Team 21: Andrew Levine, Brandon Joel Gonzalez, Hyun Ji (Erin) Jung, Sarah Raizen
MEAM510 - Fall 2019 - Yim

Final Report

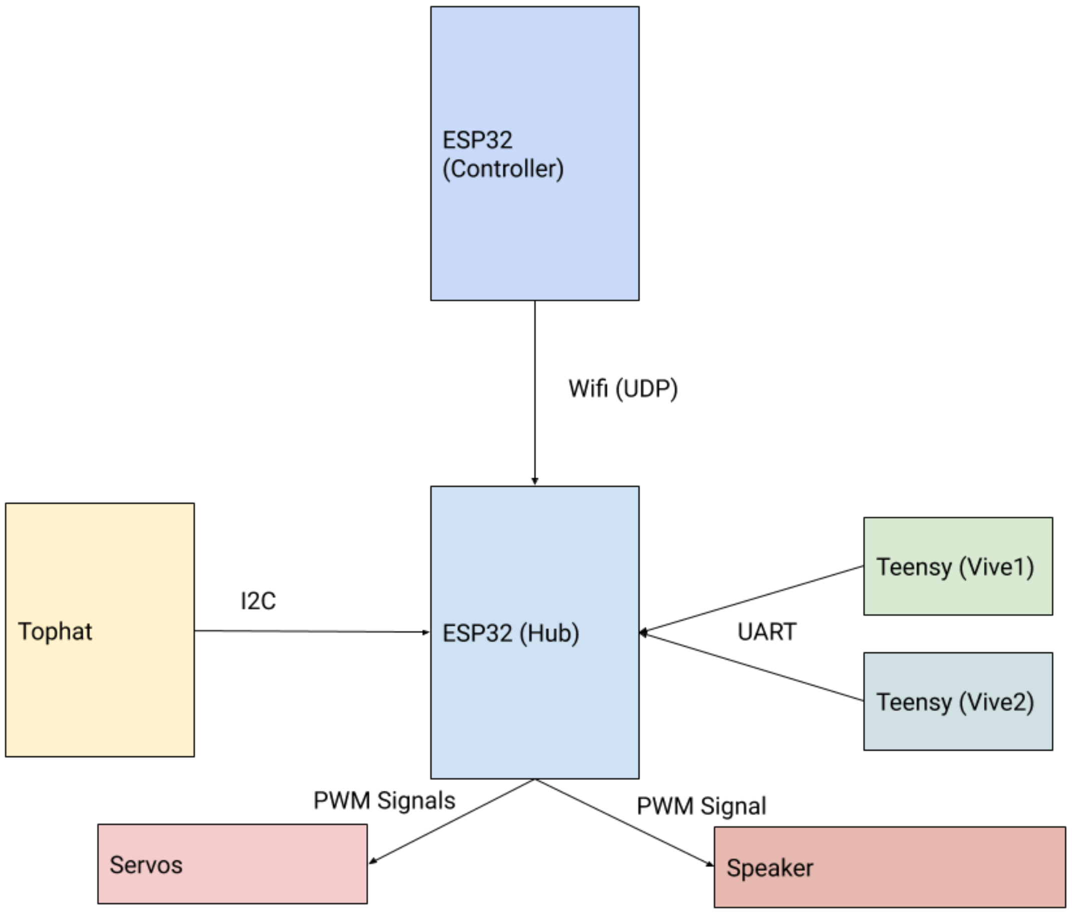
Game Strategy and Functional Overview

In the games, our robot focused on doing damage to the opposing team on the ground level. Our robot was relatively small and very nimble so it was able to deftly navigate around the swinging arms of death. We maintained superb control over the bot, with it responding timely and accurately to the commands sent from the controller. We did not attempt to scale the bridge because we had coordinated with other members of our meta-team and they were given that area to focus on. However, given our 4-wheel drive and sturdy construction, our robot would have been able to fulfill that role as well.

For our minimum functionality requirements we designed the robot with 4-wheel differential drive. The wheels were controlled by continuous servos. The attack arm was controlled by a non-continuous servo. The attack arm was designed so that it could fold backwards at the start of the game in order to maintain the size limitations on the robots. The arm was designed to have three eyes at the end to remain consistent with our theme relating to an alien. The width of the eyes was also strategically advantageous as it helped us reach opponents' whiskers. The robot was WiFi controlled with a controller. The controller featured 2 joysticks each featured a potentiometer for the up-down motion and the left-right motion. The up-down direction was used to control the speed of the wheels. The left-right direction on one of joysticks was used to control the attack arm.

The top hat was fully integrated with our system. The robot would stop when health was depleted. The whisker switch was embedded securely in the acrylic layers of the body of the robot. The top hat LED was secured to the top of the robot in order to display the health and alive status. The information received on the top hat was communicated to our central ESP32 using I2C communication. Additionally, because we are a team of 4, we included sound in our design. The speaker played 4 tones: an attack tone, a damaged tone, a death tone, and a respawn tone.

Functional Block Diagram



Mechanical Design Overview

The mechanical design was essentially four levels cut out of acrylic connected by standoffs. All components were attached via mechanical fasteners such as screws, nuts, and standoffs, except for the batteries and the photodiodes which were attached with tape. Each servo motor was attached from the bottom of the chassis through a piece that interfaced with the chassis and then was screwed to the servos. Each level of the robot featured holes that were dimensioned so that protoboards could be attached to them. In total, we had 5 protoboards within the robot. As we wanted to design our robot to look like an alien, the upper levels were cut out of green acrylic and the bottom level out blue. The attack arm was likewise cut out of green acrylic and featured three googly eyes at the end. Overall our design was robust but also retained the capability to be taken apart to make changes.

As expected, the mechanical design went through many changes as the project progressed. One of the challenges faced during the mechanical construction was fitting everything we needed within the size constraints. The first full assembly of the robot took a couple iterations as we modified the heights of the standoffs in order to ensure everything could fit and remain under 6 in. Another issue related to the photodiodes and the Vive sensing circuits. We initially wanted to secure them to the top of the robot so that they would be able to easily read the Vive signal and remain fastened securely in place. However this changed once we realized that the top hat LED was required to be mounted on the top. We then moved the circuits to a lower level but due to the obstructions, the signals received by the photodiodes were very poor. We resolved to desolder the photodiodes from their circuits and attach them to wires so that they could be featured on the top of the robot and receive signals.

The wheels were a large concern for the robot, as the wheels of our car fell off of the servos easily, despite their press fit. Therefore, the wheels were secured with screws from the outside to the servo, to prevent disassembly. Also, four wheels were chosen in order to provide the robot stability. With a four wheel differential drive design, the wheels had to be placed equidistant apart so that the robot would be able to turn in all directions.

Electrical Design Overview

When approaching the electrical design of the robot, one thing the team wanted to ensure was the robustness of the circuitry. The robots were going to be smashed a lot and therefore, every circuit was soldered to its own protoboard and all of the connections between boards and batteries were made using molex connectors.

The Vive circuits were very difficult circuits to design. Originally, the circuit filtered out the noise from the photodiode before entering the buffer. However, the filter circuit had an effect on the signal coming into the buffer. We decided to first pass the photodiode signal through the buffer, then filter out the noise from the ambient light. In order to determine the right resistor, a potentiometer was placed in the circuit and adjusted with a constant capacitance to see which resistance best removed the noise. Once the optimal RC value was found, we noticed that as the capacitance in the RC value is increased and the resistance is lowered, the filter works even better. Originally, a capacitance of 10 picoFarads was being used with a MegaOhm resistor. However, increasing the capacitance to 10 microFarads and using a 100 ohm resistor, despite having the same RC value, filtered out significantly more noise. Then, once the noise was filtered out, the signal had been attenuated. In order to account for the attenuation, a non-inverting op-amp configuration was used to amplify the signal with a gain of 10.

Once the Vive circuit was reading correct values, we had to connect the teensies to the ESP32 to communicate the data. We decided to use UART communication because it was the simplest to enact. However, UART only sends one byte at a time, and often the bytes are either left out or duplicated. Therefore, in order to ensure the accuracy of the information being transmitted, the bytes were sent as a packet, using beginning and end bytes with a message exactly 4 bytes long. Once the ESP32 received a start byte, it would begin to read the message. But if an end byte never came, or if the message was longer than 4 bytes, the ESP32 would disregard the packet.

After figuring out the UART communication, we needed to find a way to power the teensies. We decided to use one 9V battery source for both teensies and connect the grounds of the teensies directly. However, when the grounds of the teensies were connected, for some reason, it affected the second teensies ability for UART communication with the ESP32 and it would stop sending messages. Therefore, we decided to use two separate 9V batteries to power the teensies individually. We used 9V batteries because the teensies do not draw a lot of current and therefore would not deplete the voltage of the 9V battery, and the batteries were incredibly light and barely added to the robot's overall weight.

When it came to the motors of the robot, we chose continuous servos because they are incredibly easy to control the speed and only require 5 V to operate. The motors were supplied voltage via two single cell Lipos in series. We needed to use Lipos because our car had 5 servos; 4 for each wheel and 1 for the attack arm, and we needed a power supply that had a large power output. Originally, we wanted to use one two cell Lipo to power the servos, but when trying to add a molex connector, the two cell Lipo exploded and all we could find in time were two single cell Lipos. There were two very large issues when it came to the grounds of the power supplies. The first problem was that the ground of the servos had to be as distantly connected to ground of the photodiodes as possible. Noise from the servos would disrupt the photodiode's signal. Therefore, the grounds of the servo were connected to the voltage regulators, that were connected to the ground of the ESP32, which connected to the ground of the Teensy, which connected to the ground of the photodiode. 4 components between the two grounds turned out to be enough to prevent the photodiode from seeing any of the servos' noise. The second issue with the power supply grounds was that they were not originally connected to the ESP32 ground. Without the connection between the two grounds, the servos had no relative voltage for the PWM signals they were receiving from the ESP32 and therefore would not spin. This issue was corrected by simply connecting the grounds via the voltage regulator. The voltage regulators were incredibly important to the circuit because they served as a circuit breaker for the circuit from the power supply. There was one occasion when the power supply was plugged in backwards and the voltage regulator exploded, but the explosion saved the servos and the ESP32 from shorting. Finally, when it came to noise prevention from the servos, a 47 microFarad capacitor was placed across the power supply.

When it came to powering the ESP32, we decided to use a 4 AA battery pack in series with a 2 AA battery pack providing the ESP32 with 9V. Originally, we were using 2 AA battery packs in series, but 12 V ended up being too much for the ESP32 and it would overheat, even though the data sheet said that the ESP32 could handle up to 12V. We decided to use AA battery packs instead of 9V batteries because we used the same input power supply to power the top hat. Turns out that the top hat drew a significant amount of current, and the current would cause the 9V battery to drop beneath the 6V required to power the tophat. We used a different power supply for the ESP32 and the tophat than the servos because voltage spikes from the servo could affect the performance of the ESP32 and the top hat and we wanted to prevent that interference.

Then, for our peripheral, we decided to incorporate a speaker into our system. The noises were controlled by sending different frequency PWMs to a MOSFET that controlled the power being supplied to the speaker. That would allow us to use the ESP32 to control the tones that the speaker made without having to store incredibly large audio files. The speaker circuit was not incredibly complicated and was the first time that something pretty much worked right away during this project.

The controller circuit was also very simple. Analog pins A5, A4, A3 were used on the ESP32 as those are the pins that work when WiFi is being used. We used joysticks because they were very simple to move with your thumbs, and very simple to read analog values from. They are made up of two potentiometers each, and the values just go from 0 to 4095 based on how much the joystick has been moved. Each joystick controlled the servos on the corresponding side of the car by moving them up and down, while moving the right joystick to the right or left controlled the swinging of the attack arm. The controller was powered via usb connection since it was the most reliable and the joystick could be stationary near a computer during the competition.

In order to control the robot, they communicated via WiFi, where the controller was the access point and the robot was the station. A problem that we encountered with the ESP32 WiFi connection was that if code was uploaded to the ESP32 on the robot, then the robot would stop communicating with the remote controller. We discovered that when new code was flashed to the robot, the robot's WiFi connection was lost. Therefore, the controller would then have to reboot, and the robot would have to reboot again in order to establish the proper connection.

Overall, the circuitry was designed to be robust. Therefore, every circuit was soldered and every connection used a molex connector. Also, each circuit was color coded in order to ease the debugging process. The overall architecture was very careful to separate the power supplies of the ESP32 and the servos, and to carefully arrange the grounds as to prevent interference, while still providing the common connection between all of the components. Finally, the robot was designed to use the lightest power sources available that provided the necessary power for the component it was powering to reduce weight. There were many problems when it came to the overall integration of the circuits, but in the end it was done and a lot was learned along the way.

Software Design Overview

We divided our code into three separate files. We have a *main.c* file that runs on the two Teensy boards for the Vive. We then have a *Controller.ino* file that runs on an ESP32, connected to a laptop, that is used to control the robot and print out debug information as needed. Lastly we have a *Robot.ino* file, which used the given *DemoBot.ino* as a skeleton to build upon for the main code that's running on the robot.

The *main.c* file is used to read Vive data. We tried to implement the same functionality for reading positional data from a Vive as from Lab 2 using the input capture functionality of the Teensy. We loaded this file onto two Teensy boards so that we could get positional data from two locations on our robot, the front and the back. We then sent this data over UART to the ESP32, described later on.

The *Controller.ino* file is relatively simple. WiFi communication is initiated. We read from three analog pins to obtain relevant joystick information for driving servos and the attack servo. The analog readings are mapped to the corresponding positions for the servos. Finally a packet is sent over WiFi containing the information to write to the servos.

The *Robot.ino* file is the longest of the code files used during this project. It contains sections for: initializing global variables, I2C communication, interrupts, top hat LED control, controlling the servos, autonomy, sound, setup of UDP WiFi, UART for the two Teensy boards, and the top hat.

In order to control the speed and direction of our robot, we were able to use the *ESP32servo.h* library because we used continuous servos. This made it relatively simple to write positions, or speeds, to our motors. Originally we were not aware of this library and we used *ledcWrite()* similar to lab 4. However, this caused some issues early on and once we discovered the *ESP32servo.h* library, the code became much more streamlined.

For communications, our robot interacted with other boards through a few different protocols. To talk to the controller, we used UDP to listen for packets, much like in Lab 4. To talk to the tophat, we used I2C communication, relying on the code given in the *DemoBot.ino* source which went unchanged. The display of the tophat LED was also based on the *DemoBot.ino* code and went unchanged. Finally, to talk to the two Teensy boards, we used UART, where we setup a protocol for transmission of the data. We had a 6-byte message, with a start and end byte and then the positional data split across the 4 intermediate bytes (the first two for the X position and the last two for the Y position). This way, our ESP32 would listen to the two serial lines and use that to figure out what positional data we were getting from the Teensy boards.

To implement sound, we connected a pin to the gate of a MOSFET. Then, we used *ledcWriteTone()* and *ledcWrite()* to the pin in order to play a sound at a specific frequency. We predefined four different frequencies for four events - spawn, attack, damaged, death - that were each called at different times. To make sure that a sound only played for a certain amount of time, we got a timestamp of when we started playing a sound using *millis()*. Then, after however many seconds passed, we would shut off that pin so that the sound would start. To make sure that sounds only played at appropriate times, we had to use a variety of different flags in the main loop which got a bit messy (those flags are denoted as “sound variables” in *loop()*). However, after a bit of fiddling with these variables, we seemed to get sounds that played at the right times for the right length!

Lastly, although we did not finish implementing autonomous mode to get the check off, we did attempt to. Our strategy was originally going to rely on both Vive sensing circuits we had completed. This would allow us to gain orientation data. However, we struggled to get reliable readings from one of them so we focused on implementing autonomy using one sensor. This meant that we the first step would be orienting the robot within the playing field. We achieved this by telling the robot to drive for a couple seconds and then determine the angle that the robot was oriented by subtracting the first from the second position. Then, using the angle and the target position we would then tell the robot to turn so that it was facing the target position. Then we would tell it to move forward until it reached the target position. This method was achieved using some geometry theorems to calculate the angles. We were not able to get the autonomy fully operational and it only worked from certain places on the field. We repeatedly faced inconsistent readings from the Vive sensors which made it difficult to implement a robust system. Attempts to get the robot to navigate from a random location to a specific point on the field were only occasionally successful.

Video

Click here for a video of the robot.

Epilogue and Retrospective

Andrew

I think I have learned more in this class than most of my classes combined. I have always had an incredible interest in electronics and software integration, but none of the classes in the mechanical engineering major touch upon those subjects. One of the lessons that really stood out to me was the importance of compartmentalizing and planning. By dividing the circuit up into completely separated parts, debugging certain problems became significantly easier. When there was a problem with the servos, we knew exactly which part of the circuit to look at. By color coding the circuits and laying out the parts so that nothing overlapped, we were able to trace the circuits much easier and figure out what the problems were. Also, nothing is more satisfying than ending up with a beautifully soldered circuit.

Another important lesson was the arrangement of the grounds. I always thought of ground as this sink for voltage, but that all grounds could be connected in any way and it would not matter. However, if there is one thing that integrating circuits has taught me, it is that the grounds need to be laid out in a very specific way. Certain components generate so much noise and that noise can be transferred via the ground.

I have also coded in many other classes before mechatronics and in actual jobs, but I never understand low level coding until this class. Mechatronics forced me to learn about bits and bytes and how they can be transferred and communicated. Coming up with a system for sending proper UART packages via singular byte transfers was an incredibly valuable experience for me. Especially coding the ESP32 to read from two alternating UARTs was very difficult, but important.

I think an important strategy that we overlooked in preparation for the actual competition was checking in on our teammates. In almost every single battle, our robot was the only robot operational on our team. We should have been checking in on the other teams during the last week to make sure that everyone was going to finish on time. Otherwise, during the gameplay, we might not have been crushed by teams who had more than one working robot.

Also, we probably should have started building the robot earlier than we did. I only started heavily constructing the robot two weeks before the competition. I think if I had started a week earlier, I would have had time to fully hash out the autonomy for our robot.

We should have also done more research into the continuous servos before using them. The servos were very simple to operate, but we wasted a significant amount of time debugging them, due to the failure on our part to connect the ground of the servos with the ESP32. If more time had gone into researching the part beforehand, that would have saved a significant amount of debugging time during the integration.

Brandon

I had a ton of fun in this project! I don't think there's really anything like it in another course at Penn. Getting the chance to build your own robot, taking control of the mechanical, electrical, and programming components as you see fit, is very exciting and incredibly fulfilling. I was fortunate to be working with talented and motivated teammates who used all their skills to help us pull it all together.

I learned a lot from working with them, whether it was the basics of mechanical design to some of the intricacies in designing our circuit boards properly. I feel like I got a much better sense of applied electronics with this project. Lots of ESE classes teach you the basics and fundamentals of circuits, but this is one of the only courses where you really apply that knowledge to real, moving systems. Furthermore, this is the only project I've done where there was a *real* possibility of breaking your own stuff and causing huge setbacks for your team. This project taught me very good safety protocols in lab work - I always made sure to work on this project with at least one teammate by my side so we could check each other before doing something stupid. To that end, the project helped my debugging skills immensely and had me checking circuits ten times over before plugging a battery in and blowing something up, which is always good practice when working with any electronics. In sum, I felt like I grew as an aspiring electrical engineer as a result of this project.

The actual game itself was also a ton of fun - when we could successfully move our robot. The tower defense game was a bit confusing to grasp but I think that was just because I was bad at reading the rules. I think that, when all four robots on each team were fully functional, the game was very exciting! Unfortunately, it was pretty rare to see all robots active on the field at once, but that's just the nature of mechatronics as each team runs into complications.

The one criticism I do have of the game, which I also wrote in the feedback survey, was the autonomous mode. I feel that having the autonomous mode didn't feel too appropriate for this project and for this class. You could teach an entire course dedicated to autonomy in robotics and vehicles, and we have several of them offered to my knowledge. The way it was implemented in this project ended up being quite frustrating, difficult, and tedious to get working, and it seems like the majority of teams gave up on trying to get it to work. I heard a while back that each robot was going to get cameras to mount in order to only see the field via our robot's "eyes", but that never came to fruition. I think that would have been an incredibly fun way to play the game, more akin to how you might build and control a robot in some foreign environment, like a cave, that humans can't easily explore. I know that streaming the data from the camera to a laptop for each robot might be complicated, but I think that could be a much more fun and informative way to add a twist to the project.

Other than that, I learned so much from this project and had lots of fun doing it. Thanks for the memories!

Erin

This was my first time designing and building a battery-powered system from scratch, and I learned so much from it. Since more ESE labs are run with ideal power supplies, I hadn't had a chance to really think about how battery operated circuits would run. Through this project, I learned about ground noises and how they can affect the reading of sensitive sensors; how unstable power supplies can invisibly cause chips to frequently reset halt execution; and most importantly, how electrical designs are integrated with the corresponding mechanical designs. Debugging the system was also a valuable experience, as I had to consider all possible sources of a bug from disconnected wires to commented out code in the microcontroller.

While our robot performed relatively well during the tournament, I wish everyone had more time to focus on solidifying the basic functions of their robots -- it was disappointing to see only a couple of robots moving in each round. If we could focus just on driving the robot remotely and reliably, the entire class would have walked away with a better understanding of what makes a circuit work and what does not. Nonetheless, this was a valuable class that really strengthened my understanding of digital circuits and low level programming.

Sarah

Overall I am pleased with the performance of our robot. Although we didn't perform well as a meta-team, our robot still did well. In the three rounds we played, our robot was able to capture a tower twice and do damage to the enemy nexus twice. I think that the level of control we had over our robot was impressive compared to some of the others I saw. I also think our choice to use 4 wheel differential drive resulted in effective 0 radius tuning and a sturdy structure. I wish we had studied the game strategy a bit more before playing and it also would have been nice to have other functional robots on our meta-team to support us during the single-elimination round.

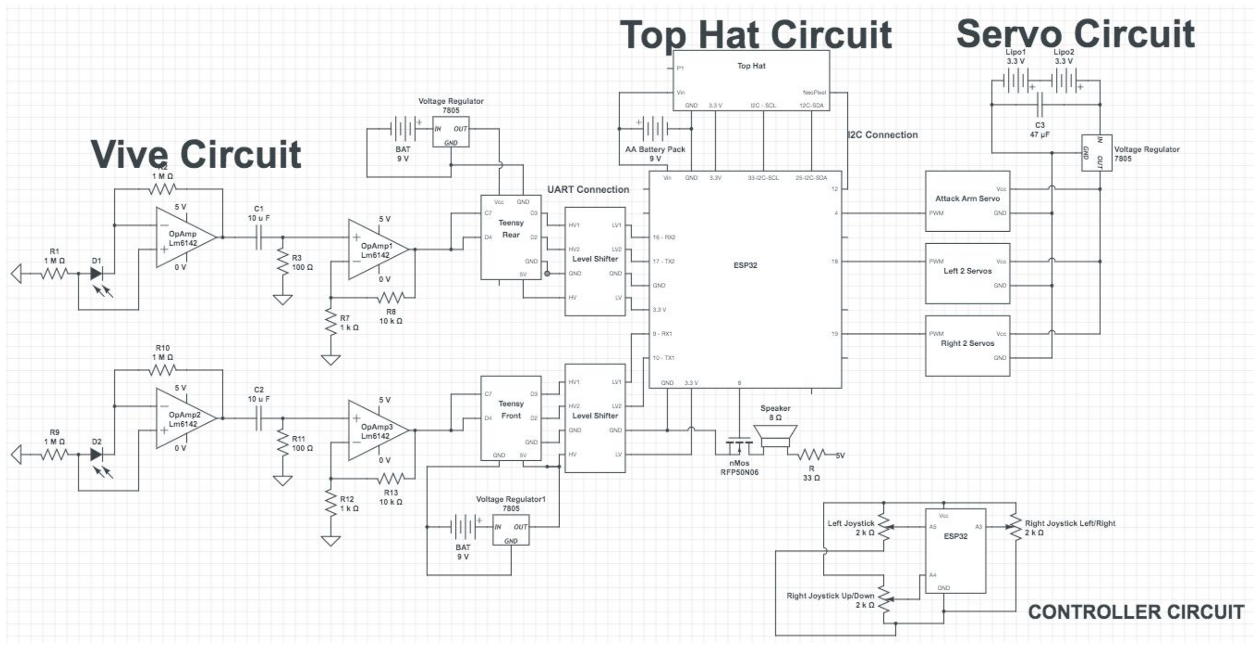
Over the course of the class, I think the most important thing I learned was how to effectively debug circuits and more generally how to persevere in the face of errors that appear to not make sense. The best part of the class is that it is extremely project-based. I enjoy working towards creating a tangible product and focusing on functional results. I struggled the most in the course with the programming. The only computing language I knew before this semester was MATLAB. I felt that much of the content about data storage and computer architecture went over my head but I still appreciated the exposure to it. Although the beginning of the semester was especially challenging, I enjoyed the opportunity to learn a new language and implement it throughout the semester.

Appendix

Code listings

See software design section for description of code. See Canvas upload for code files themselves.

Circuit diagrams



BOM

Line #	Part Name				Part Number	Qty	Source	Cost	Notes and references
1	Mechatronics Battle Bot: Jeremy					1			
2									
3		Top Hat S/A				1			
4			Whisker Switch		1	1			provided by TAs
5			LED Display		2	1			provided by TAs

6			Protoboard		3	1	GM lab		
7			Voltage Regulator		4	1	GM lab		
8			ESP32		5	1			
9			AA battery		6	4			
10			Speaker		7	1	Adafruit	1.950	
11		Driving S/A				1			
12			Continuous Servos		8	4	Adafruit	11.950	
13			Wheels		9	4	Adafruit	2.950	
14			Protoboard		3	1	GM Lab		
15			ESP32		5	1			
16			LiPo Battery		10	1			provided by TAs
17									
18		Autonomous Mode S/A				1			
19			Photodiode		11	2	GM lab		reuse from previous lab
20			Voltage Regulator		4	2	GM lab		
21			Teensy		12	1			
22					13	1			
23		Mechanical Housing S/A				1			
24			1/8" Acrylic Sheet		14	2	RPL		
25			1" Stand Offs		15	12	GM lab		
26			1/2" Stand Offs		16	8	GM lab		
27			4-40 3/8" Screws		17	8	GM lab		
28			4-40 1/4" Screws		18	1	GM lab		
29			4-40 Bolts		19	8	GM lab		
30			battery holder		20	4	Digikey	1.210	
			3D printed case		21	8	RPL		

31		Remote S/A				1			
32			ESP32		5	1			
33			Joystick		22	2	BE lab		
34			Protoboard		3	2	GM Lab		
35			Portable Battery		23	1			personally own
36			4-40 1/4" Screws		18	8	GM lab		
37			4-40 3/8" Screws		17	16	GM lab		
38			1" Standoffs		15	4	GM lab		
39			1/4" Standoffs		24	8	GM lab		
			1/8" MDF Sheet		25	1	RPL		
		Attack Arm S/A							
			Servo Motor		26	1			reuse from Waldo Lab
			4-40 3/8" Screws		17	2	GM Lab		
40			Attack Arm		27	1	RPL		
41		MATERIAL COST TOTAL							
42									
43		PRODUCT ASSEMBLY				1			
44			Top Hat S/A			1		1.95	
45			Driving S/A			1		59.6	
46			Autonomous Mode S/A			1		0	
47			Mechanical Housing S/A			1		4.84	
			Remote S/A			1		0	
48			Attack Arm S/A			1		0	

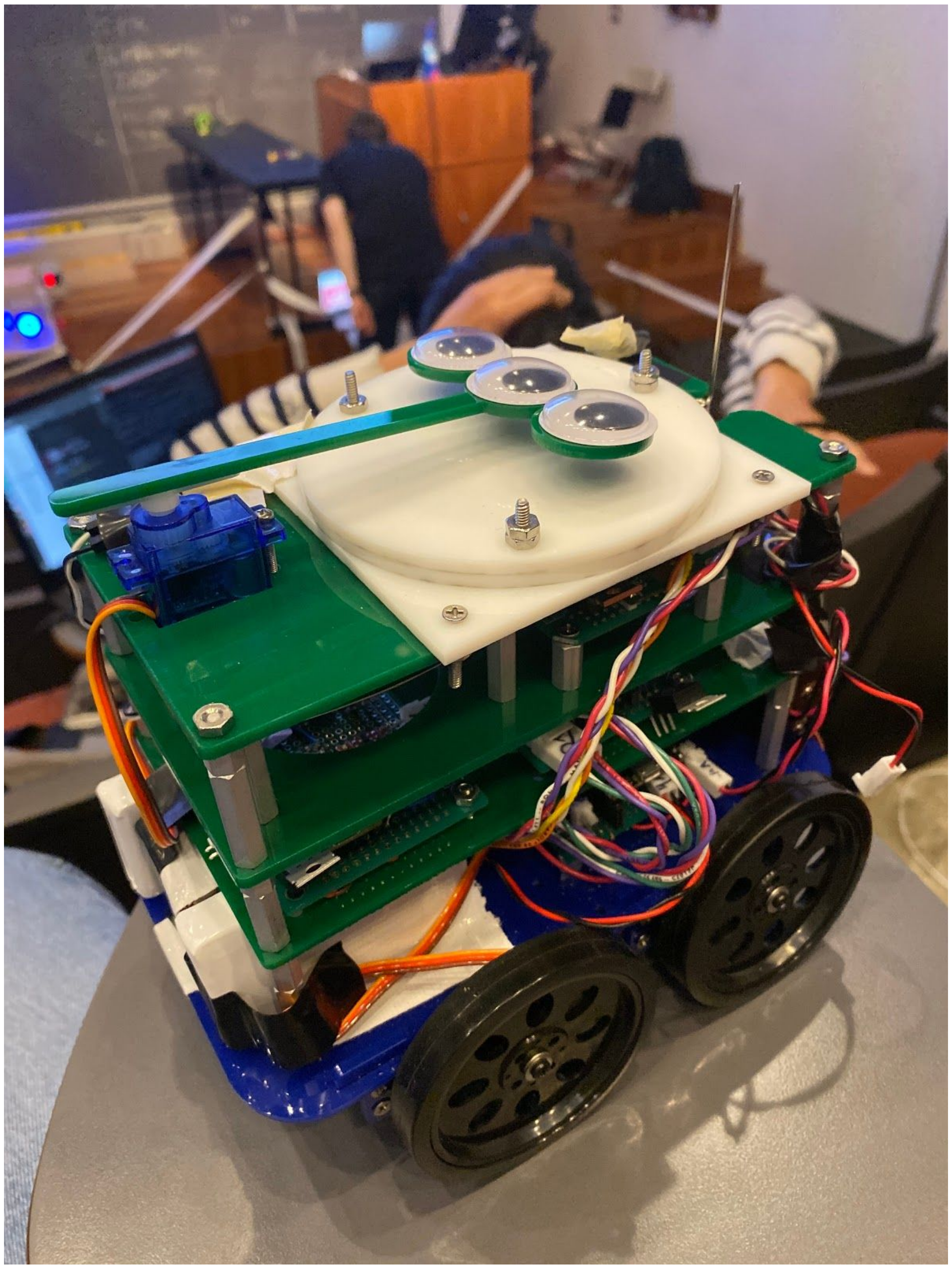
49		PRODUCT ASSEMBLY TOTAL						66.71	
50									
51	TOTALS								

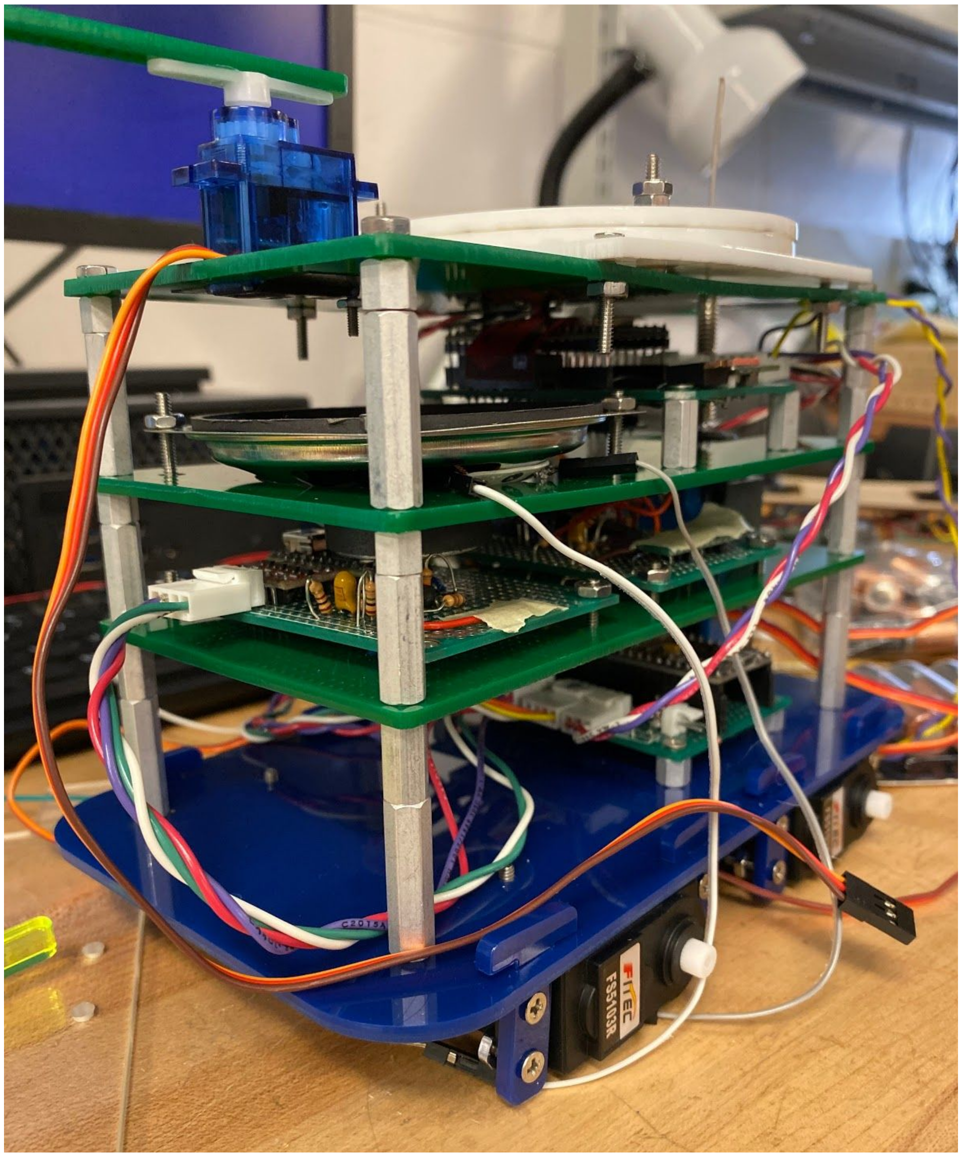
Data Sheets

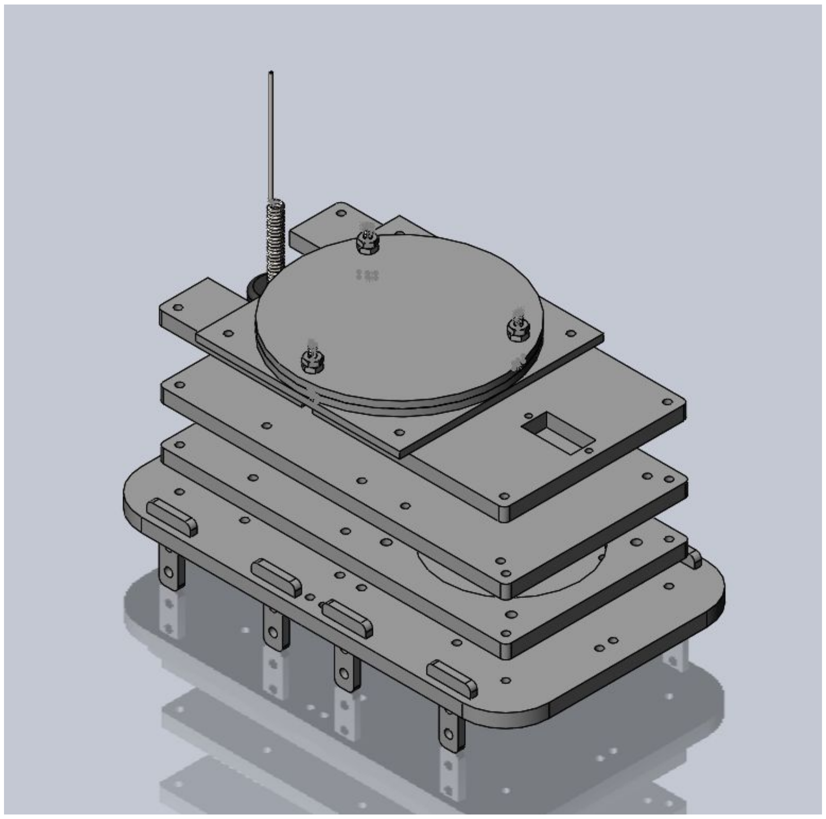
Links to ordered parts:

- Continuous servos
    - https://media.digikey.com/pdf/Data%20Sheets/Adafruit%20PDFs/154_Web.pdf
- Wheels
    - https://cdn-shop.adafruit.com/product-files/167/167.pdf
- Speaker
- Battery holder

Photos/renderings of robot

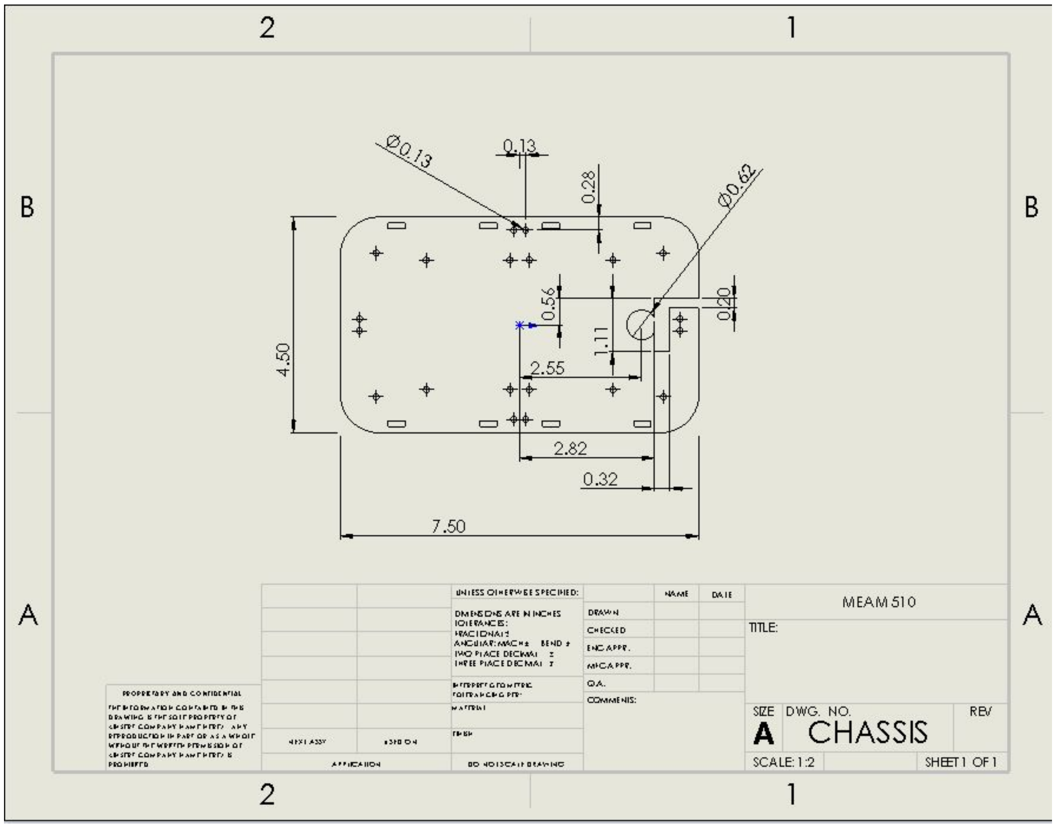

Our beautiful boy, Jeremy

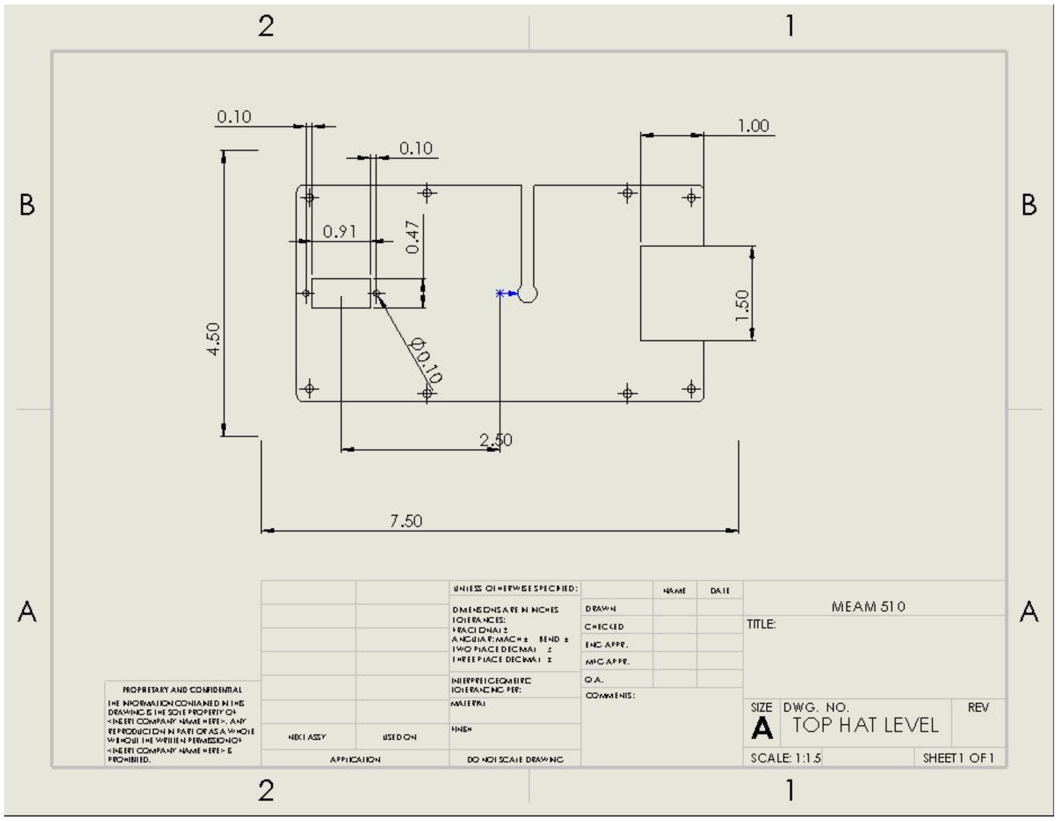

Jeremy, work-in-progress (wheels and batteries missing)


Solidworks rendering of Jeremy

CAD Drawings

The CAD drawings below show the bottom level, or chassis, and the top level. The chassis features the interface with the motors, space to attach 2 protoboards (although we only ended up using one), and a slot to put the whisker switch. The top level features a place to secure the attack arm servo, the top hat LED, and space for the whisker switch. Both boards also include holes for standoffs. The other upper levels include the same outer dimensions as the top hat level but with different holes corresponding to what functionalities were attached to them.


Ø0.13
0.13
0.28
Ø0.62
0.56
0.20
1.11
4.50
2.55
2.82
0.32
7.50
MEAM 510
TITLE:
SIZE A
DWG. NO. CHASSIS
REV
SCALE: 1:2
SHEET 1 OF 1


0.10
0.10
1.00
0.91
0.47
4.50
Ø0.10
1.50
2.50
7.50
MEAM 510
TITLE:
SIZE A
DWG. NO. TOP HAT LEVEL
REV
SCALE: 1:1.5
SHEET 1 OF 1