p. 1)

With these features, many of the problems faced by traditional forms of transportation can be reduced or even eliminated. The main problems include environmental pollution, congestion, accidents, parking, safety, and wasted energy. With the use of solar power, we are relying on clean and renewable forms of energy instead of relying on fossil fuels. According to the United States Environmental Protection Agency, "In the United States, greenhouse gas emissions caused by human activities increased by 7 percent from 1990 to 2014" (EPA, 2017). With the help of a solar powered ATN, which Spartan Superway is focused on, the transportation sector can start reducing their carbon footprint. By building the guideways above land, it allows there to be more space on the ground and can allow cities to more efficiently use their space. These guideways will stretch across cities and towns making it widely accessible to many people. With this setup, people can use the land beneath the guideway as places for people to interact like markets, parks or even shopping centers. If cities switch to ATNs, the amount of vehicles on the streets will be heavily reduced and thereby reduce traffic, parking problems and accidents. With the automated features of an ATN, riders can request a podcar in a matter of minutes and takes people to their destinations without any stops or transfers. This design aims for a direct origin-to-destination service, which can accommodate up to 5 passengers. The control system for ATNs will be optimized for the times when it is traveling and when it is at rest, further reducing the amount of wasted energy.

As discussed earlier, even though ATN's are new and promising form of transportation, there are only five installments around the world which fully embody all the features it has to offer. They are Morgantown PRT, ParkShuttle, Masdar PRT, Heathrow T5, and Skycube. The Morgantown PRT (1975), as seen in Figure 5, connects all three campus of West Virginia University along with the downtown area within a eight mile track. Each vehicle used for travel can accommodate up to 20 people with eight seats included. According to the West Virginia University, with a speed of 30 mph, these vehicles carry "15,000 people ride the PRT during the school year every day" (West Virginia University). While operation, there are three modes that the vehicles run on: demand, schedule, and circulation. 'Demand' mode is used during non-peak hours of the day, where it takes individuals to their desired location. While on peak-demand

situation, it switches from ‘demand’ to either ‘schedule’ or ‘circulating’ mode to accommodate the demands for various people traveling in one pod.

Figure 5. West Virginia University ATN. Retrieved from http://www.solarskyways.net/2017/04/summer-2017-research-experience-for.html

The ParkShuttle in Netherlands is ATN system which was started in 1999 which connects business park Rivium and the residential area Fascinatio as well as business park Brainpark III with the metro and bus station Kralingse Zoom. “1.8km of (dual-lane) guideway (at grade) with 6 vehicles connecting 5 stations” (ATRA). There are no emissions associated with this transport because all the energy for this system is stored in lead acid batteries which has a range of 75+ km. It is similar to a driverless bus and guided on the track with the help of magnets on the road. The vehicles can hold up to 20 passengers with 480 passenger per hour per direction.

The Masdar PRT is a prime example for ATN’s which is utilized since 2010 in the carbon neutral, car free city of Masdar City, Abu Dhabi (2getthere, Masdar). This system operate

for 18 hours a day between two stations on track which is 1.4 km long. There are 10 on-demand vehicles available for passengers to use with rechargeable lithium ion batteries and magnetic inlaid guideway strips to keep it on track. Even though this is an ATN, due to its single route system and function, this operates more like a shuttle service.

The ULTra (Urban Light Transit) PRT system in Terminal 5 of the Heathrow airport in London runs between terminals and business passenger car park 2.4 miles away (ARUP). This system includes 18 podes that runs over the largest ascended guideway. This system uses rubber tires to run on an open guideway network. This give the bogie a reduction in weight, quick to assemble it together and a low cost of maintenance.

Skycube is one the last ATN to exist today. Skycube runs in the Nature Shuttle Park in Suncheon Bay of South Korea. This PRT connects the Suncheon "Dream Bridge" with the Suncheon Bay Ecological Park. According to Advanced Transit Association, " 4.6km of elevated, bi-directional guideway, 40 vehicles with seating capacity for 6 to 9 passengers per car plus accompanying luggage/baby buggies etc. The system is fully compliant to international Rail Vehicle Accessibility Regulations (RVAR) with level access and wide, bi-parting doors allowing ease of access for all passengers including the mobility impaired" (ATRA). This single lane guided system operates like a shuttle service.

SPI Communication Protocol

Serial Peripheral Interface (SPI) communication is widely used for microcontrollers and peripheral ICs send and receive data between two systems over short distances. For SPI, there would be one master controller and any number of slave modules. There are 4 main lines that are common to all modules and they are:

1. SCK: Serial Clock; For synchronized data transmission, clock pulses are generated by the master.
2. MISO: Master In Slave Out; the master reads data from the slave.
3. MOSI: Master Out Slave In; the slave reads from the master.
4. SS: Slave Select; this pin is used to enable or disable specific slaves to send or receive data.

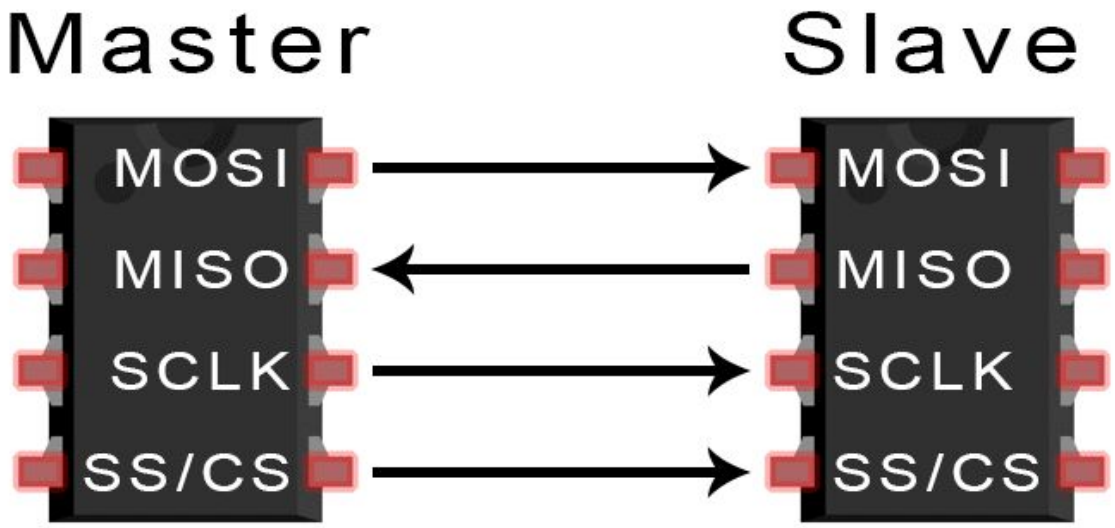


Figure 6. SPI configuration between master and slave

According to the Circuits Basics, "One unique benefit of SPI is the fact that data can be transferred without interruption. Any number of bits can be sent or received in a continuous stream" (Circuit Basic, 2018). SPI is a bi-directional communication protocol, so we can send data from the master to the slave and vice versa. Some advantages the circuit basics website talks about are:

1. "No start and stop bits, so the data can be streamed continuously without interruption
2. No complicated slave addressing system like I2C
3. Higher data transfer rate than I2C (almost twice as fast)
4. Separate MISO and MOSI lines, so data can be sent and received at the same time" (Circuit Basic, 2018)

With these features, SPI was the best candidate for wireless communication. This research allowed us to follow through with programming the wireless communication parts of our major programs using this communication protocol.

CHAPTER 3

Final Design Solution

Controls Process

Before we could start constructing our system, we first needed to figure out what process we needed to go through when presenting our system. Our system process path is represented in the flowchart seen in Figure 7.

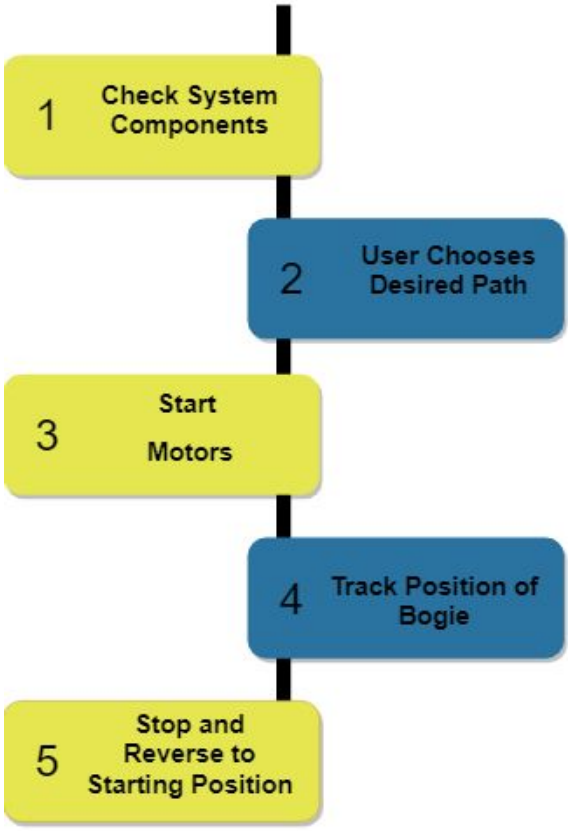


Figure 7. Controls system process final design flowchart

As a safety precaution, we first want to check and make sure all the electrical components in the system are working. First, we will check if the linear actuators, as seen in Figure 8, work to ensure that the path selection process works. Second, we will check if the hall effect sensors, as seen in Figure 9, work to ensure that the position tracking system works. Third, we will make sure that the motor and the motor controller are working and talking to each other in the appropriate manner.

Figure 8. 524N Linear Actuator

Figure 9. Hall Effect sensor

Then, we will have the user choose the path they want to travel, either turning left or going straight. Next, we will start the motors in either manual or automatic mode. In manual mode, the user can control the speed of the bogie using a potentiometer. In automatic mode, position tracking will determine the speed of the bogie. In order to track the position of the bogie, we will be placing magnets on the track in the three spots seen in Figure 10, and then a hall effect sensor would pass the magnets as it travels and send a signal to our controller alerting the user where the bogie is. Once the bogie is at the end of the track, we will flip the brake and reverse switches to stop the bogie and have it return back to its starting location.

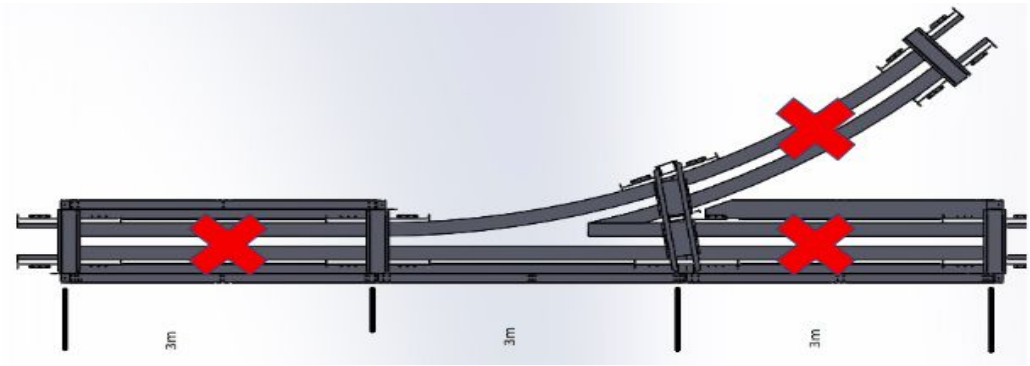


Figure 10. Hall Effect Magnet Placement on Track

Hub Motor Setup and Control

To begin controlling the motor, the Brushless DC motors (BLDC) had to be wired up per manufacturer specification to BLDC motor controllers. The motors, motor controllers and all essential adapters and were provided by the full-scale motor team. The schematic for the wiring setup is displayed in Figure 11. The schematic can also be found in the controller manual in the repository along with any components found in the schematic.

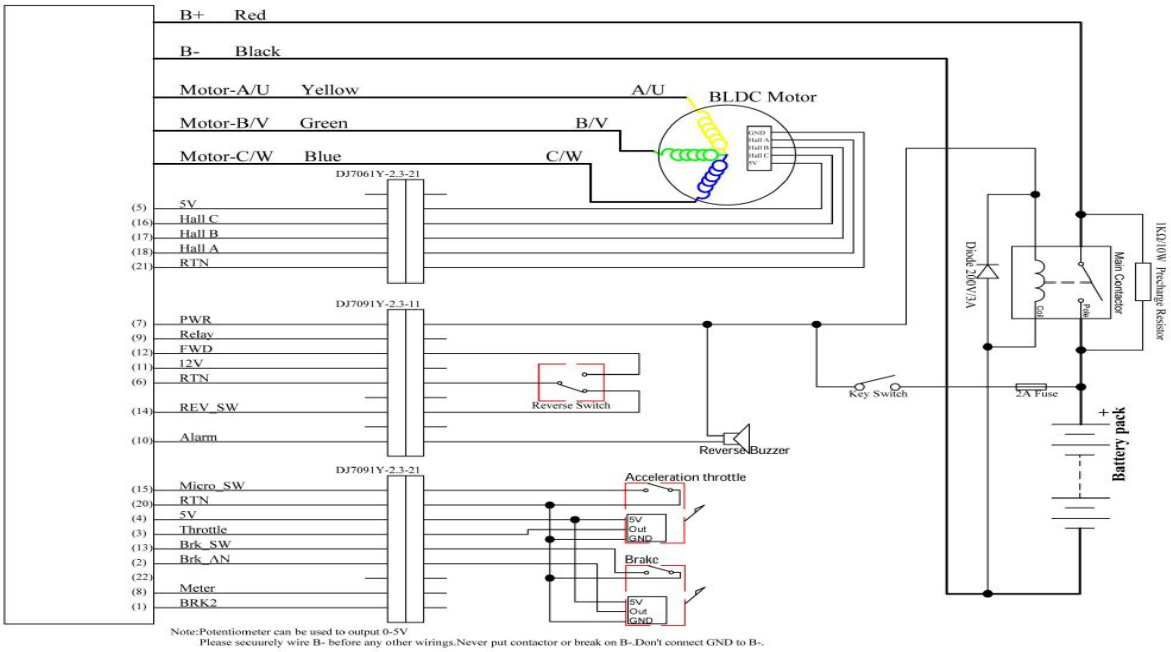


Figure 11. Motor Controller Wiring Schematic (Larger image can be found in Appendix C)

When wiring the motors and motor controllers, various terminal connectors for small gauge wires were needed to connect high current and voltage connections. Other larger gauge wires were crimped using a crimping tool to install molex connector pins. To avoid burning wires and short circuits, the small gauge wire was used for heavier currents and voltage. Once wiring has been completed and connections have been secured, the circuit is turned on with a 48 volt power supply. With the circuit powered on, we are able to set the controller pins to their proper settings. From the wiring diagram there are two analog signal receivers. These correspond to the brake and throttle analog outputs. These analog outputs can be controlled by a simple

variable potentiometer. With the potentiometers wired up, we can now vary the speed of the motors. By increasing the value of the potentiometer, the throttle begins to increase on a range of 0 to 5 volts. From reading the potentiometer, we noticed the motor will not begin to start until the throttle has received at least a 1 volt analog signal. Afterwords, as the signal is increased the motor continues to speed up until it a full 5 volts is read from the potentiometer. The same goes for the brake analog signal but this is not very reliable and does not work most of the time during our test runs. The next step was setting up all the switches. The switches included in the motor controller schematic control the brake, throttle acceleration, and reverse polarity. These were controlled using single pole single throw switches (SPST). The brake switch allows us to instantly stop the motors if needed, immediately cutting the signal to the motors and stopping rotation. If the brake switch is switched back on accidentally, the motor controller does not allow the motor to restart until the throttle pins reads 0 voltage as an added safety feature. This assures if there is any remaining voltage in the system when the brake is turned back on, that the motors will remain off.

Wireless Communication

The most difficult part of this year was figuring out how to have two Arduinos talk to each other wirelessly. The main goal of this section to to explain how the wireless communication works and how to avoid common problems that we ran into. The wireless communication hardware that we worked with were NRF24LO1+ transceiver modules. However, there are two different types of these chips. Figure 12 shows the adapter module and Figure 13 shows just a module.

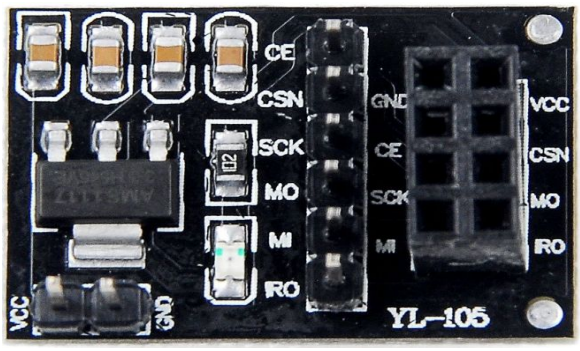


Figure 12. NRF24LO1+ Adapter Module

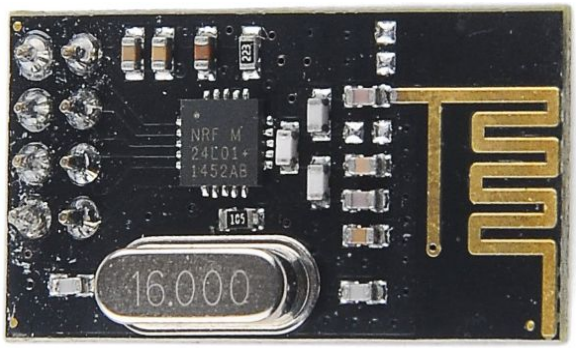


Figure 13. NRF24LO1+ Module

The best way to learn about how these modules work, we would recommend following the instructions at https://howtomechatronics.com/tutorials/arduino/arduino-wireless-communication-nrf24lo1-tutorial/. It explains how to wire the transceiver modules to your Arduino and how to write a program to learn how they work. The tutorial uses the module seen in Figure 13, and tells the reader to plug the VCC port into the 3.3V. When we tried to do this and run the sample program, it would not work. The serial monitor in Arduino IDE would print weird symbols and number, and it took us a month to figure out the problem. The problem was that with the 3.3V, there wasn't enough current to send a steady signal from one Arduino to another. Other tutorial websites also explained to not plug the VCC into the 5V pin onto the Arduino, otherwise the module could be fried, so we were confused and stuck.

We then bought the adapter module, which has a 3.3V regulator on it. This allows us to plug the VCC into the 5V pin on the Arduino and prevent the board from getting fried. We attached the two types of modules together, as seen in Figure 14, plugged in VCC into the 5V pin on the Arduino, ran the sample program, and it finally worked.

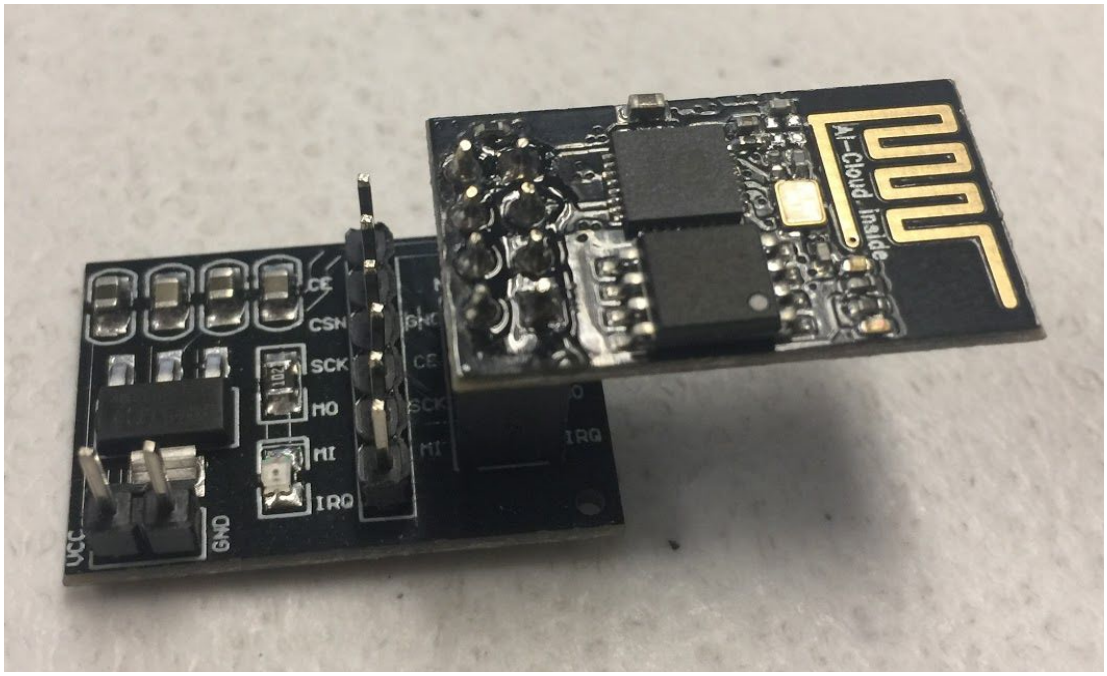


Figure 14. Connected NRF24LO1+ Transceiver Modules

Digital-to-Analog Conversion for Motor Control

Once we were able to properly use these modules and send data successfully, the next step was to figure out how to control the speed of the motor wirelessly. The motors that we are using have a throttle control that goes into the motor controller. See the motor datasheet to see which wire it is. A 0 to 5 volt signal is sent into the motor controller through this line, and is normally controlled by a potentiometer. The more voltage sent into the controller, the faster the motor goes. The problem is that you can't just send an analog signal from the Arduino into the controller because the motor controller would not be able to read a PWM signal. Instead, we needed to smooth out the signal, as seen in Figure 15, so the motor controller could receive a constant voltage reading.

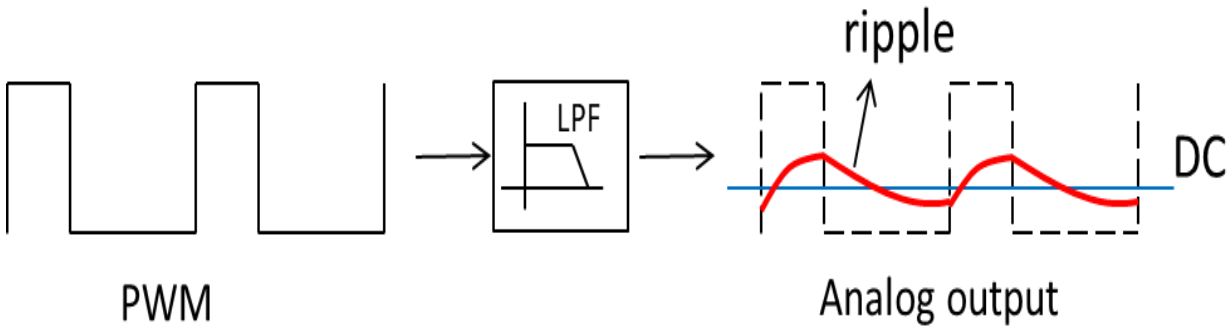


Figure 15. How a DAC smooths out a PWM signal. Retrieved from https://e2e.ti.com/blogs_/b/msp430blog/archive/2018/09/27/designing-an-audio-playback-Application-with-the-msp430-smart-analog-combo

To do this, we used an MCP4725 digital-to-analog converter to act as a low-pass filter and smooth out the signal, as seen in Figure 16.

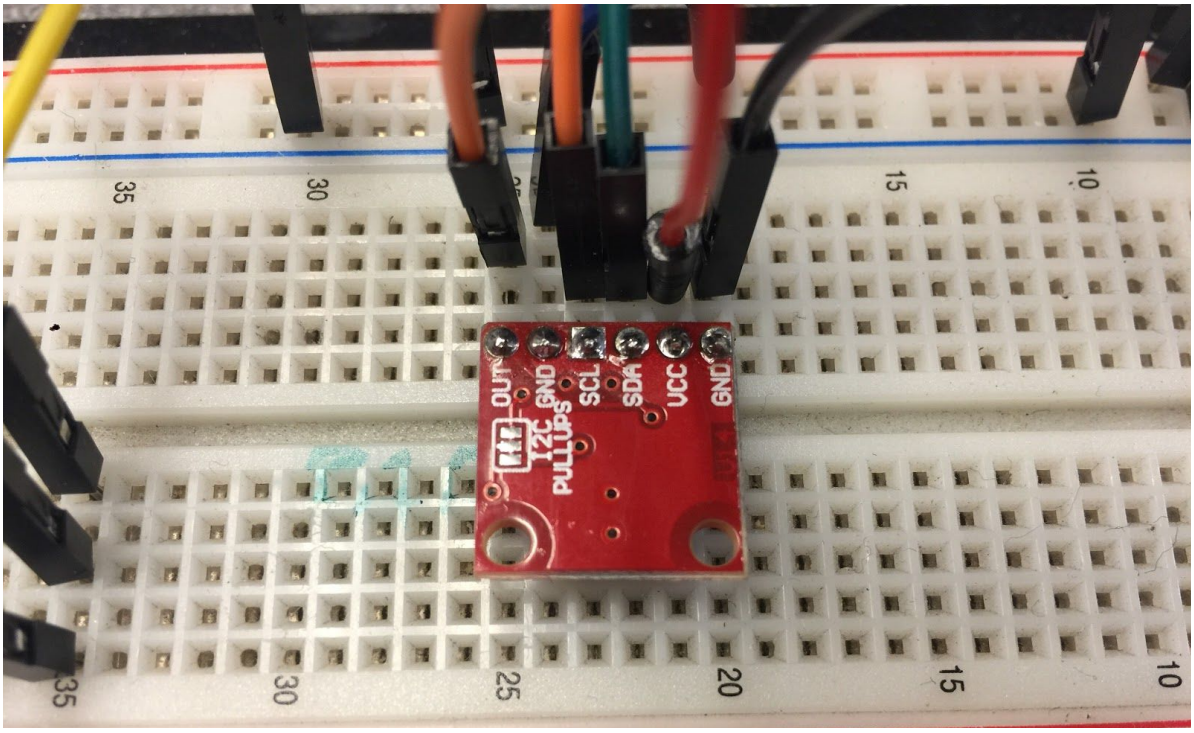


Figure 16. MCP4725 DAC wiring setup

This DAC works by receiving 5 volts from the arduino and then will output a desired voltage between 0 and 5 volts. The VCC pin plugs into the 5V Arduino pin, the right GND pin plugs into the GND Arduino pin, the SDA pin plugs into pin 20 on the Arduino, the SCL pin plugs into pin 21 on the Arduino, the left side GND pin plugs into the GND Arduino pin, and the OUT pin connects to the throttle wire going into the motor controller. To check and make sure that the DAC is working properly, upload the code in Appendix A named MCP4725 Test Code. Then place the ground lead from a multimeter to the left GND pin from the DAC, and place the positive lead from the multimeter to the OUT pin from the DAC. You should see the voltage jump from 0 volts to 5 volts with 3 second intervals.

Once the DAC is working, we can combine the wireless communication code with the DAC code and control the speed of the bogie wirelessly. In the controller program, you read the value of the potentiometer that controls the speed of the motor, and then map that value from 0 to 1023 from the potentiometer to 0 to 4095 for the dac. We map the potentiometer value to values between 0 to 4095 because from the MCP4725 Test Code, you can see that setting the dac voltage to 0 sends 0 volts into the motor controller, and setting the dac voltage to 4095 sends 5 volts into the motor controller. Setting the potentiometer to 0 will send 0 volts into the motor

controller, and setting the potentiometer to 1023 will send 5 volts into the motor controller. Once you have that dac output value, you can send the value to the bogie program, which will read the dac value and then set the output voltage for the dac to control the speed of the motor. The main parts of the program for this process can be seen in Figure 17 for the controller side, and Figure 18 for the bogie side.

```
//////////////////////////  CONTROLLER  ////////////////////////////////
void loop()
{
  pot_value = analogRead(pot_pin); //read potentiometer on controller
  uint32_t dac_value; //initialize DAC output value

  dac_value = map(pot_value, 0, 1023, 0, 4095); //map pot values to DAC values

  Serial.println(dac_value);
  radio.write(&dac_value, sizeof(dac_value)); //send dac value to bogie

  delay(50);
}
```

Figure 17. Controller snippet of speed control code.

```
//////////////////////////  BOGIE  ////////////////////////////////
void loop()
{
  radio.read(&dac_value, sizeof(dac_value)); //read DAC value from controller
  dac.setVoltage(dac_value, false); //set DAC output voltage to mapped value from pot
}
```

Figure 18. Bogie snippet of speed control code.

We setup the wireless controller, which will be talked about in the next section, to display the voltage being sent into the motor controller, and as you can see in Figure 19, the actual voltage being sent into the motor controller is very close to our program's expected voltage output value.

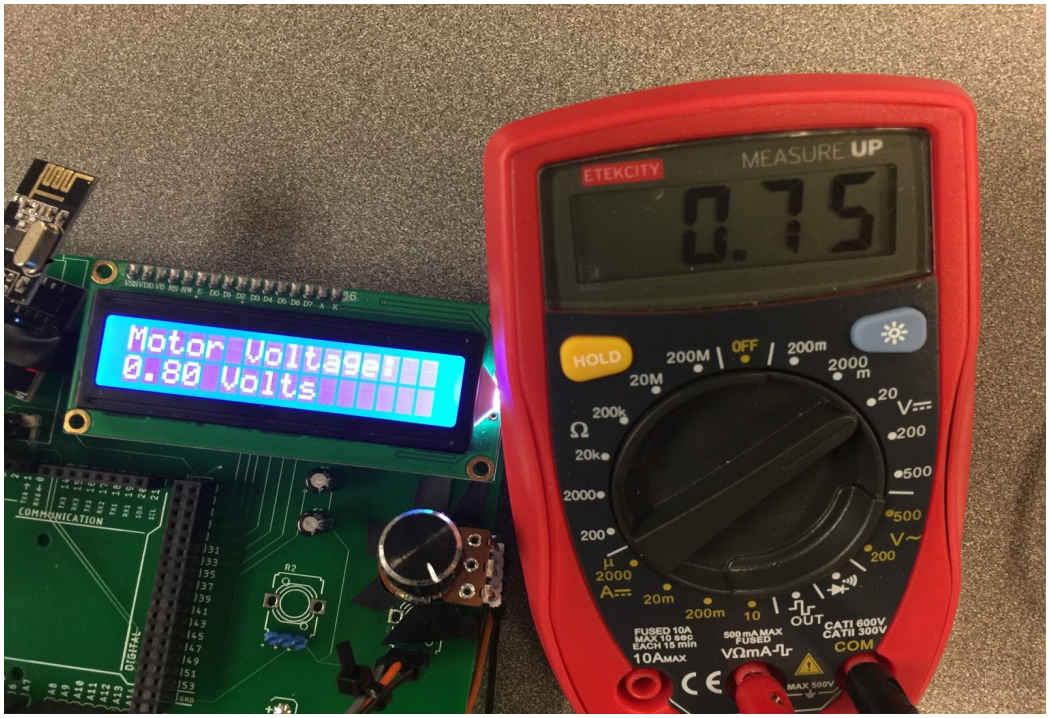


Figure 19. Comparison of expected voltage value to actual voltage value for speed control.

Controller Design and Setup

To ensure safety during testing however, we prefered to try and run the bogie wirelessly using a wireless controller. This wireless controller would be capable of sending a wireless signal to the motors and control them remotely to avoid high voltage and heavy equipment. The controller would be able to control the bogie manually and autonomously given a certain input.The wireless communication will be explained later in this report. To begin designing a wireless controller, we had to create a wiring schematic and choose what parts were to go on the controller. We decided to include momentary button switches for autonomous control and integration of the emergency brake, two potentiometers for brake and throttle analog, an Arduino MEGA, and a 16 by 2 LCD display to display feedback from the system. For the controller to work, the parts would have to be mounted onto a printed circuit board (PCB). Using Autodesk Eagle, we were able to design the wiring schematic and design the PCB. After a week of getting used to the new software, we came up with a wiring schematic for the controller which is displayed below.

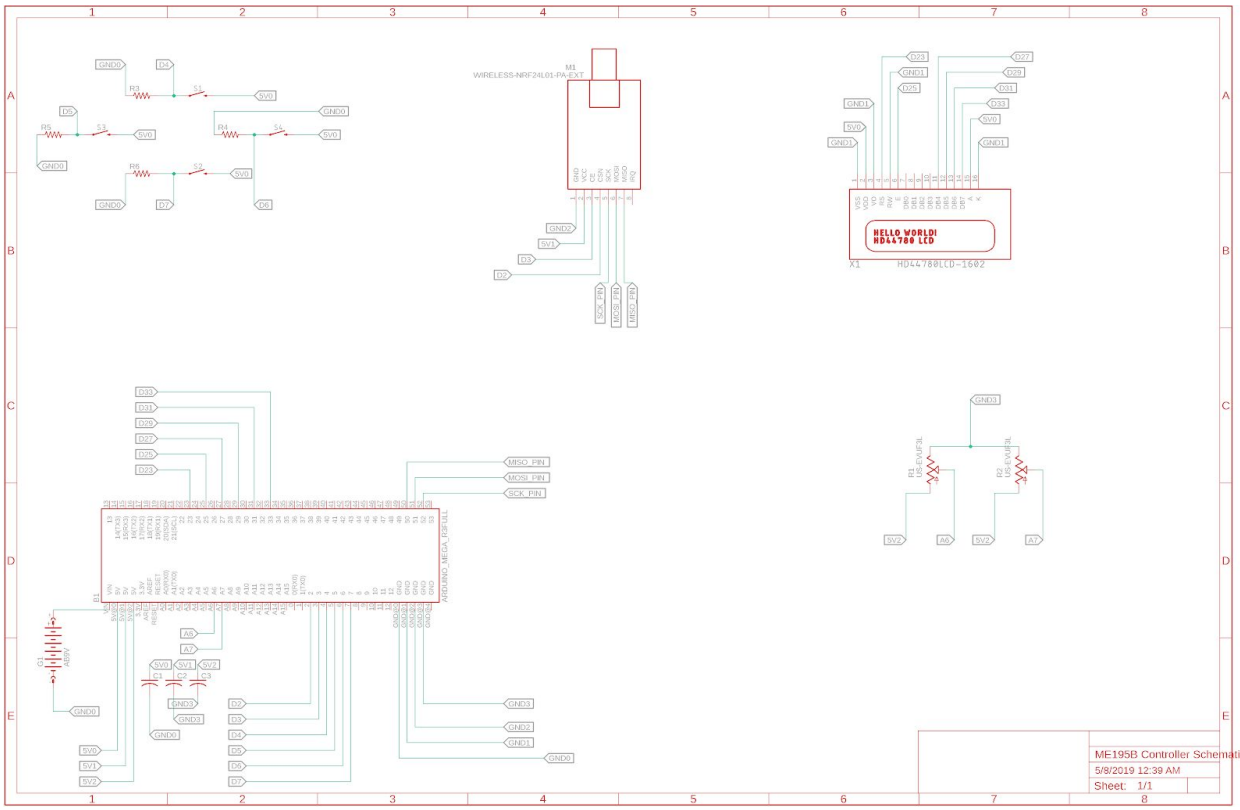


Figure 20. Wireless Controller Schematic

Once the schematic was finished, we then exported all the components to Eagle's PCB feature. Once the components were placed onto the board in the positions we wanted them, we then routed all the pins to their proper pinouts. Autorouter in Eagle is not very efficient and puts routes to close to each other. Manufactures may need about 3 to 7 millimeters in space between routes and vias and autorouter does not account for that. When purchasing a PCB, manufacturing is a big factor. The better the machining was, the more confident we were in placing our routes. The finished PCB is pictured below.

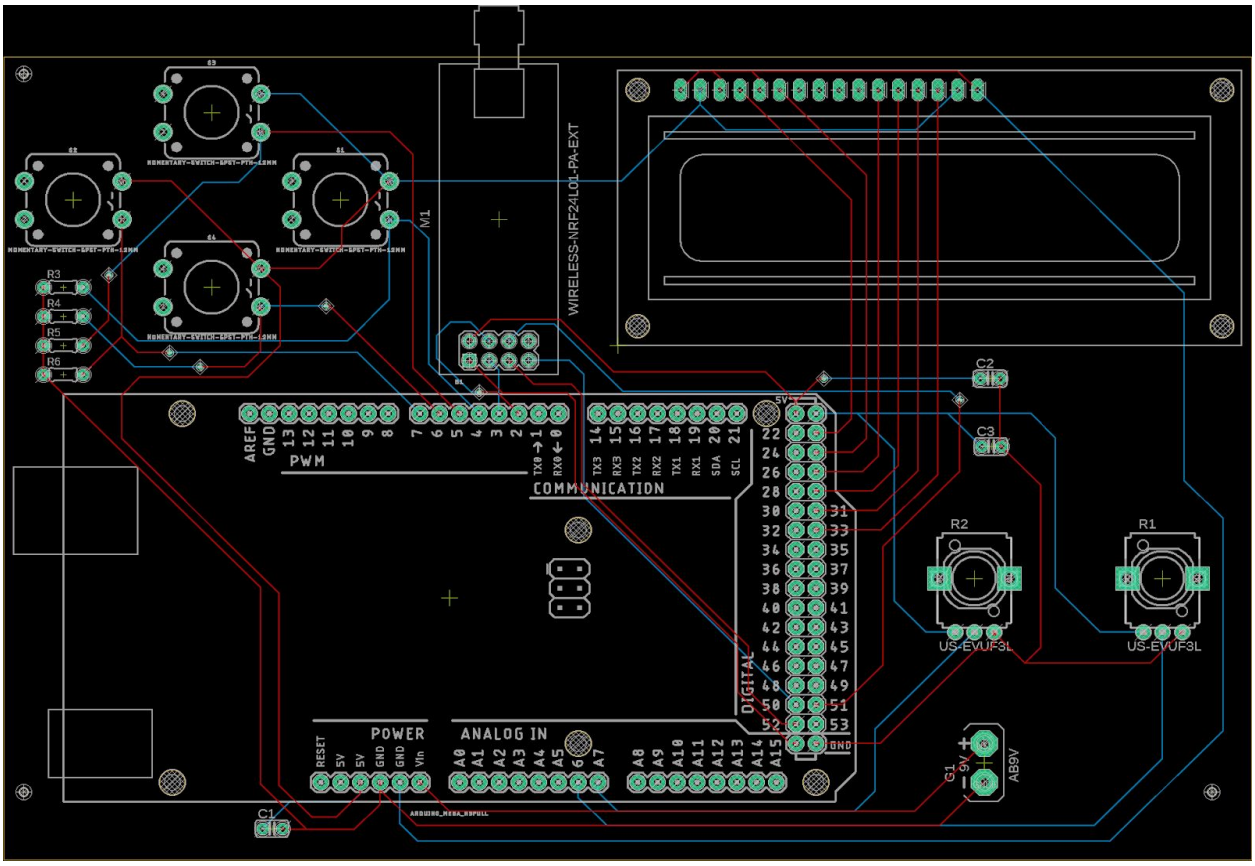


Figure 21. Wireless Controller PCB design

With the PCB design finished we sent of the files to be manufactured through the website JLCPCB which manufactures all types of PCBs' and was recommended to us by our controls team coach. Once the files were uploaded and the order was completed, processing began and the boards came in with about a 1 to 2 week lead time. The fully designed PCB is displayed below.

We then soldered all the components onto the board, as seen in Figure 22, and began testing. In order to send a proper signal we had to convert the pulse width modulation signal the Arduino outputs to a digital to analog converter (DAC) to obtain the 0 to 5 voltage required to start the motor. This is gone over more in detail in the wireless communication section of the report. With the motors now running wirelessly, we moved into autonomous controls where now we begin to program autonomous functions to control the bogie with only a couple of inputs. For this please refer to the programming section of the report. With the motor setup now fully

operational, we can now work on the process of controlling the speed of the motors to cruise at a specific speed. Due to time constraints we were not able to obtain graphical data of the acceleration and speed of the motors and possibly can be done next academic year to map the voltage to velocity and program set values for autonomous control. This concludes the electrical and most of the hardware of the motor control setup. The next sections will discuss wireless SPI communication and the Arduino code itself.

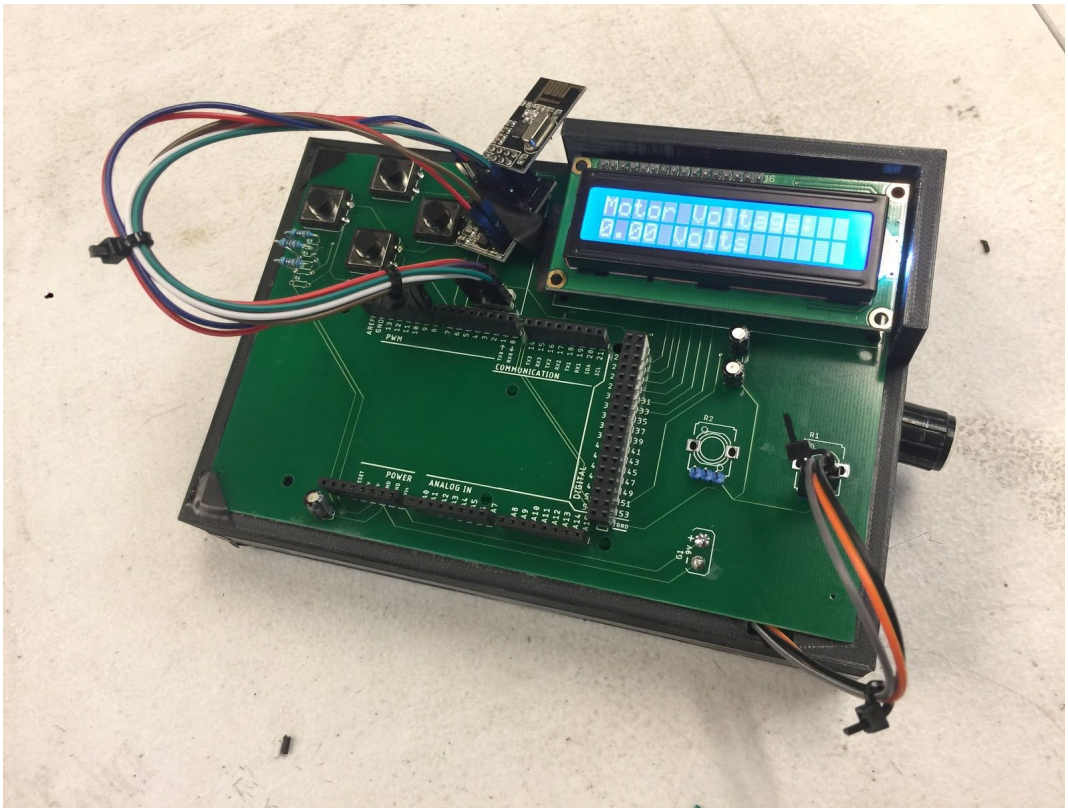


Figure 22. Finished PCB controller with 3D printed case

Major Code for Controller

The controller code is the brain of the whole system. It will run the main code for the controller and decide on what kind of voltage should be sent to the DAC. The input needed for this decision is the hall effect count from the bogie. The controller was made of PCB boards which we designed and manufactured by JLC PCB, a LCD display, 2 potentiometers, 4 switches with 4 pull-up resistor, a nRF24L01+ module with the adapter for it, and an Arduino Mega. The

code for the controller was much longer than the code for the bogie because all of the work done here for controlling the bogie.

```
//////////////////////////////////// LIBRARIES ////////////////////////////////////
//Wireless Communication Libraries
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>

//LCD Libraries
#include <LiquidCrystal.h>


//////////////////////////////////// DEFINITIONS ////////////////////////////////////
LiquidCrystal lcd(23, 25, 27, 29, 31 , 33); //LCD Screen Definition

//NRF24L01 Wireless Communication Pins
RF24 radio(8, 9); // CE, CSN

// NRF24L01 address through which two modules communicate.
const byte addresses[][6] = {"00001", "00002"};

#define button_pin1 2
#define button_pin2 3
#define button_pin3 4 //Controller button pins, change pins according to PCB
```

Figure 23. Libraries and definitions for the controller code

The main components on the PCB which needed libraries were the nRF24L01+ wireless communication module and the LCD screens which had special functions that needed to be executed in order for it to work. The libraries for the nRF24L01+ and the LCD display were not included in the Arduino IDE, so we had to download some of the libraries from the internet. Both of them were found on GitHub for free download and installing the zip file into Arduino was easy. Refer the links in Appendix C for the libraries. The SPI library is for the SPI communication protocol files that are need for the wireless communication to function properly.

Most of the pins on the Arduino were used by all the components on the PCB. The LCD screen required the most amount of pins. There are eight pins for data from D0 to D7, where the first 4 pins are only optional. We made the LCD display work with pins D4 to D7 because we do not need all the 8 lines for our application. For the wireless communication module, we used all the pins except for the IRQ pin, which was used the attachinterrupt function of the arduino, it is mainly used to put the system to sleep until some data is being received . For our use, we didn’t

need it to work in the background, most of the wireless data transmission happened real time whenever we needed and our arduino will be on at all times. Refer the datasheets below for the pin layouts. We installed 4 buttons on the controller but due to the problem with the nRF24L01+ where it does not recognize the fourth button, we had to go with 3 buttons instead.

```
//////////////////////////////// VARIABLES ////////////////////////
int button_pin1_status = 0;
int button_pin2_status = 0;
int button_pin3_status = 0; //Controller button pin status

uint32_t dac_value; //dac voltage to be sent to bogie

int pot_pin1 = 0; //Potentiometer 1, change pin according to PCB
int pot_value1 = 0; //Potentiometer 1 Value

int e_brake_pin = 5; //E-brake pin, change pin according to hookup

float volt_count;

// Function Prototype
int buttonread(int, int, int);
int manual_straight();
int auto_straight();
```

Figure 24. Variables for the controller code

For the program to work the way it is, we need to store important values into variables. We had to read all the button presses so that different commands can be done by the bogie. The values for the DAC are calculated on the controller and sent to the bogie. We used the data type 'uint32_t' because it guarantees that it will be a unsigned 32 bit resolution from 0 to (2^{32} - 1). The DAC value changes with the potentiometer value within in different functions depending on whether it is in manual or automatic mode. From the DAC value, we have to compare it to 5V so that a Arduino can control using the voltage supplied by it. This is the variable called volt_count has to be a floating point value because it needs to read all the small variations from the potentiometer so that there would not be any jumps in voltage. Finally the functions that we implemented in the code has to be declared with data type of the variables for input before starting the code so that the Arduino knows they are functions. The int values within the 'button_read' functions are for the input values that are needed for the function to work properly. We will use the e_brake_pin variable to control the E-brake that the articulation and braking team created.

```
//////////////////////////////// Setup ///////////////////////////

void setup()
{
  // put your setup code here, to run once:
  Serial.begin(9600);
  lcd.begin(16,2);
  lcd.print("Motor Voltage: ");

  pinMode(button_pin1, INPUT);
  pinMode(button_pin2, INPUT);
  pinMode(button_pin3, INPUT); //setting controller pins as inputs

  pinMode(e_brake_pin, HIGH);

  radio.begin();
  radio.openWritingPipe(addresses[1]); //00002
  radio.openReadingPipe(1, addresses[0]); //00001
}
```

Figure 25. Setup for the controller code

For the setup, we needed to start the LCD screen and set the buttons to input or output depending on what needs to be read. The LCD screen will always show the Motor Voltage so that we can monitor it if anything goes wrong. The button pins are set to input so that it reads all the button presses on the controller. The e_brake_pin is always set to high, which means the electro-magnets are engaged. It will be set to low in the cause when the E-brake is released. Finally we need to setup the the open channels for the data to flow between the bogie and the controller. If there are two addresses that means it is setup to be a bi-directional flow of data. One will be set to read from the bogie and the other one will be used to write commands to the bogie.

```
//////////////////////////////////// Loop /////////////////////////////////////
void loop()
{
  pot_value1 = analogRead(pot_pin1);
  // put your main code here, to run repeatedly:
  button_pin1_status = digitalRead(button_pin1);
  button_pin2_status = digitalRead(button_pin2);
  button_pin3_status = digitalRead(button_pin3);
  digitalWrite(e_brake_pin, HIGH); //reset e-brake status
  buttonread(button_pin1_status, button_pin2_status, button_pin3_status);

  dac_value = map(pot_value1, 0, 1023, 0, 4095);
  volt_count = map(dac_value, 0, 4095, 0, 500);
  lcd.setCursor(0,1);
  lcd.print(volt_count/100);
  lcd.setCursor(5,1);
  lcd.print("Volts");

  delay(50);
}
```

Figure 26. Loop for the controller code

For the loop, we basically read each element which is connected up to the controller first. Even though we specified that the E-brake will be high in the setup, we write it to be high again just so we reset the it when it starts looping. After that the program goes into the 'button_read' function, where it decided between automatic or manual modes to control the bogie. I will covering how the function works in the next section. Within this function and depending on the combination of button presses, we have other functions which would calculate the appropriate values for the DAC. Values from the function will be sent back to the main program to be mapped between the potentiometer, the DAC and operating voltage of the Arduino so that the LCD display can display the voltage being sent. First it is being mapped between the potentiometer and the DAC. The range for the potentiometer is from 0 - 1023, when it is 0, there will be no current flowing through it whereas, if it is at 1023, it mimics a fully closed circuit where current easily flows through it. The DAC that we are using has a 12 bit resolution, so the range for this is from 0 - 4095. After mapping it between the potentiometer and the DAC, we had to map it between the DAC and the operating voltage of the Arduino, which from 0V - 5V. This is the value being displayed on the LCD display.

```
//////////////////////////////// Read Button ////////////////////////////////////////

int buttonread(int button_pin1_status, int button_pin2_status, int button_pin3_status)
{
  if (button_pin1_status == LOW && button_pin2_status == LOW && button_pin3_status == HIGH)
  {
    dac_value = 0;
    lcd.setCursor(0,0);
    lcd.print("Manual: Going Straight");
    manual_straight();
    delay(300);
  }

  else if(button_pin1_status == LOW && button_pin2_status == HIGH && button_pin3_status == LOW)
  {
    dac_value = 0;
    lcd.setCursor(0,0);
    lcd.print("Automatic: Going Straight");
    dac_value = auto_straight();
    delay(300);
  }
  else
  {
    lcd.setCursor(0,0);
    lcd.print("Select Mode");
    delay(300);
  }
  return dac_value;
}
```

Figure 27. Button_Read function

For the button_read function, we need three main input from the controller to make it work and they are the the button_pin_status variables. After that it is ran through a if - else condition. Before every condition, the DAC voltage is set to zero so that it reset for each command. If button 1 and 2 are pressed, then it will run the first condition, it will prints ‘Manual: going straight’ on the LCD display and jumps into the maunal_straight function. If buttons 1 and 3 are pressed, then it will run the second condition, it will prints ‘Automatic: going straight’ on the LCD display and jumps into the auto_straight function. If no buttons are pressed, it will display ‘Select Mode’ on the LCD display until some combination of buttons are pressed. The idea of pressing buttons in combination is so that no one makes any accidental presses and only people who made the code will be able to operate it.

```
//////////////////////////////// Manual Straight ////////////////////////////////////////
int manual_straight()
{
  delay(5);
  uint32_t dac_value;
  radio.stopListening();
  pot_value1 = analogRead(pot_pin1);
  button_pin1_status = digitalRead(button_pin1);
  button_pin2_status = digitalRead(button_pin2);
  button_pin3_status = digitalRead(button_pin3);
  if (button_pin1_status == LOW && button_pin2_status == LOW && button_pin3_status == LOW)
  {
    dac_value = 0;
    lcd.setCursor(0,0);
    lcd.print("Emergency Brake");
    digitalWrite(e_brake_pin, LOW);
    radio.write(&e_brake_pin, sizeof(e_brake_pin));
  }
  else
  {
    dac_value = map(pot_value1, 0, 1023, 0, 4095); //map pot value to dac output value
  }
  radio.write(&dac_value, sizeof(dac_value));
  delay(5);
  int right_hall_counter;
  radio.startListening();
  while(!radio.available());
  radio.read(&right_hall_counter, sizeof(right_hall_counter));
  lcd.setCursor(0,0);
  lcd.print("Bogie passed: ");
  lcd.setCursor(0,14);
  lcd.print(right_hall_counter);
  if (right_hall_counter = 4)
  {
    buttonread(button_pin1_status, button_pin2_status, button_pin3_status);
  }
  return dac_value;
}
```

Figure 28. Manual_Straight function

The manual_straight function is when buttons 1 and 2 are pressed. This is when the we travel along the straight part of the track controlled by the potentiometer. For this we need to read values from the potentiometer and the buttons. We have two conditions to satisfy here, one for when something happens during the travel and we need to stop it - so by pressing all three buttons at once, we set the DAC voltage to zero and releases the emergency brake. If all three buttons are not pressed then it will map the value between the potentiometer and the DAC. This

value is then sent to the bogie which makes it moves. After setting it to a comfortable speed, the hall-effect count value from the bogie is sent to the controller so that it can make better decisions on when to stop or reset back to its initial state. The LCD display will print out where the bogie is using the hall-effect sensor. When the count hits 4, that means it reached its initial starting point. For the motors to brake and reverse we have implemented 2 switch for each action. When the bogie reaches the end of the track, we would zero out the potentiometer in order to stop and then we would manually go and turn the brake on first and then the reverse. After that we will use the potentiometer again to run it back to the starting point. When hall - effect sensor reads 4, it goes back to the button_read function. The value calculated for the DAC voltage is sent back to the loop so that the LCD display can print it out.

```
//////////////////////////////// Auto Straight ////////////////////////////,
int auto_straight()
{
  delay(5);
  radio.stopListening();
  if (dac_value < 4095)
  {
    dac_value = dac_value + 819;
    delay(50);
  }
  radio.write(&dac_value, sizeof(dac_value));
  int right_hall_counter = 0;
  delay(5);
  radio.startListening();
  while (!radio.available());
  radio.read(&right_hall_counter, sizeof(right_hall_counter));
  delay(5);
  radio.stopListening();
  if (right_hall_counter = 2)
  {
    dac_value = 0;
    radio.write(&dac_value, sizeof(dac_value));
    delay(100000000);
  }
  if (right_hall_counter = 5)
  {
    dac_value = 0;
    radio.write(&dac_value, sizeof(dac_value));
    buttonread(button_pin1_status, button_pin2_status, button_pin3_status);
  }
  return dac_value;
}
```

Figure 29. Auto_Straight function

The auto_straight function is activated when buttons 1 and 3 are pressed. The main condition for setting the speed is by comparing the DAC voltage to its maximum value. If the DAC voltage is smaller than 4095 then it will add 819, which is 1V for the DAC, each time to ramp up the speed from 0 RPM slowly. After it reached a desired value, we send it to the bogie to be read. Then scanning for the hall_effect sensor is the next important task to perform. The hall effect count has to reset before the bogie starts moving. When the right_hall_counter reads 0 that means it is at the starting position, from here it perform the calculate needed to set the DAC voltage so that the bogie starts moving. When the right_hall_counter reads 2, that mean it needs to stop because it is reaching the end of the track. So we set the DAC voltage to 0 to bring the bogie to a stop. We implemented a big delay so that we have each time to go over to the bogie in order to manually turn the brake and reverse switches on. After that, it will continue to move with the previously calculated DAC voltage until it reads 5 on the right_hall_counter. When it reads 5, that means the bogie return back to its starting position. Therefore we set the DAC voltage to 0 to stop it. A DAC value is sent back to the main loop to be displayed by the LCD display.

Major Code for Bogie

The bogie moves because of the inputs from the controller that the team designs. The bogie does not have much processing to do within the code. It basically just writes the DAC value sent from the controller onto the bogie and that value controls how much voltage to be sent to the motors. The components on the bogie consists of the nRF24L01+ with its adapter, the MCP4725 DAC, a hall effect sensor on the right side of the bogie, and an Arduino Mega, which is connected to the motor controller. Refer the data sheet given in the appendix for pin layouts.

```
//////////////////////////////////// LIBRARIES ///////
//Wireless Communication Libraries
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>

//DAC
#include <Wire.h>
#include <Adafruit_MCP4725.h>
Adafruit_MCP4725 dac;

//////////////////////////////////// DEFINITIONS /////
//Hall Effect sensor on both sides of the bogie
#define e_brake_pin 11 //change according to pinout

volatile byte state = LOW;

//NRF24L01 Wireless Communication Pins
RF24 radio(7, 8); // CE, CSN

const byte interruptPin = 9;
```

Figure 30. Libraries and definitions for the bogie code

Some parts of the code are similar to the controller code. The main components that needed libraries were the nRF24L01+ and the MCP4725 DAC. The links for these libraries are available in the appendix of this report. Other than the SPI communication protocol, MCP4725 DAC requires I2C communication protocol for data transfer. So we have to include the wire library for I2C communication, this is already built into the Arduino IDE.

In terms of definitions, we need one pin for the emergency brake and another for the hall effect sensor. For the hall effect sensor, we need to set a variable, called state, as low as default. When it reads a magnetic field, a voltage will be created and the state value will change to high. We set this as a volatile byte cause the variable ‘state’ will change from high to low or low to high depending on the position of the magnetic field with respect to the hall effect sensor. In order for it to work, a pull-up resistor has to be placed between the Arduino pin and the output. Finally we need to set the CE and CSN pins for the nRF24L01+, just like we did for the controller.

```
//////////////////////////////// VARIABLES /////////////////////////////////////////
// NRF24L01 address through which two modules communicate.
const byte addresses[][6] = {"00001", "00002"};

//Position detection varible

volatile int right_hall_counter = 0;

uint32_t dac_value;

//////////////////////////////// SETUP /////////////////////////////////////////////
void setup()
{
  pinMode(e_brake_pin, HIGH);

  pinMode(interruptPin, INPUT_PULLUP); //hall effect

  attachInterrupt(digitalPinToInterrupt(interruptPin), positiondetect_right, FALLING);

  dac.begin (0x60);

  radio.begin();
  radio.openWritingPipe(addresses[1]); //00002
  radio.openReadingPipe(1, addresses[0]); //00001
}
```

Figure 31. Variables and setup for the bogie code

Similar to the controller code, we need to setup open channels for the nRF24L01+. We have 2 addresses that will act as a bi-directional data transfer between the bogie and the controller. The DAC has to be initialized using the 'uint32_t', because we need a 32-bit resolution for it work. Finally, a variable for the right hall effect sensor has to be made so that each variation in voltage would be recorded. In terms of setup, we have the emergency pin is always set to high so that the electromagnets are engaged. We set the hall effect pin to 'input_pullup' so that the arduino can read all the voltage from it. After that we use the attachinterrupt command, in order to read the hall effect in the background while the rest of the program is being ran. For this command there are three parameters to be fulfilled - the specific pin that would be interrupted to read data, an ISR function and the Mode. The first one is figure out which pin will be used to read data from the sensor. When a voltage is detected it will stop the main loop and head to the function that needs to be read. When an interrupt occurs, the code

skips to the function that we designed to be for that sensor. Finally, mode has to be chosen to see what kind of data we need to be read. There are 4 options and they are low, change, rising and falling. In our case, when the hall effect sensor reaches the magnet, it will produce a voltage and it moves to the function from the main code. At this point, even though the sensor read high, we did not start counting yet. When it moves past the magnet and the sensor goes from high to low, then the counting function will add 1 to the count. As in the previous code, we had to setup which addresses are used for reading and writing data to and from the bogie and the controller wirelessly. When the DAC uses I2C as the communication protocol, an address has to chosen like the nRF24L01+, in order to send the right data at the right time. For our DAC model, the address is set to 0x60.

```
/////////////////////////////////// LOOP //////////////////////
void loop() {
   dac_value = 0;

  if(radio.available())
  {
    radio.read(&dac_value, sizeof(dac_value));
  }

  dac.setVoltage(dac_value, false);
}

/////////////////////////////////////// INTERRUPTS ////////////
//HALL SENSOR DETECTION//
//RIGHT
void positiondetect_right()
{
  right_hall_counter = right_hall_counter +1;

  radio.stopListening();
  radio.write(&right_hall_counter, sizeof(right_hall_counter));
  delay(5);
}
```

Figure 32. Loop and interrupt function for bogie code

The loop for the bogie is really simple. The only task it has to perform is to write data from the controller, mainly the DAC voltage value in order to run the motors. Initially we set it to be zero but by pressing buttons or by turning the potentiometer on the controller, the value

changes accordingly. When the wireless communication module is available, it will write values into the appropriate variables. Using the syntax dac.setVoltage(dac_value, false), we are able to set that as the voltage for the DAC. The final part to the code is the interrupt function. As discussed earlier, when a signal is detected the whole code pauses and runs this function called positiondetect_right(). When the bogie moves past the magnet and voltage goes from high to low, then the right_hall_counter variable adds one to it. Each time this happens, the function adds one and at each value corresponds to some condition within the controller program that has to be satisfied. After this, we had to send the data back to the controller so that it can perform more tasks.

Analysis/Validation/Testing

This semester, we were unable to test our system as a whole considering the bogie was not fully put together and the track was not fully put together. We could not validate if our automatic mode worked on our controller and we could not test to make sure the position tracking system code worked. However, what we could test and validate was that the motors and motor controllers worked properly, the manual mode program worked, and the wireless communication worked. We wrote a simpler and smaller program to make sure that the manual mode worked on the controller just so that we could present to the class and our guests how the controller and motors worked. We also wrote a small program to make sure that the hall effect sensors worked, we just couldn't test if they would work on the track itself. Overall, what we could test and validate was able to work properly. Hopefully next year, the entire controls system will be able to be tested and troubleshooted.

CHAPTER 4

Budget

The team had many revisions of the Bill of Material google sheets to get to the final total of expenses. Most of the purchase were contributed by all the team members and the AS government reimbursed it us. Our initial budget that we estimated was $567.91 with all the essentials needed to get the project started. We were meeting our estimated budget till half through the second semester but as we ran into trouble with the motors, the wireless communication modules and motor controllers. We had to buy new modules, wires and other miscellaneous parts to make sure all the components to work together. In the end, our budget came out to be $789.30. The components that we bought were put to good use on the bogie and the controller. Most of the parts came in a kit with different configurations, like resistors, diodes, capacitors, heat shrink tubes, wires, wire connectors, ring terminals and linear actuators. The extras can be used by next years team so that when they start their project, they will have all the necessary equipment to start the project. All the extra parts are stored in the cabinets behind the work area.

Even though our budget was high compared to other teams, we required components that were essential to start working on the project. We bought whatever was necessary at the time so that there would be no trouble moving ahead with the project. Before starting the project, the team salvaged parts from previous years and most of them were still usable. So that definitely helped move thing along in the beginning. Prof. Furman even gave us hall effect sensors and a rotary encoder for us to test and use. With the help of the professor and the parts available at superway, we brought our budget down by around $100. The detailed Bill of Materials is available in the appendix below and also on the Spartan Superway archive. Most of our purchases were done through Amazon or Digikey. With Amazon's one to two day shipping, we were able to get all the parts as soon as possible. The PCB that Justin created was sent to JLC PCB, they gave us a first time customer discount of about $20. We used that money to pay for expedited shipping so that we would have the controller by evaluation day.

Results and Discussion

The work done by the full-scale controls team this academic year was a big step in the right direction. After getting the motors to run successfully and implementing wireless control technology in the form of a wireless PCB controller sets up next years controls team with a lot to build upon. With baselines set among all the main objectives set forth in the beginning of the academic year, future controls teams will be able to take our work and build upon in and create possibly a more advanced system that will substantially improve the project as a whole. The work done this year by us will allow future teams to get off the ground earlier, test parts and work more on the design process by being able to iterate multiple designs within a year instead of working on one design and hoping it will work at the end. With the control of the motors fully operational, once a track is permanently built, real life testing can be done versus all simulation work. Simulation work provides a great baseline and what to expect but testing the designs in real time will give more useful information. This will encourage more rapid prototyping amongst teams so they can test their designs and look to continuously build upon the project and use their time as efficiently as possible.

At this point in time, it will allow next year's team to get a great head start into what they do and don't need to do. Learning how to create PCB will allow next years team to develop better mounts for hardware components and fit them into smaller spaces on the drive train. The existing Arduino code does not have to be written from scratch and can be built upon to improve manual and autonomous control of the bogie along with how to setup wireless SPI connection which will allow them to work with ideas outside of huge wire nests. Overall, we believe we have placed a great foundation for future teams and provided the building blocks for which they can improve Spartan Superway and get it closer to becoming part of our everyday lives. We encourage future teams to learn, grow, and expand on what we have provided them and urge them to learn as much as they can about control systems, coding, and electrical engineering. As mechanical engineers, we are not well versed in software and hardware, but a few weeks of understanding and learning what we have done, the software and hardware we have used, and the technical skills we have learned will allow for Spartan Superway to grow for the foreseeable future.

Conclusion and Recommendations

Reflecting on the work the team completed over the last year, we can confidently say that our project was a success and we were able to meet all of our design specifications. The most substantial goal was being able to control the bogie remotely, which we were able to do consistently. We were also able to create a remote control to house all the components we needed to control the bogie remotely. Since the track wasn't complete, we were unfortunately unable to test and see if the motors were able to meet their expected design specifications, but we are still confident that the motors would be able to move the bogie throughout the track and reach the 2 mph goal. We also were not confident enough with our program to test and see if our position tracking system would work, but we have the hall effect sensors working properly so we just need more time to test that part of our code. Besides these setbacks, our controls system was able to be put together and work in the way that we wanted it to.

Over the course of the year, while we were responsible for different aspects of the project, we all shared what we learned and were able to accumulate skills that will be directly applicable to our lives as engineers. The team experienced problems with project timing and procrastination, so we learned how important it is to make an accurate Gantt chart and stick with it throughout the course of the project. We also learned how important cross-team communication is, considering our group needed to incorporate electrical components from multiple teams. In terms of technical skills, we learned about sensor and linear actuator integration into systems, PCB design, and brushless DC motors. Lastly, a lot of work went into the programming of our system, so we learned how important proper formatting and commenting is so that troubleshooting the program can be a simpler process with organized code.

While a lot of work was done this year, there are some things that we were unable to complete, mainly due to time constraints. For next year's full-scale controls team, we recommend doing a lot of research on the specific components that are or will be in the final system during the first semester of senior project. This will make the process of putting the system together and programming the bogie and controller simpler during the second semester. We ran into a lot of problems with the wireless communication because we did not do enough research on the modules itself. In terms of what to improve upon next semester, we recommend

improving the position tracking system and making it more accurate by including a rotary encoder. We also recommend programming more system diagnostics in the major codes so the user knows more information about the system. The last recommendation would be to use as much of our work as possible. Our team wanted to start from scratch, but that just gave us a bigger workload and costed us more time when we could have just studied what last year's team did to improve the system. The controls teams are making great progress and getting a lot of work done each year, and it's amazing to see our work helping to get Superway closer to a complete Automated Transit Network.

References

"Advanced Transit - Stadsregio Rotterdam Business Park Rivium." *ATRA*, www.advancedtransit.org/advanced-transit/applications/rivium/

"Advanced Transit - Suncheon Bay Ecotrans PRT." ATRA, www.advancedtransit.org/advanced-transit/applications/suncheon/.

"Basics of the SPI Communication Protocol." *Circuit Basics*, 23 May 2018, www.circuitbasics.com/basics-of-the-spi-communication-protocol/.

"Climate Change Indicators: Greenhouse Gases." *EPA*, Environmental Protection Agency, 22 Feb. 2017, www.epa.gov/climate-indicators/greenhouse-gases.

Ellis, S., Fabian, L., Furman, B., Muller, P., & Swenson, R. (2014, September). *Automated Transit Networks(ATN): A review of the State of the Industry and Prospects for the Future* [PDF File]. Retrieved December 13, 2018, from https://www.inist.org/library/2014-09.Furman.Automated%20Transit%20Networks.MTI%201227.pdf

Fast Facts on Transportation Greenhouse Gas Emissions. (2018, August 27). Retrieved May 14, 2019, from https://www.epa.gov/greenvehicles/fast-facts-transportation-greenhouse-gas-emissions

"Masdar City Archives." *2getthere*, www.2getthere.eu/tag/masdar-city/.

National Express Transit. (August 17, 2017). 5 Transportation Challenges in Urban Areas. Retrieved May 14, 2019, from https://www.nationalexpresstransit.com/blog/5-transportation-challenges-in-urban-areas/

"PRT (Personal Rapid Transit) | PRT Facts." *PRT (Personal Rapid Transit) at West Virginia University*, prt.wvu.edu/about-the-prt/prt-facts

Rodrigue, J. (2018, November 08). Urban Transport Challenges. Retrieved May 14, 2019, from https://transportgeography.org/?page_id=4621

San Francisco Bay Area 5th Worst Traffic Congestion In The World, Study Finds. (2018, February 06). Retrieved December 13, 2018, from https://sanfrancisco.cbslocal.com/2018/02/06/san-francisco-5th-worst-traffic-congestion-in-the-world-inrix-study/

Sources of Greenhouse Gas Emissions. (2018). Retrieved May 14, 2019, from https://www.epa.gov/ghgemissions/sources-greenhouse-gas-emissions

Transportation Research Board of the National Academies. (2013). *Critical Issues in Transportation* [PDF File]. Retrieved December 13, 2018, from http://onlinepubs.trb.org/onlinepubs/general/criticalissues13.pdf

Appendices

Appendix A - Arduino Code

MCP4725 Test Code:

```
#include <Wire.h>
#include <Adafruit_MCP4725.h>

Adafruit_MCP4725 dac; // constructor

void setup(void) {

  dac.begin(0x60); // The SPI Address: Run the SPI Scanner if you're not sure

}

void loop(void) {

    uint32_t dac_value;

    // About 1V Out from MC4725
    // About 2V Out from LM358
    dac.setVoltage(819, false);
    delay(3000);

    // About 2V Out from MC4725
    // About 4V Out from LM358
    dac.setVoltage(1638, false);
    delay(3000);

    // About 3V Out from MC4725
    // About 6V Out from LM358
    dac.setVoltage(2457, false);
    delay(3000);

    // About 4V Out from MC4725
    // About 8V Out from LM358
```

```
    dac.setVoltage(3276, false);
    delay(3000);

    // About 5V Out from MC4725
    // About 10V Out from LM358
    dac.setVoltage(4095, false);
    delay(3000);


}
```

Main Controller Program:

```
//////////////////////////////////// LIBRARIES /////////////////////////////////////////////
//Wireless Communication Libraries
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>

//LCD Libraries
#include <LiquidCrystal.h>


//////////////////////////////////// DEFINITIONS /////////////////////////////////////////////
LiquidCrystal lcd(23, 25, 27, 29, 31 , 33); //LCD Screen Definition

//NRF24L01 Wireless Communication Pins
RF24 radio(8, 9); // CE, CSN

// NRF24L01 address through which two modules communicate.
const byte addresses[][6] = {"00001", "00002"};

#define button_pin1 2
#define button_pin2 3
#define button_pin3 4 //Controller button pins, change pins according to PCB

//////////////////////////////////// VARIABLES /////////////////////////////////////////////
int button_pin1_status = 0;
int button_pin2_status = 0;
int button_pin3_status = 0; //Controller button pin status
```

```
uint32_t dac_value; //dac voltage to be sent to bogie

int pot_pin1 = 0; //Potentiometer 1, change pin according to PCB
int pot_value1 = 0; //Potentiometer 1 Value

int e_brake_pin = 5; //E-brake pin, change pin according to hookup

float volt_count;

// Function Prototype
int buttonread(int, int, int);
int manual_straight();
int auto_straight();


//////////////////////////////// Setup ///////////////////////////////////////////

void setup()
{
  // put your setup code here, to run once:
  Serial.begin(9600);
  lcd.begin(16,2);
  lcd.print("Motor Voltage: ");

  pinMode(button_pin1, INPUT);
  pinMode(button_pin2, INPUT);
  pinMode(button_pin3, INPUT); //setting controller pins as inputs

  pinMode(e_brake_pin, HIGH);

  radio.begin();
  radio.openWritingPipe(addresses[1]); //00002
  radio.openReadingPipe(1, addresses[0]); //00001
}


//////////////////////////////// Loop ////////////////////////////////////////////
void loop()
```

```
{
  pot_value1 = analogRead(pot_pin1);
  // put your main code here, to run repeatedly:
  button_pin1_status = digitalRead(button_pin1);
  button_pin2_status = digitalRead(button_pin2);
  button_pin3_status = digitalRead(button_pin3);
  digitalWrite(e_brake_pin, HIGH); //reset e-brake status
  buttonread(button_pin1_status, button_pin2_status, button_pin3_status);

  dac_value = map(pot_value1, 0, 1023, 0, 4095);
  volt_count = map(dac_value, 0, 4095, 0, 500);
  lcd.setCursor(0,1);
  lcd.print(volt_count/100);
  lcd.setCursor(5,1);
  lcd.print("Volts");

  delay(50);
}


//////////////////////////////// Read Button ////////////////////////////////////////

int buttonread(int button_pin1_status, int button_pin2_status, int button_pin3_status)
{
  if (button_pin1_status == LOW && button_pin2_status == LOW && button_pin3_status ==
HIGH)
  {
    dac_value = 0;
    lcd.setCursor(0,0);
    lcd.print("Manual: Going Straight");
    manual_straight();
    delay(300);
  }

  else if(button_pin1_status == LOW && button_pin2_status == HIGH && button_pin3_status
== LOW)
  {
    dac_value = 0;
    lcd.setCursor(0,0);
```

```
   lcd.print("Automatic: Going Straight");
   dac_value = auto_straight();
   delay(300);
  }
  else
  {
   lcd.setCursor(0,0);
   lcd.print("Select Mode");
   delay(300);
  }
  return dac_value;
}


//////////////////////////////// Manual Straight ////////////////////////////////////////
int manual_straight()
{
  delay(5);
  uint32_t dac_value;
  radio.stopListening();
  pot_value1 = analogRead(pot_pin1);
  button_pin1_status = digitalRead(button_pin1);
  button_pin2_status = digitalRead(button_pin2);
  button_pin3_status = digitalRead(button_pin3);
  if (button_pin1_status == LOW && button_pin2_status == LOW && button_pin3_status ==
LOW)
  {
   dac_value = 0;
   lcd.setCursor(0,0);
   lcd.print("Emergency Brake");
   digitalWrite(e_brake_pin, LOW);
   radio.write(&e_brake_pin, sizeof(e_brake_pin));
  }
  else
  {
   dac_value = map(pot_value1, 0, 1023, 0, 4095); //map pot value to dac output value
  }
  radio.write(&dac_value, sizeof(dac_value));
  delay(5);
```

```
  int right_hall_counter;
  radio.startListening();
  while(!radio.available());
  radio.read(&right_hall_counter, sizeof(right_hall_counter));
  lcd.setCursor(0,0);
  lcd.print("Bogie passed: ");
  lcd.setCursor(0,14);
  lcd.print(right_hall_counter);
  if (right_hall_counter = 4)
  {
    buttonread(button_pin1_status, button_pin2_status, button_pin3_status);
  }
  return dac_value;
}


//////////////////////////////// Auto Straight ////////////////////////////////////////
int auto_straight()
{
  delay(5);
  radio.stopListening();
  if (dac_value < 4095)
  {
    dac_value = dac_value + 819;
    delay(50);
  }
  radio.write(&dac_value, sizeof(dac_value));
  int right_hall_counter = 0;
  delay(5);
  radio.startListening();
  while (!radio.available());
  radio.read(&right_hall_counter, sizeof(right_hall_counter));
  delay(5);
  radio.stopListening();
  if (right_hall_counter = 2)
  {
    dac_value = 0;
    radio.write(&dac_value, sizeof(dac_value));
    delay(100000000);
```

```
  }
  if (right_hall_counter = 5)
  {
    dac_value = 0;
    radio.write(&dac_value, sizeof(dac_value));
    buttonread(button_pin1_status, button_pin2_status, button_pin3_status);
  }
  return dac_value;
}
```

Main Bogie Program:

```
//////////////////////////////////// LIBRARIES /////////////////////////////////////////
//Wireless Communication Libraries
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>

//DAC
#include <Wire.h>
#include <Adafruit_MCP4725.h>
Adafruit_MCP4725 dac;

//////////////////////////////////// DEFINITIONS /////////////////////////////////////////
//Hall Effect sensor on both sides of the bogie
#define e_brake_pin 11 //change according to pinout

volatile byte state = LOW;

//NRF24L01 Wireless Communication Pins
RF24 radio(7, 8); // CE, CSN

const byte interruptPin = 9;

//////////////////////////////////// VARIABLES /////////////////////////////////////////
// NRF24L01 address through which two modules communicate.
const byte addresses[][6] = {"00001", "00002"};

//Position detection varible
```

```
volatile int right_hall_counter = 0;

uint32_t dac_value;

////////////////////////////////// SETUP ////////////////////////////////////////////////
void setup()
{
  pinMode(e_brake_pin, HIGH);

  pinMode(interruptPin, INPUT_PULLUP); //hall effect

  attachInterrupt(digitalPinToInterrupt(interruptPin), positiondetect_right, FALLING);

  dac.begin (0x60);

  radio.begin();
  radio.openWritingPipe(addresses[1]); //00002
  radio.openReadingPipe(1, addresses[0]); //00001
}
////////////////////////////////// LOOP /////////////////////////////////////////////////
void loop() {
   dac_value = 0;

  if(radio.available())
  {
    radio.read(&dac_value, sizeof(dac_value));
  }

  dac.setVoltage(dac_value, false);
}

////////////////////////////////////// INTERRUPTS ////////////////////////////////////
//HALL SENSOR DETECTION//
//RIGHT
void positiondetect_right()
{
  right_hall_counter = right_hall_counter +1;

  radio.stopListening();
```

```
  radio.write(&right_hall_counter, sizeof(right_hall_counter));
  delay(5);
}
```

Appendix B - Bill of Materials

(Link to BOM: https://docs.google.com/spreadsheets/d/1jSPuLsKxgV-ihB-dZJXJIyE0iR47rRj9MzP8CKsaJmw/edit#gid=0)

CONTROLLER

Part No.	Description	Model #	Quantity	Unit Cost (include taxes)	Total
1	NRF24L01+	Makerfire 10pcs Arduino N	1	$11.98	$11.98
2	NRF24L01+ Adapter	Makerfire 4pcs NRF24L01+	1	$8.99	$8.99
3	Tact Tactile Push Button	DAOKI 100Pcs 6x6x6 mm M	1	$4.48	$4.48
4	Toggle switch	Nilight Heavy Duty Rocker	1	$10.40	$10.40
5	LED display	Available at Superway	1		$0.00
6	Potentiometer	WGCD WMYCONGCONG 1	2	$12.99	$25.98
7	PCB Board 40Pin Male Pin I	Glarks 70Pcs 2.54mm Stra	2	$9.93	$19.86
8	Tactile Push Button	Hilitchi 250-Pcs 6 x 6mm 1	1	$12.99	$12.99

BOGIE

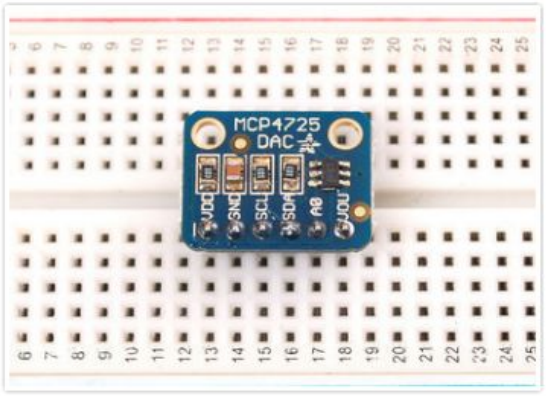
Part No.	Description	Model #	Quantity	Unit Cost (include taxes)	Total
9	Arduino Mega	ARDUINO MEGA 2560 REV	2	$33.00	$66.00
10	Arduino Mega Shield	Electronics-Salon Prototyp	3	$22.99	$68.97
11	Motors	Provided by Motor Team	3	$0	$0
12	Motor Controller	Provided by Motor Team	2	$0	$0
13	Hall Effect	Available at Superway	2	$0	$0
14	Linear Actuator	Install Essentials 524N High	3	$16.75	$50.25
15	Super Capacitors	Provided by Wayside Powe	30	$0	$0
16	Battery	12 Volt car battery - availab	2	$0	$0
17	Contactor	Provided by Motor Team	2	$6.99	$13.98
18	Fuse	Buck Converter (200 mA), S	1	$12.00	$12.00
19	Power Cables	Striveday™Flexible Silicone	1	$15.99	$15.99
20	1 Channel Relay	5v 1 channel Relay Module	2	$5.79	$11.58
21	2 Channel Relay	SunFounder 2 Channel DC	2	$6.79	$13.58
22	2 Amp Fuse	KO Fuse 2 Amp Inline Glass	2	$14.19	$28.38
23	Digital to Analog Convertor	HiLetgo MCP4725 12 Bit I2	2	$5.29	$10.58
24	Printed PCBs	JCLPCB -	5	$2.74	$18.52

MISCELLANEOUS

Part No.	Description	Model #	Quantity	Unit Cost (include taxes)	Total		
25	Magnets	Available at Superway					
26	H-bridges	Qunqi L298N Motor Drive (	3	$6.99	$20.97		
27	Resistor	Assorted Values	2	$7.99	$15.98		
28	Transistor	Transistor Kit 10 Value 200	1	$8.99	$8.99		
29	DC - DC Buck Convertor	Provided by Wayside Powe	2				
30	Clear Acrylic	Material for controller	1	$33.99	$33.99		
31	PCB	Custom made from Eagle b	1	$100	$100		
33	Limit Switches	URBESTAC 250V 5A SPDT 1	1	$8.99	$8.99		
34	Electrical tapes	Nova Supply's Pro Grade Bl	2	$14.49	$28.98		
35	Heat Shrink Tube	Ginsco 580 pcs 2:1 Heat Sh	1	$6.99	$6.99		
36	Breadboard Wires	ELEGOO 120pcs Multicolor	2	$6.98	$13.96		
37	Capacitors	OCR 24Value 500pcs Electr	1	$16.89	$16.89		
38	Diodes	Chanzon Fast Switching/Sc	1	$7.99	$7.99		
39	Heat Shrink Connectors	120 PCS Wirefy Heat Shrinl	1	$16.99	$16.99		
40	Wire Terminal Crimper	Titan Tools 11477 Ratchetir	1	$19.98	$19.98		
41	Alligator Clips	WGGE WG-026 10 Pieces a	1	$8.93	$8.93		
42	Heat Shrink Connectors	Marine Connectors, 285Pcs	1	$25.99	$25.99		
43	CONN PIN 20-24AWG CRIMI	Digi-Key	100	$0.17	$16.94		
44	CONN SOCKET 20-24AWG	Digi-Key	100	$0.13	$12.51		
45	CONN RECEPT 6POS 3MM	Digi-Key	25	$0.39	$9.86		
46	CONN PLUG 6POS 3MM D	Digi-Key	25	$0.39	$9.86		
						Total =	**$789.30**

Appendix C - Data Sheets for Components and Libraries

MCP4725 Datasheet:



Now that the header is attached, we can wire it up. We'll demonstrate using an Arduino.

FIrst, connect **VDD** (power) to a 3-5V power supply, and **GND** to ground.

The DAC uses I2C, a two-pin interface that can have up to 127 unique sensors attached (each must have a different **ADDRESS**).

- **SDA** to I2C Data (on the Uno, this is **A4** on the Mega it is **20** and on the Leonardo digital **2**)
- **SCL** to I2C Clock(on the Uno, this is **A5** on the Mega it is **21** and on the Leonardo digital **3**)

There's two other pins remaining.

- **A0** allow you to change the I2C address. By default (nothing attached to A0) the address is hex **0x62**. If A0 is connected to **VDD** the address is **0x63**. This lets you have two DAC boards connected to the same SDA/SCL I2C bus pins.
- **VOUT** is the voltage out from the DAC! The voltage will range from 0V (when the DAC value

Using with Arduino

Next up, download the Adafruit MCP4725 library. This library does all of the interfacing, so you can just "set and forget" the DAC output. It also has some examples to get you started

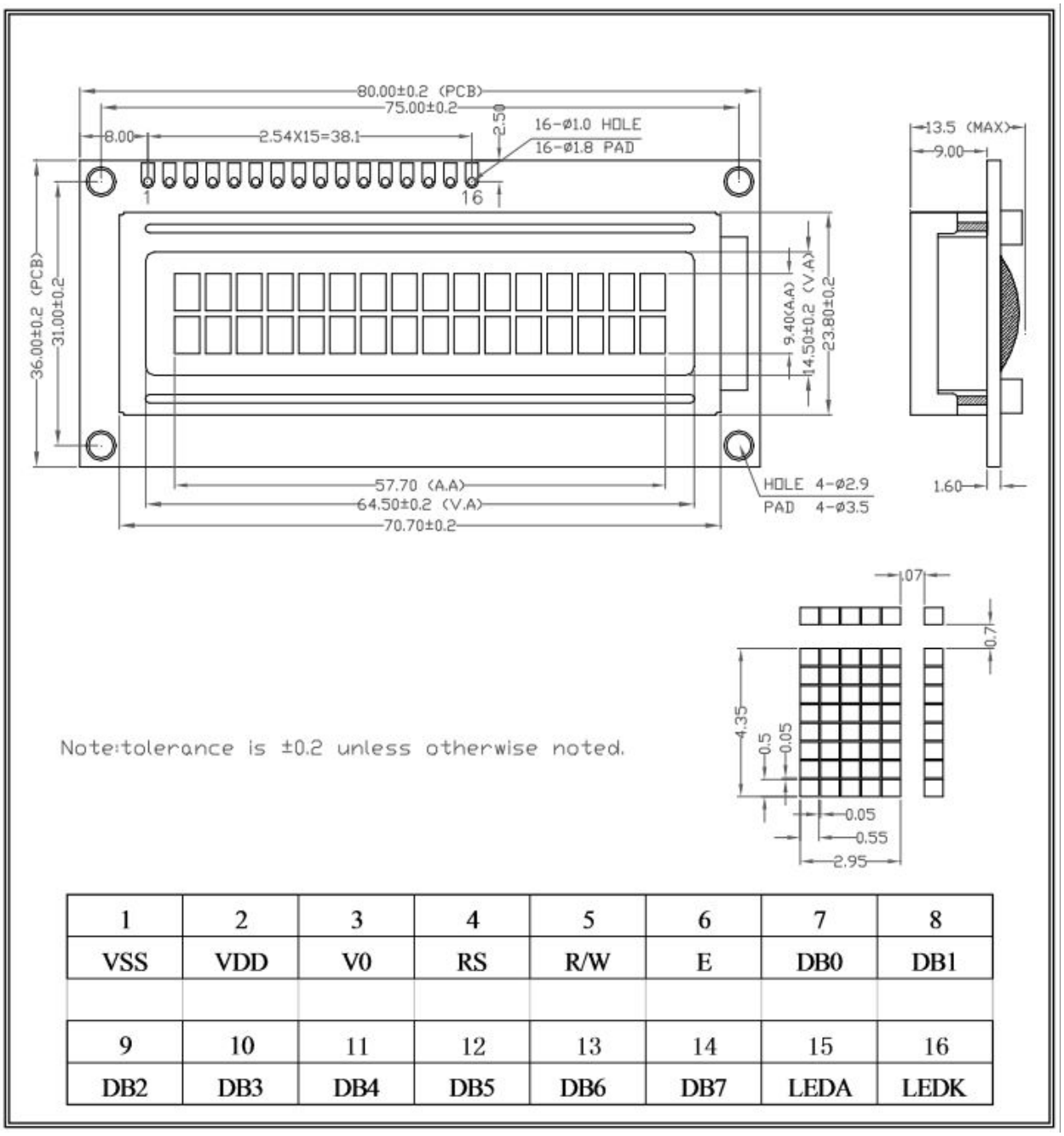
To download it, visit the GitHub repository and click the DOWNLOADS button in the top right corner (http://adafru.it/aPz), rename the uncompressed folder **Adafruit_MCP4725.** Check that the **Adafruit_MCP4725** folder
contains **Adafruit_MCP4725.cpp** and **Adafruit_MCP4725.h**

Place the **Adafruit_MCP4725** library folder your ***sketchbookfolder*/libraries/** folder. You may need to create the **libraries** subfolder if its your first library. You can figure out your ***sketchbookfolder*** by opening up the Preferences tab in the Arduino IDE.

Restart the IDE.

Open up the **File→Examples→Adafruit_MCP4725→trianglewave** sketch and upload it to the Arduino. Then connect your oscilloscope (or an LED + resistor if you don't have access to an oscilloscope)

LCD Screen Datasheet:



1	2	3	4	5	6	7	8
VSS	VDD	V0	RS	R/W	E	DB0	DB1
9	10	11	12	13	14	15	16
DB2	DB3	DB4	DB5	DB6	DB7	LEDA	LEDK

NRF24LO1+ Datasheet:

Absolute Maximum Ratings

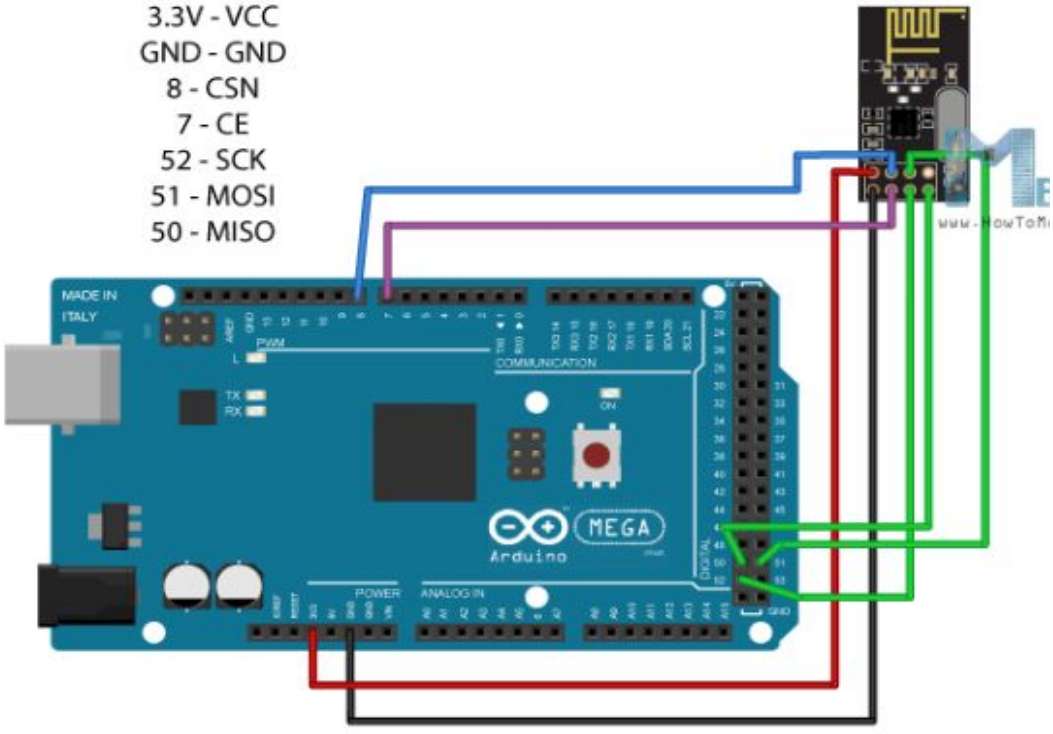
Supply voltages
VDD ..... - 0.3V to + 3.6V
VSS 0V
Input voltage
V_I ..... - 0.3V to 5.25V
Output voltage
V_O ..... VSS to VDD

Total Power Dissipation
P_D (T_A=85°C) 60mW

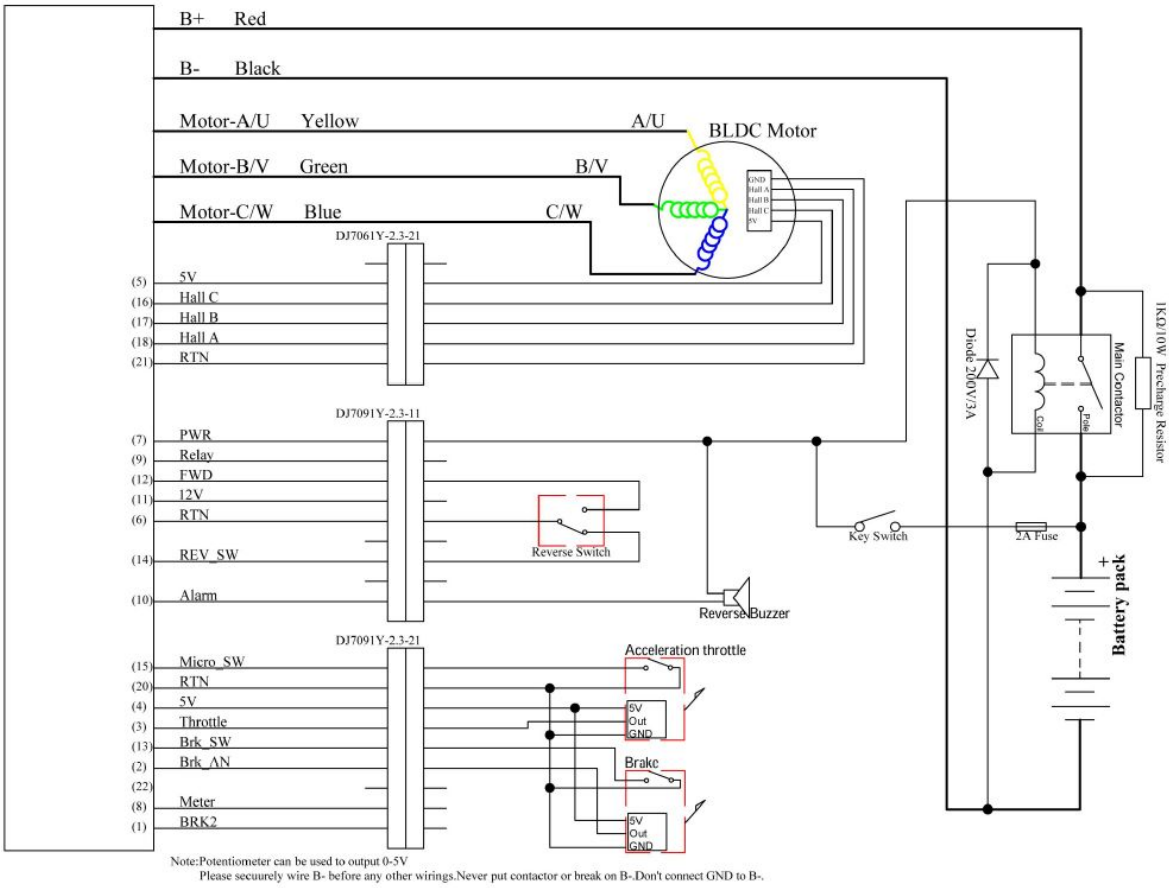
Temperatures
Operating Temperature…. - 40°C to + 85°C
Storage Temperature……. - 40°C to + 125°C

Arduino Mega - NRF24L01

3.3V - VCC
GND - GND
8 - CSN
7 - CE
52 - SCK
51 - MOSI
50 - MISO

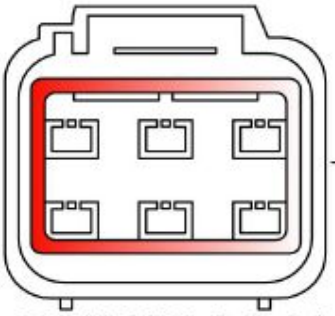


Motor Controller Datasheet:



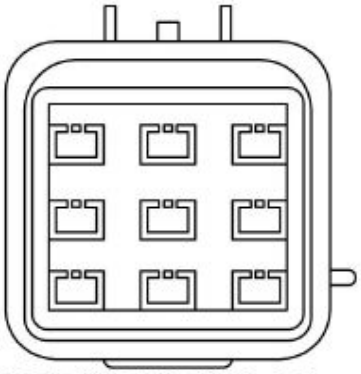
Motor Controller Pin Outs DataSheet:

3.2.1 Pin definition of KBS-X Controller



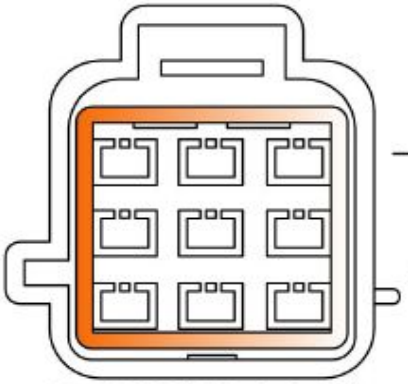
DJ7061Y-2.3-21

Black GND (21)		5V Purple (5)
Yellow Hall A (18)	D-Green Hall B (17)	D-Blue Hall C (16)



DJ7091Y-2.3-11

Orange REV_SW (14)	Black GND (6)	White FWD (12)
Red 12V (11)		Blue Relay (9)
	Pink PWR (7)	Yellow Alarm (10)



DJ7091Y-2.3-21

Gray Micro SW (15)	Dark Green Throttle (3)	Raddle BRK2 (1)
Black GND (20)	Dark Blue Meter (8)	Brown BRK_SW (13)
Purple 5V (4)	Dark Gray Brake AN (2)	(22)

Motor Controller Error Codes Datasheet:

Red LED Codes

LED Code		Explanation	Solution
1,2	¤ ¤¤	Over voltage error	1. Battery voltage is too high for the controller. Check battery volts and configuration. 2. Regeneration over-voltage. Controller will have cut back or stopped regen. 3. This only accurate to ± 2% upon Overvoltage setting.
1,3	¤ ¤¤¤	Low voltage error	1. The controller will clear after 5 seconds if battery volts returns to normal. 2. Check battery volts & recharge if required.
1,4	¤ ¤¤¤¤	Over temperature warning	1. Controller case temperature is above 90℃. Current will be limited. Reduce controller loading or switch Off until controller cools down. 2. Clean or improve heatsink or fan.
2,1	¤¤ ¤	Motor did not start	Motor did not reach 25 electrical RPM within 2 seconds of start-up. Hall sensor or phase wiring problem.
2,2	¤¤ ¤¤	Internal volts fault	1. Measure that B+ & PWR are correct when measured to B- or RTN. 2. There may be excessive load on the +5V supply caused by too low a value of Regen or throttle potentiometers or incorrect wiring. 3. Controller is damaged. Contact Kelly about a warranty repair.
2,3	¤¤ ¤¤¤	Over temperature	The controller temperature has exceeded 100℃. The controller will be stopped but will restart when temperature falls below 80℃.
2,4	¤¤ ¤¤¤¤	Throttle error at power-up	Throttle signal is higher than the preset 'dead zone' at Power On. Fault clears when throttle is released.
3,1	¤¤¤ ¤	Frequent reset	May be caused by over-voltage, bad motor intermittent earthing problem, bad wiring, etc.
3,2	¤¤¤ ¤¤	Internal reset	May be caused by some transient fault condition like a temporary over-current, momentarily high or low battery voltage. This can happen during normal operation.
3,3	¤¤¤ ¤¤¤	Hall throttle is open or short-circuit	When the throttle is repaired, a restart will clear the fault.
3,4	¤¤¤ ¤¤¤¤	Non-zero throttle on direction change	Controller won't allow a direction change unless the throttle or speed is at zero. Fault clears when throttle is released.
4,1	¤¤¤¤ ¤	Regen or Start-up over-voltage	Motor drive is disabled if an over-voltage is detected at start-up or during regen. The voltage threshold detection level is set during configuration.
4, 2	¤¤¤¤ ¤¤	Hall sensor error	1. Incorrect or loose wiring or a damaged hall sensor. 2. Also be caused by incorrect hall angle configuration (60 degree or 120 degree)
4, 3	¤¤¤¤ ¤¤¤	Motor over-temperature	Motor temperature has exceeded the configured maximum. The controller will shut down until the motor temperature cools down.

The Red LED flashes once at power on as a confidence check and then normally stays Off. "1, 2" means the Red flashes once and after a second pause, flashes twice. The pause time between multiple flash code groups is two seconds.

Component Libraries:

Adafruit MCP4725 library: https://github.com/adafruit/Adafruit_MCP4725
Arduino LCD library: https://github.com/arduino-libraries/LiquidCrystal