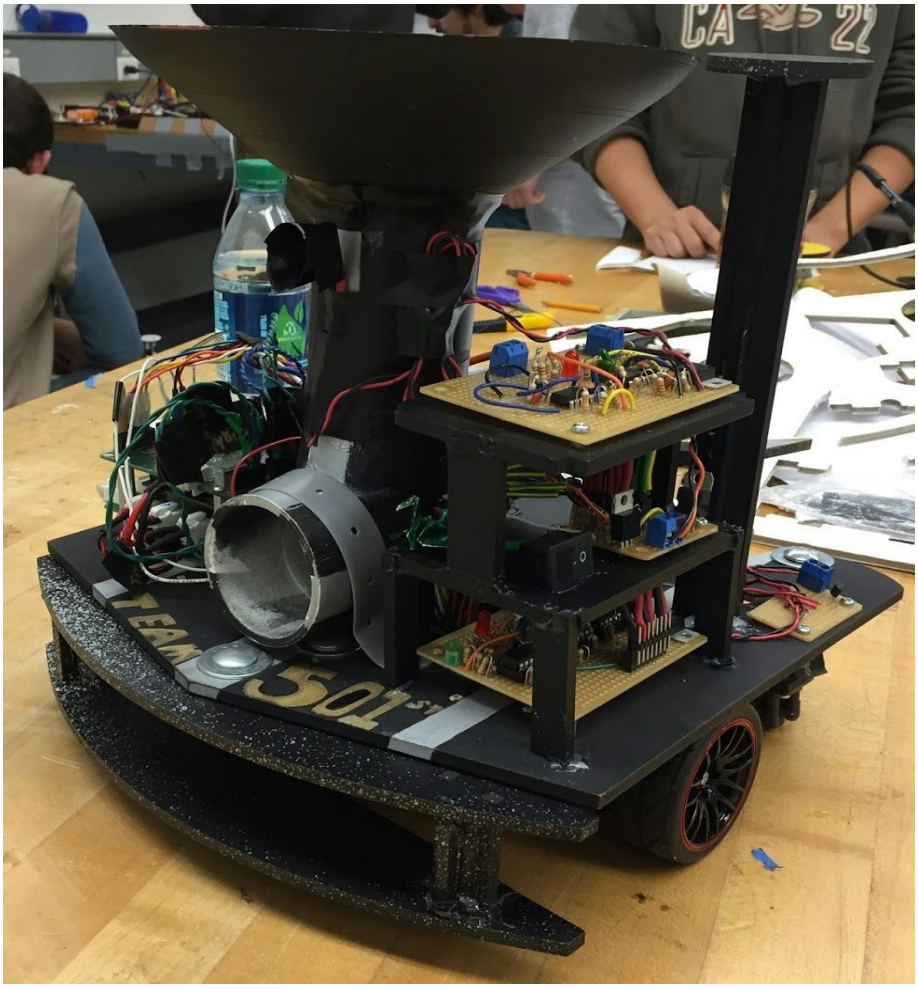
CMPE 118 Final Report

Team 501st



Team members:

Victor Ardulov
Josh Gutterman
Ian Zentner

Introduction:

Over the course of the previous 10 weeks, we spent our first four familiarizing ourselves with different aspects and components of mechatronic design. This ranged from programming state-machines, to designing, testing and building electronics, as well as utilizing tools like SolidWorks to design and cut mechanical components. In the end of these laboratory assignments we had gained insight and skills that allowed us to create these individual components.

Following these misadventures, we were given a task to build a robot that would load ammo (ping-pong balls), find its opponent using a 2kHz IR beacon, fire said loaded ammo at our opponent and finally find a hyper portal (24-26kHz track wire) and park our bot in that space.

While we had experience in each of these field independently we had not yet designed and implemented a complete system. Furthermore, the three of us had ever worked on a team together before. So below is a description of our design concepts, implementation and results. We express our choices, reasoning, challenges, and overall success.

In the end we demonstrate the shortcomings as well as the overwhelming accomplishment achieved, as well as important insights that we all earned and gained from this project as a whole.

Solidworks + Prototype + Design:

For our design process we began by sitting down and assessing the constraints and overall goals we had for the design. The constraints that were set up were set up such that our final bot could not exceed and 11in X 11in X 11in cube. Also there was a need to be able to support a beacon, that had to be 11 in off the ground, and most importantly nothing could obstruct the beacon.

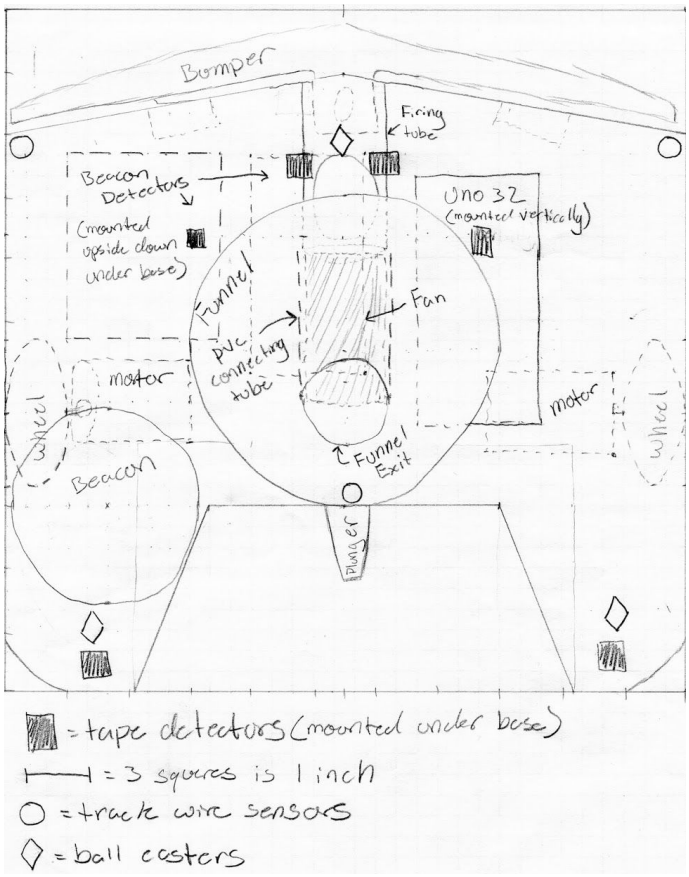
Next we evaluated the design of our bot as a tool to accomplish the tasks. Because we knew that our movements could prove to be imprecise, we designed a backup plunger that allowed us to back with a wide angle and still align with the ammo tower. We also shaped the bumper in a way that made sure that it wouldn't get stuck in the ammo tower itself, this motivated a wider, rounded end on the plunger.

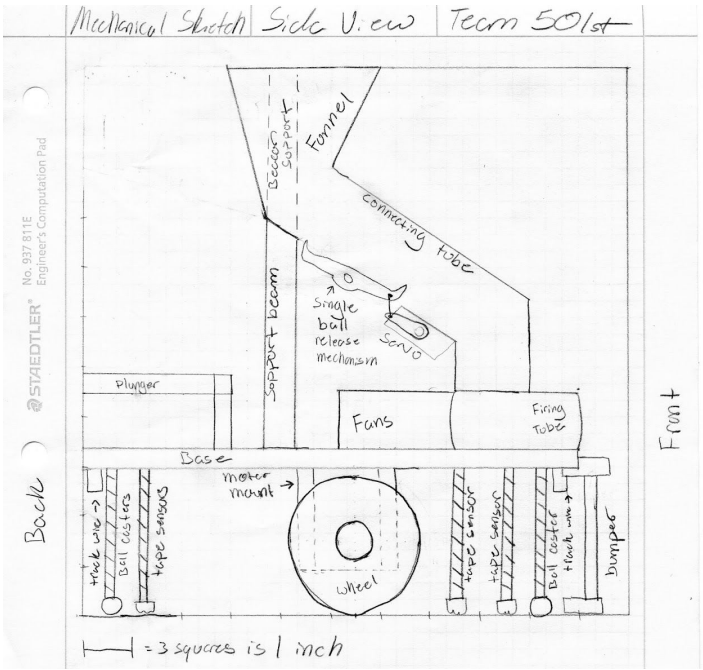
For the shape of the base, we assumed that we would want to maximized volume so that we would have as much real-estate as we could physically achieve. We shaped our base to make room for the plunger in the back. In the front we made a rounded bumper that was suspended from the base.so the font of the base given a traingular shape. We then rounded all of the corners on the base to make sure it would not catch on the corners.

We decided on placing all of the electronics on the top of the base for easy access, devising a shelf where all of the perforated boards lived. This made it modular and easy to find identify non working electronics and fix them quickly. This also made wiring the sensors into the Uno stack very simple and straightforward.

Sketches and Diagrams:

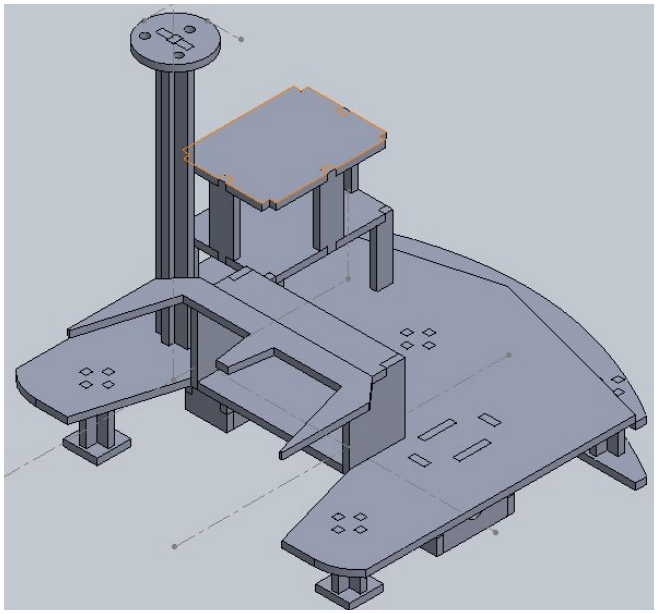
Below are some of the design sketches that we initially presented when we first began this process:



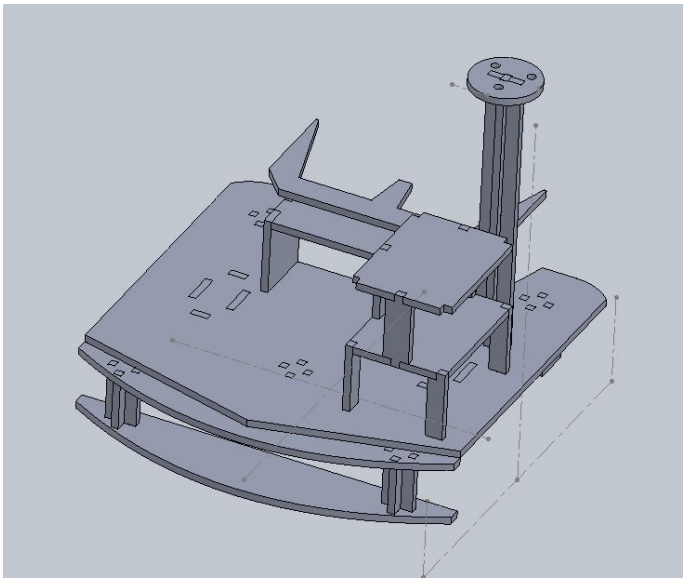


As we progressed through the build we added more detail and refined our design to actual pieces that were available. For example we replaced the skids with nice ball casters that and the tube that fed the balls to the shoot became straightened out. we also moved our plunger up to ensure that it would hit the panel and modified our bumper to be larger so that it could detect more bumps and nothing accidentally rolled underneath our bot.

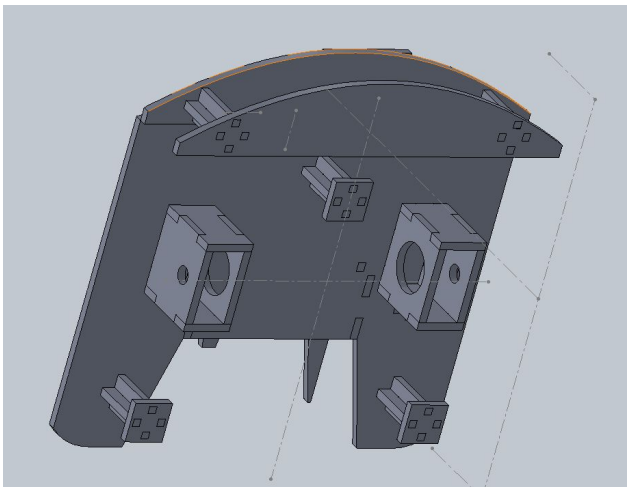
Solidworks Sketches:



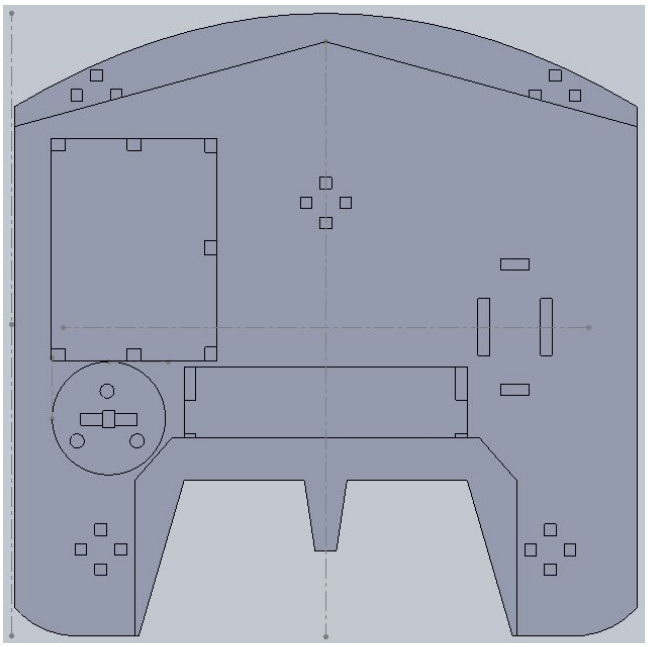
The rear plunger was designed to assist the bot in reversing on the tower and properly collecting ammo. Two bump sensors were attached to the rear plunger to tell the state machine when the bot was square against the ammo tower.

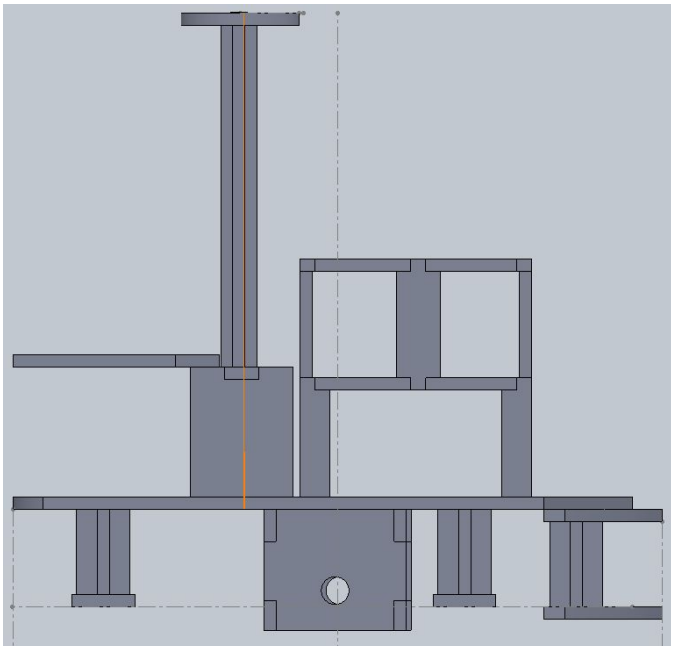


The stacks of MDF on the front left side of the bot were designed to support the Beacon Detector, Track Wire, and and Tape Sensor circuits. The empty right side of the bot was designed for the Uno stack and any excess wires. Since our bot's base was raised so high off the ground, the front bumper was extended down to ensure we could detect any low bumps; as well as add flair and style.



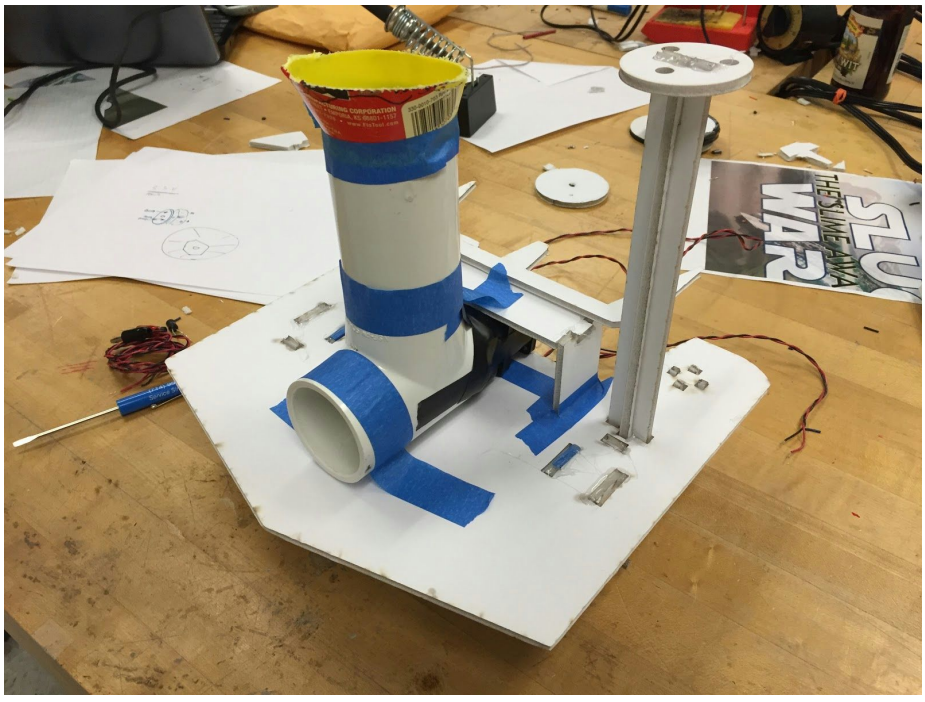
Three ball casters were used to support the bot's weight. The ball casters themselves were supported from the three pillars protruding from the underside of our main base, positioned for optimal support. On the underside of the bot near the rear cutout for the plunger is where the battery was intended to reside.

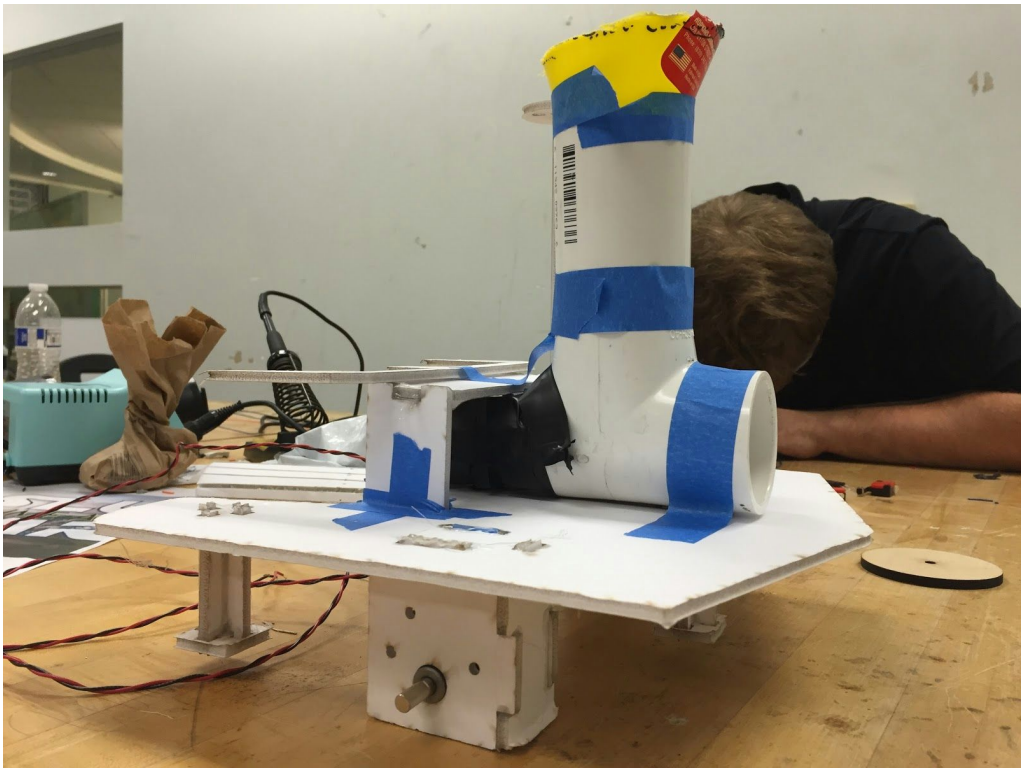


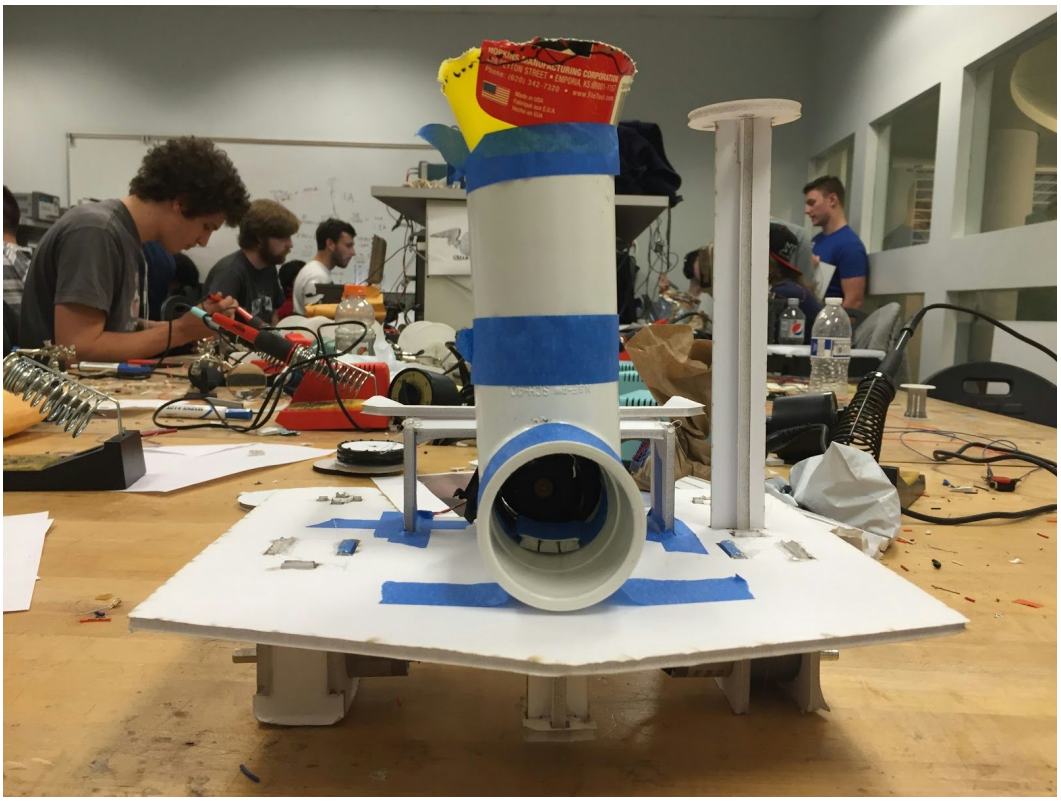


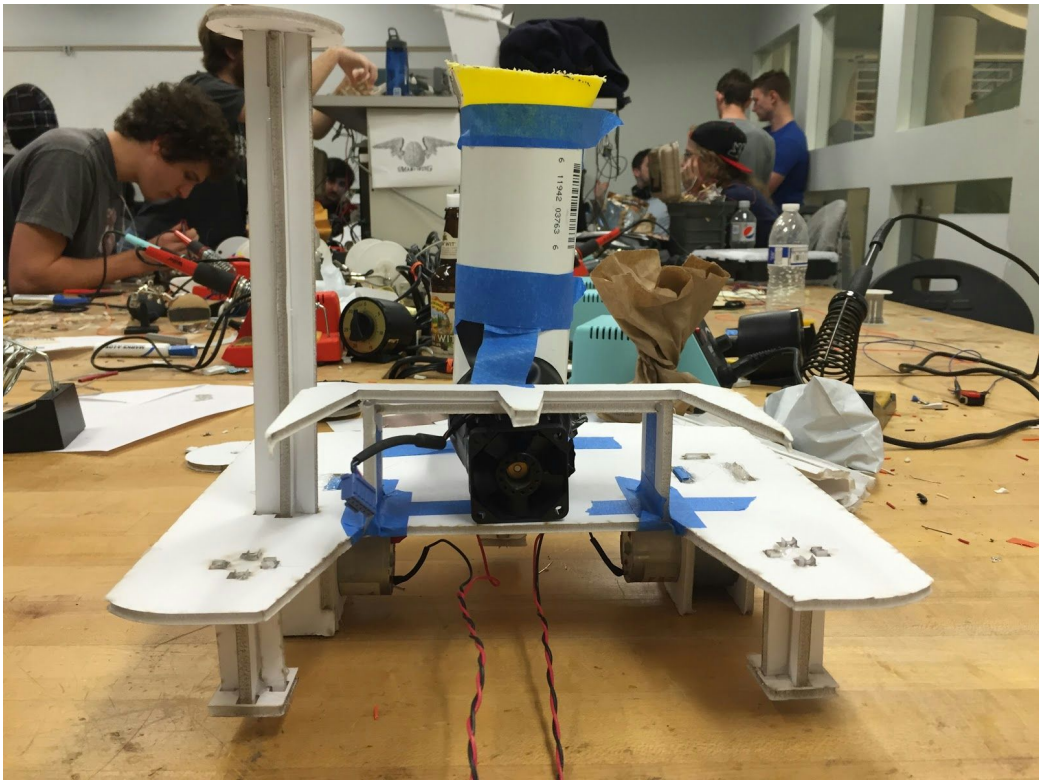
We designed the motor mounts to fit around a lip that encompassed the shaft of our motors. We did not laser cut any hole to place holes on our motor mounts because we could not precisely measure the position of the holes on the motors so we forwent without them and drilled them out my hand later. Similar tactics were taken with mounting holes for the electronics, and the UNO stack.

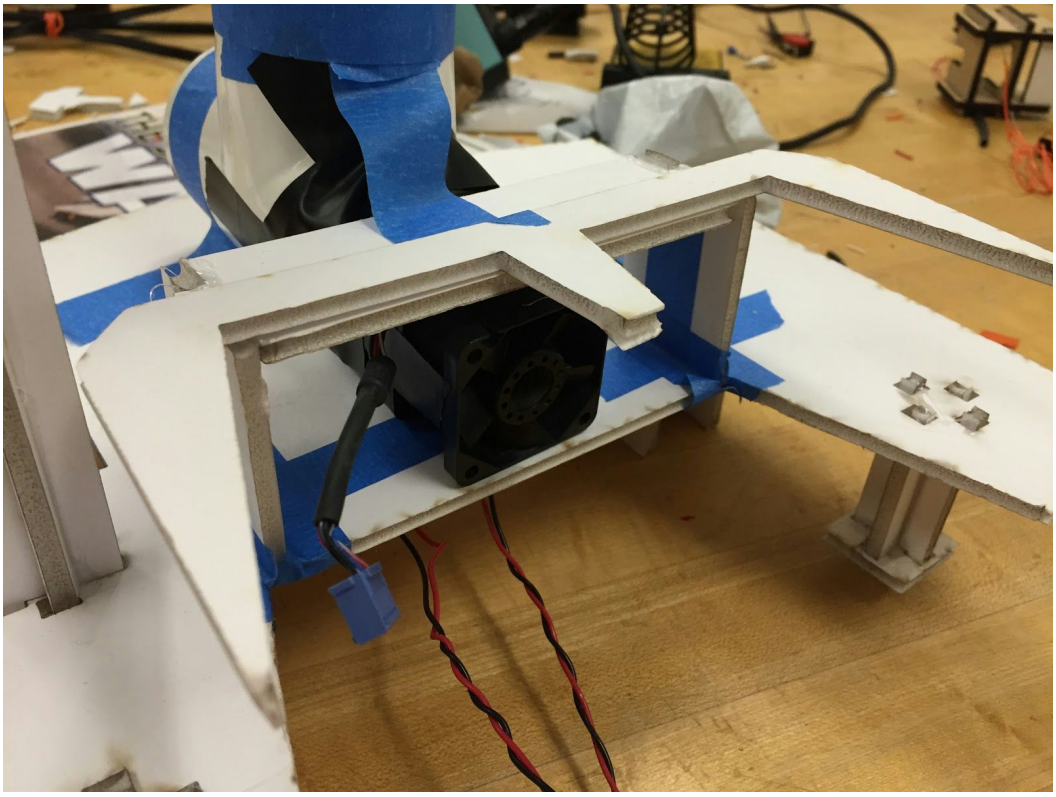
Foamcore Prototype:











Ball Collector and Launcher:

When collecting ammo, we assumed we would be approaching the tower sloppily; leading us to realize that we needed a wide collector. A small soccer cone proved to do the trick, with a small lip to catch balls that potentially rolled out too hard,

PVC pipes, and T-joints were the selected method to funnel balls. By cutting a small slit in the pipe were able to insert a small arm that would act as a stopper, the arm was attached to small servo which would actuate up or down, when we needed it.

For the launching mechanism we went with something that would incredibly easy to mount operate and power. We salvaged a few 12V DC server fans. They operated pulling no more than 400ma at 10V and full spin, and a servo was mounted to hold the balls back until we were ready for launch.

The launcher was made by sawing off one of the ends from a PVC T-joint, and taping the fan to the sawed off corner the T-joint. The shaft then descended into the T-joint and the remaining exit hole acted, as a small launcher.

Electronics:

Once we we decided on a design, we were able to figure out what kind of sensors we wanted to use and how many of them we needed to make. This included a beacon detector, six tape detectors, three track wire detectors and front and back bump sensors. Also with the idea for a fan ball launcher, a servo motor and a fan would need to be hooked up. Our plan was to get the sensors done and working as soon as possible, making sure we aren't scrambling last minute to solder sensors up, and also to write and test the proper hardware drivers of them, to make sure they will all work together on the same bot.

Starting with the beacon detector, because we had one soldered up from lab 2, we began to debug the circuit because although it was functioning perfectly the output was not what we liked. Giving us us an output from 3.3V to 2.8V. We needed to increase the difference in these numbers so we could add a hysteresis in the code, and not mistakenly think we found the beacon when we really have not. After analysing the circuit, there was an additional high pass filter that was doing double duty, so we removed it. Additionally we re-calculated the comparator resistor values to give us a greater output range. After adjusting the comparator values to 20KΩ, 4KΩ and 10KΩ we got a larger difference in out output from 3.3V to 1.3V. This gave us the difference we needed to properly adjust for it in coding. But after we made this adjustment, when we tested it again, we noticed it would only pick up the beacon from a max distance of 3 feet and it was never consistent with this range. KNowing it worked at 6 feet, and

not knowing how we far we would need to detect an opponent, we began investigating. It turns out one of the big yellow wires going across the circuit was not 100% secured down to the board, and depending on the position of the wire, it would cause the detector to lose range and flicker. WE swapped out the wire with a new one and everything worked perfectly. We were originally going to build a second beacon detector and place one on either sides of the launcher so we would know when our robot is square with the opponent (both beacon detectors are high). But after the time spent laying out the design for the perf board along with the time debugging the one beacon detector we already had, we decided time would be best spent working on other sensors we need.

Next we tackled the track wire sensors, because again we had built one in a previous lab. Using Ian's design as per being the simplest, we breadboarded it up to make sure it gave use the readout and distance we wanted. After this was confirmed, we began planning out 3 track wire sensors for one perf board. Our goal was to put two in the front and one in the back of the robot so we would know when the entire bot was inside the hyperloop. It took a good part of a day to plan and place all three to fit on one board but we were able to fit everything in perfectly. After testing each part once soldered, we eventually had three working soldered track wire sensors. We used female connector pins to attach the MCP6004 op-amp chip just incase one does not 100% work or blows out later down the line. This way we can just pull out the old on and pop in a new one without re-soldering. In the end, once we designed our state machine, and had a tactic to wall follow the center obstacle to search for the active hyperloop, we only used the track wire that was in the back of the robot, because as our bot wall followed in a clockwise manner we knew that most of our robot would be in the active hyper loop once the back of the robot sensed it, triggering a full stop.

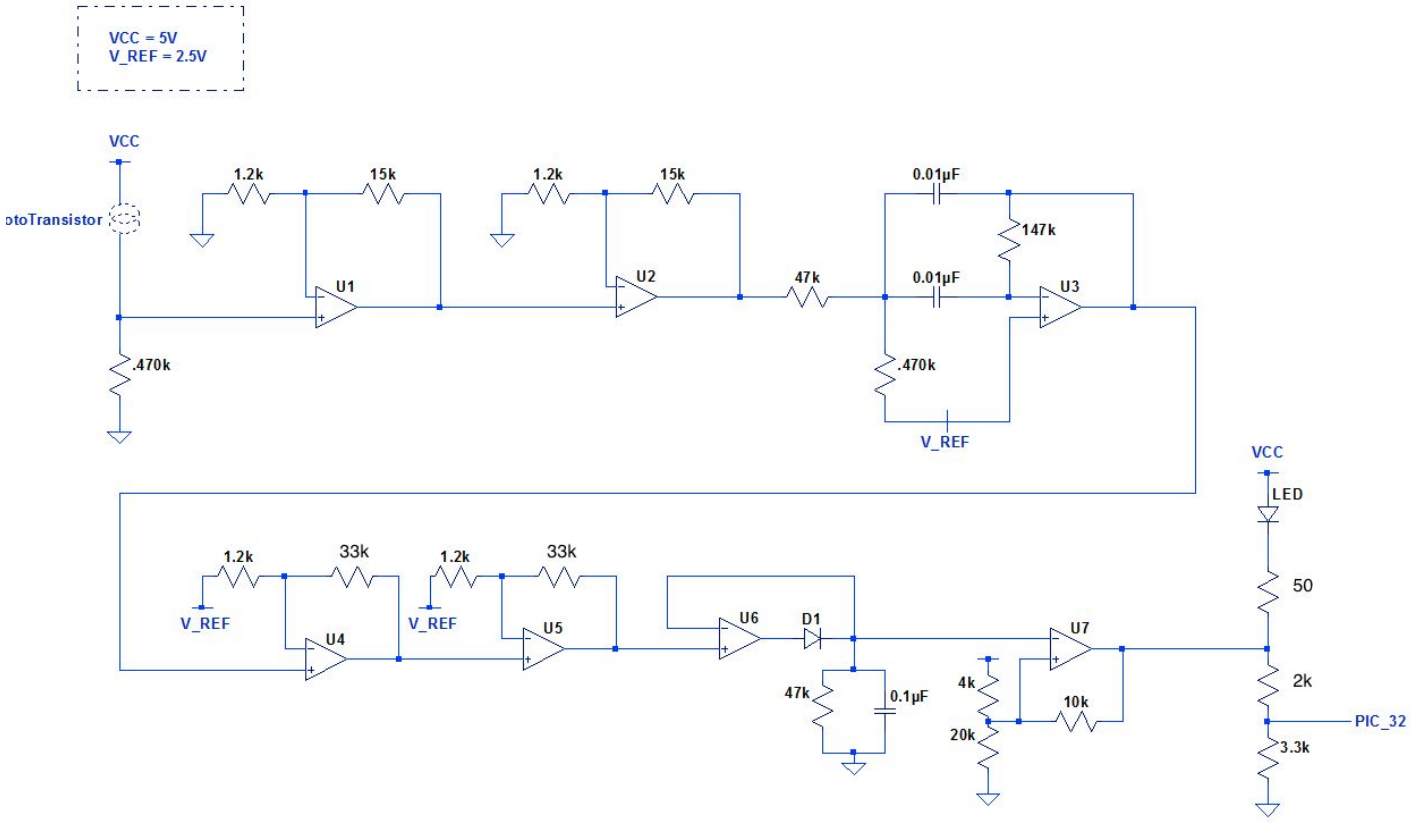
The next sensor we worked on was the tape sensors. After deciding on using 6 of them, and knowing each tape sensor used 4 wires, we aimed to keep the circuit as simple as possible. After checking the data sheets for the tape sensors, since we have never used them before, and some prototyping on the breadboard, with just two resistors, one for the photo transistor and one for the infrared emitter. We wanted an easy way of attaching al 6 sensors, for simplicity and incase one of the tape sensors stopped working. So we again used female connector pins and soldered the sensors appropriately onto male header pins. THis made for a much easier and cleaner circuit to solder. Later down the line, once we began coding the state machine, we decided to implement synchronous sampling, especially because our robots base was so high off the ground, allowing a lot of light from above to interfere with the tape sensors. This wasn't too much of an adjustment when it came to the circuit itself, new resistor values had to be calculated for the infrared emitter as we were attaching it to the TIP 122. AFter all was soldered on we ran into no problems reading the sensors.

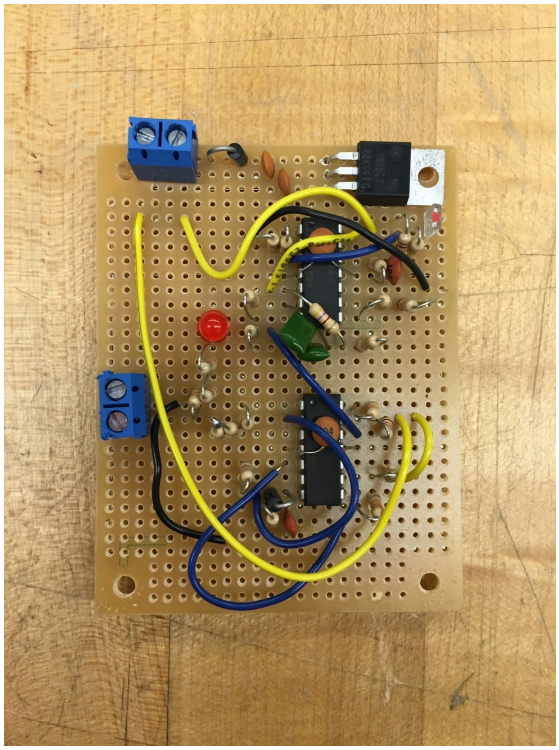
The fan for the launcher worked similarly to that of the tape sensors, in that we are using the TIP 122 to turn off and on the fan with the addition of one resistor between the PIC and the base.

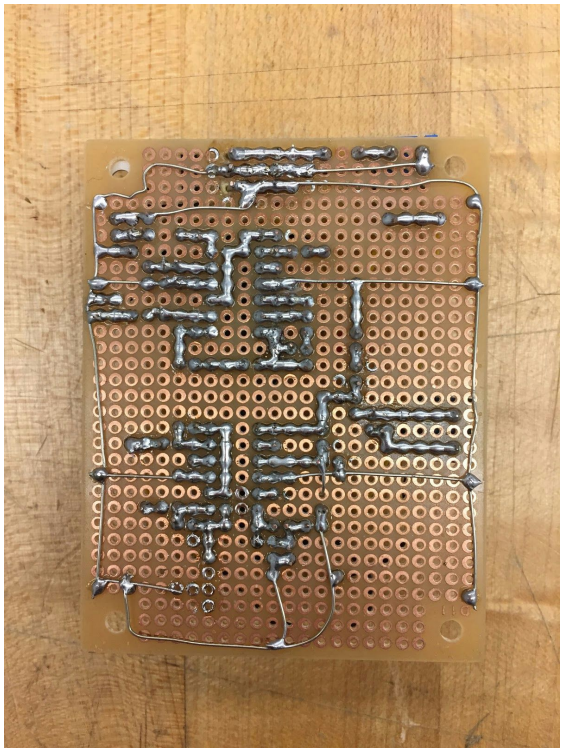
Lastly the bump sensors were the easiest as they did not need much additional hardware to work. A 22 Ω resistor was soldered to the output going to the PIC, other then that they just needed to be powered.

All schematics and sensor pictures can be seen below:

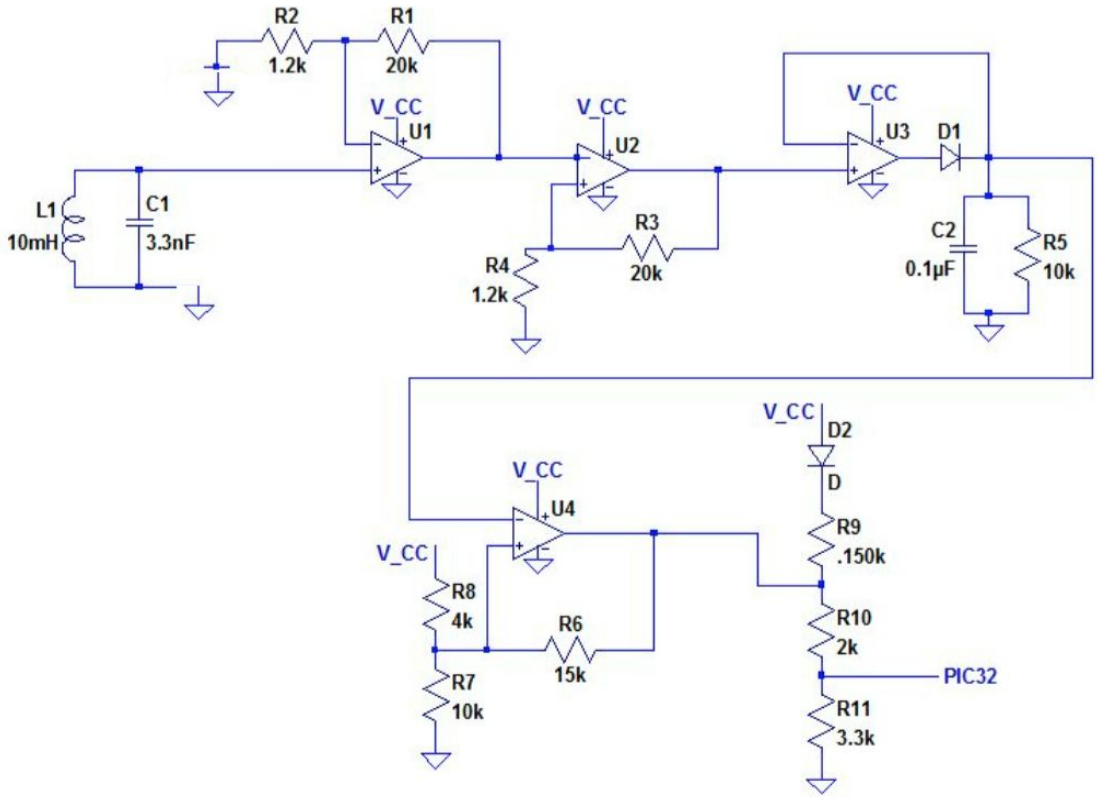
Beacon Detector

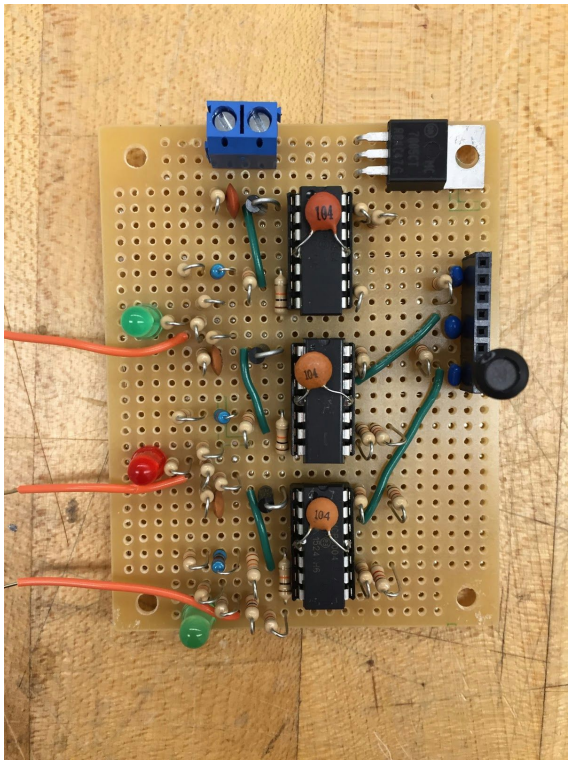


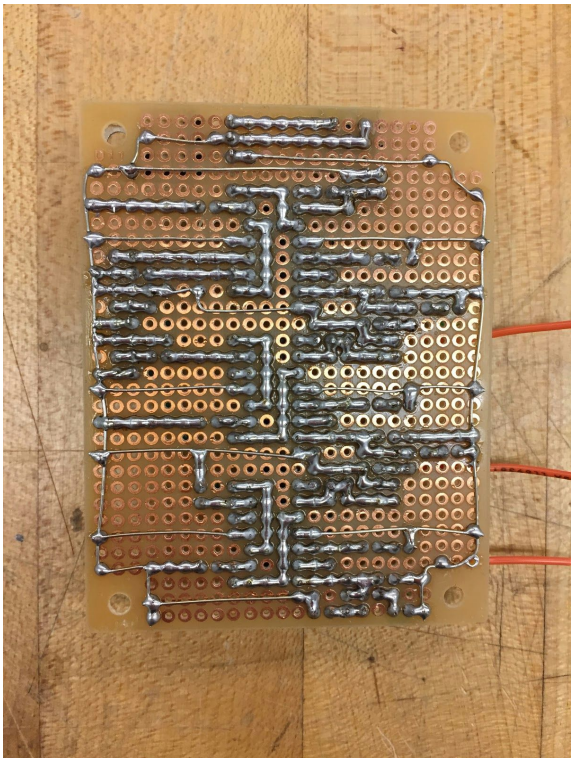


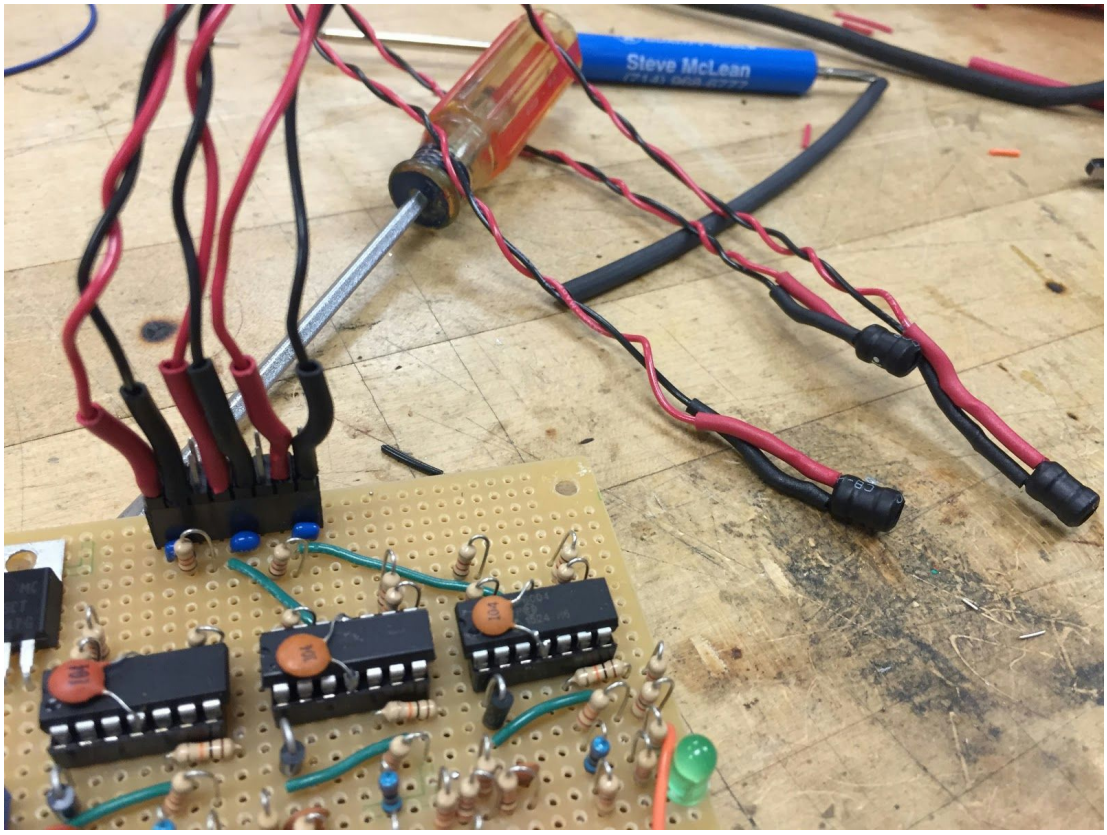


Track Wire

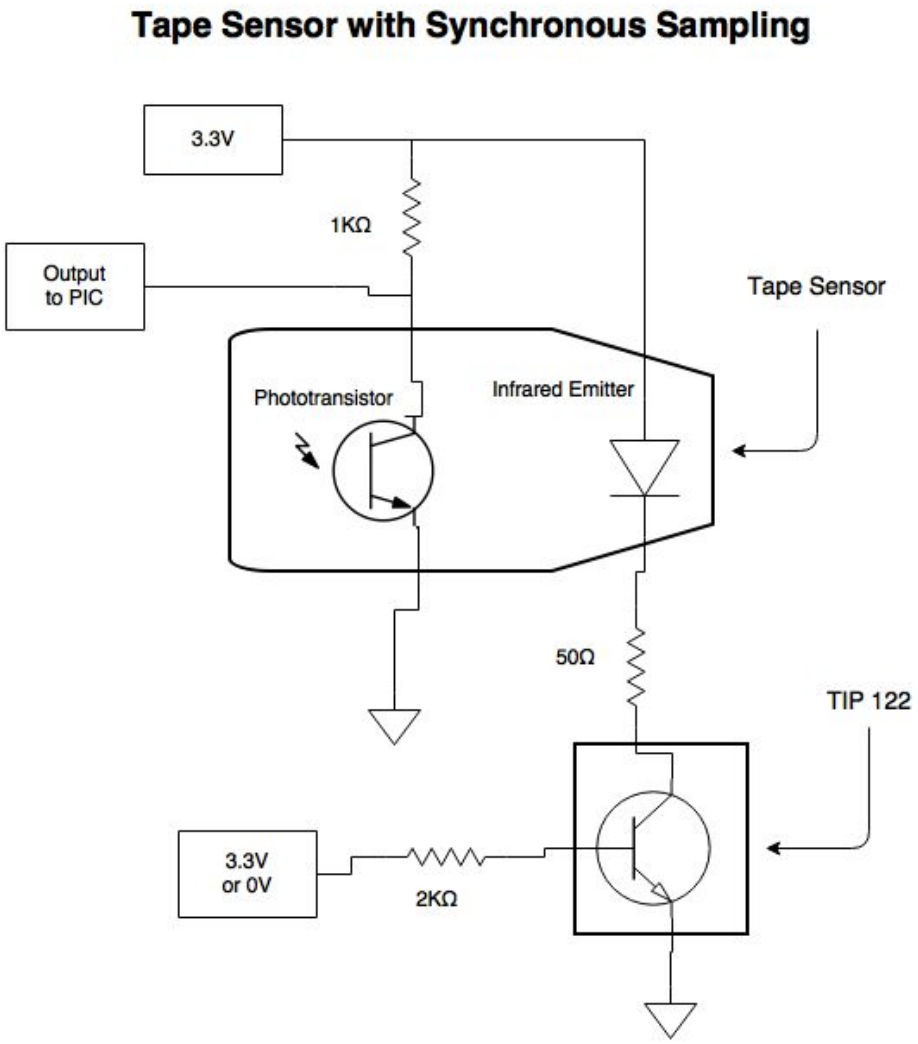

R2
1.2k
R1
20k
V_CC
U1
V_CC
U2
V_CC
U3
D1
L1
10mH
C1
3.3nF
R3
20k
R4
1.2k
C2
0.1µF
R5
10k
V_CC
D2
V_CC
U4
D
R9
.150k
V_CC
R8
4k
R6
15k
R10
2k
PIC32
R7
10k
R11
3.3k

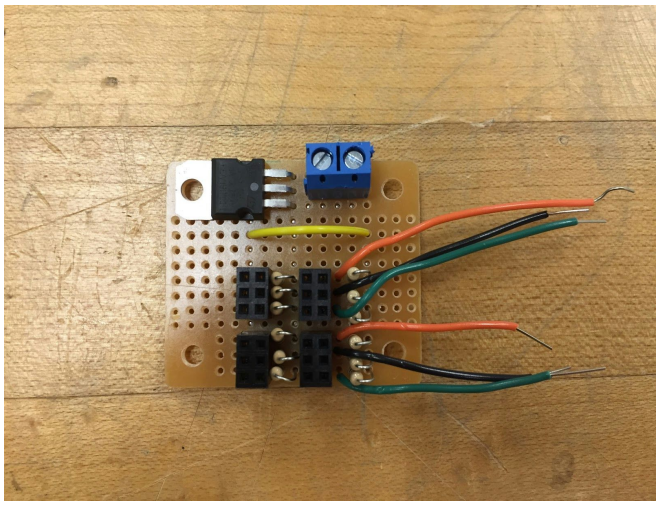


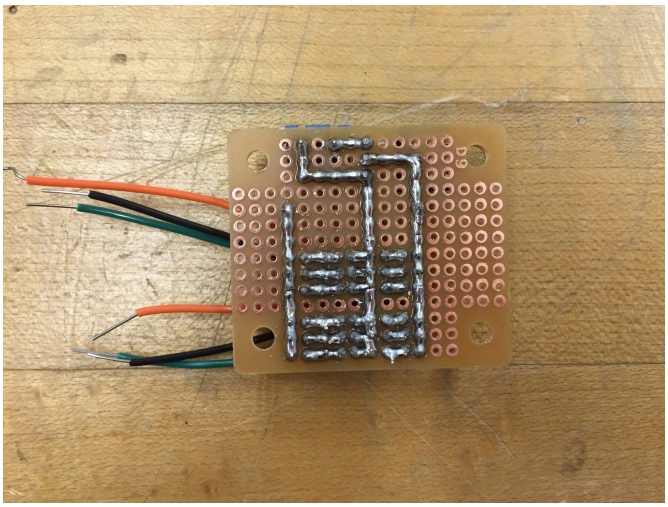


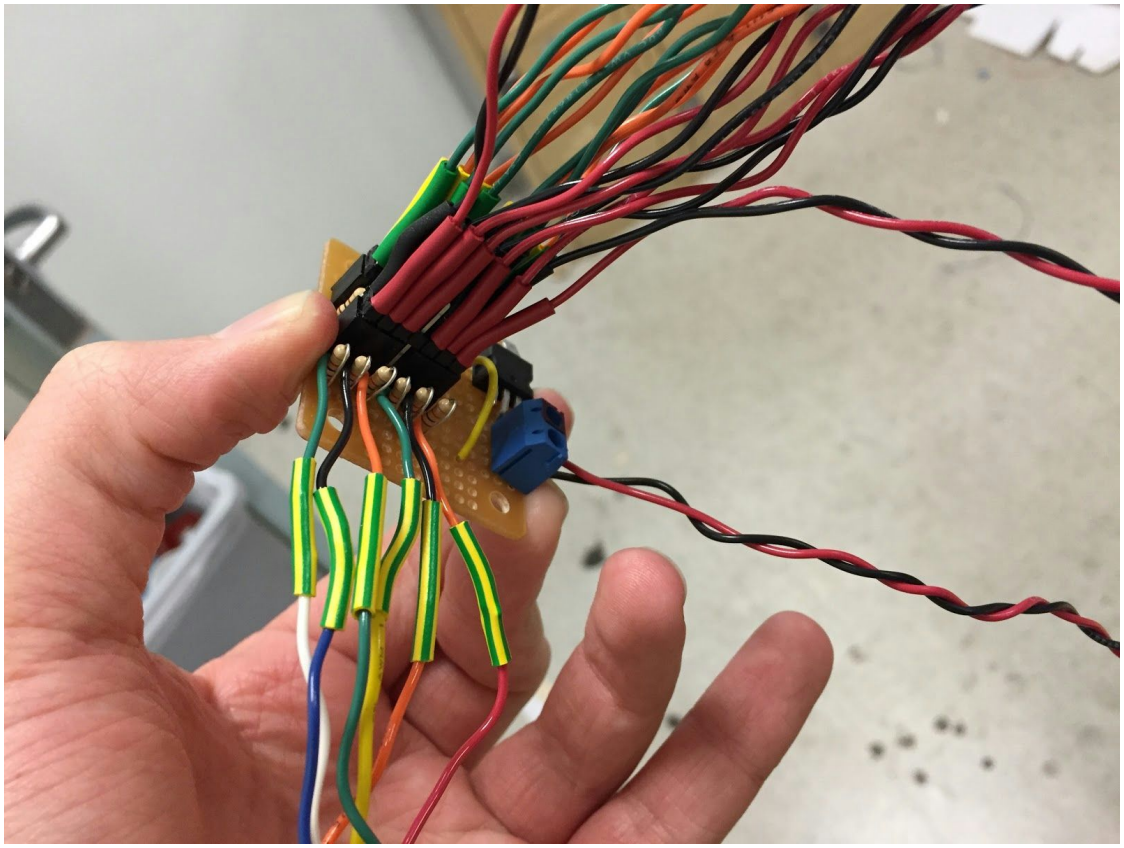

Steve McLean

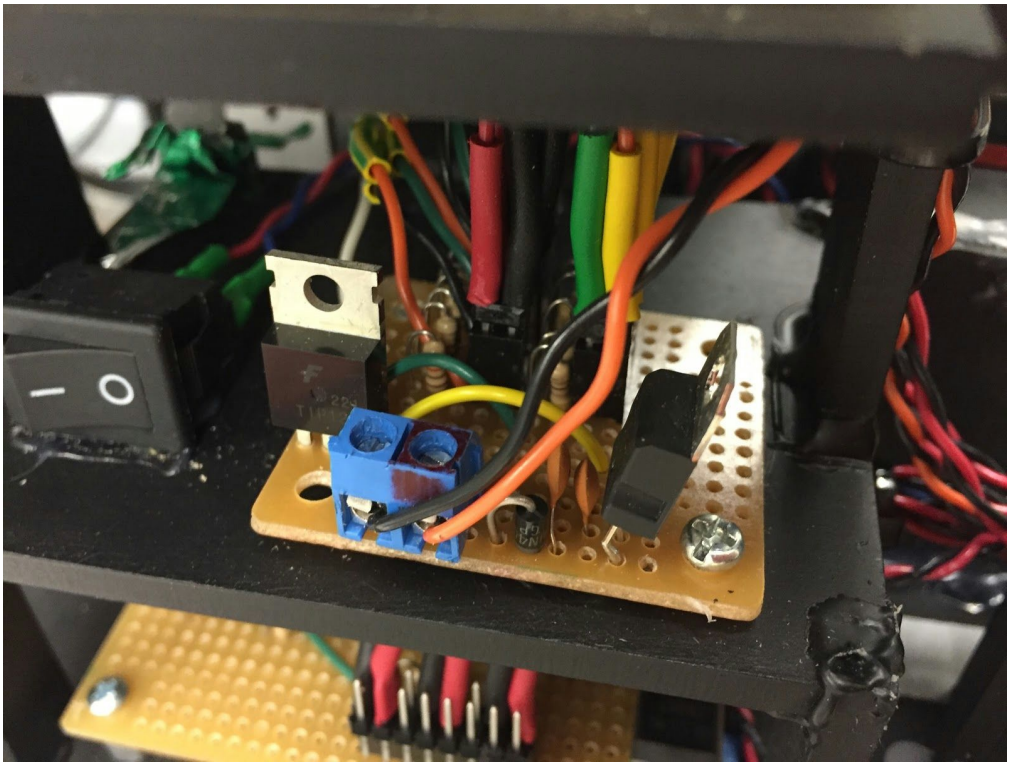
Tape Sensors





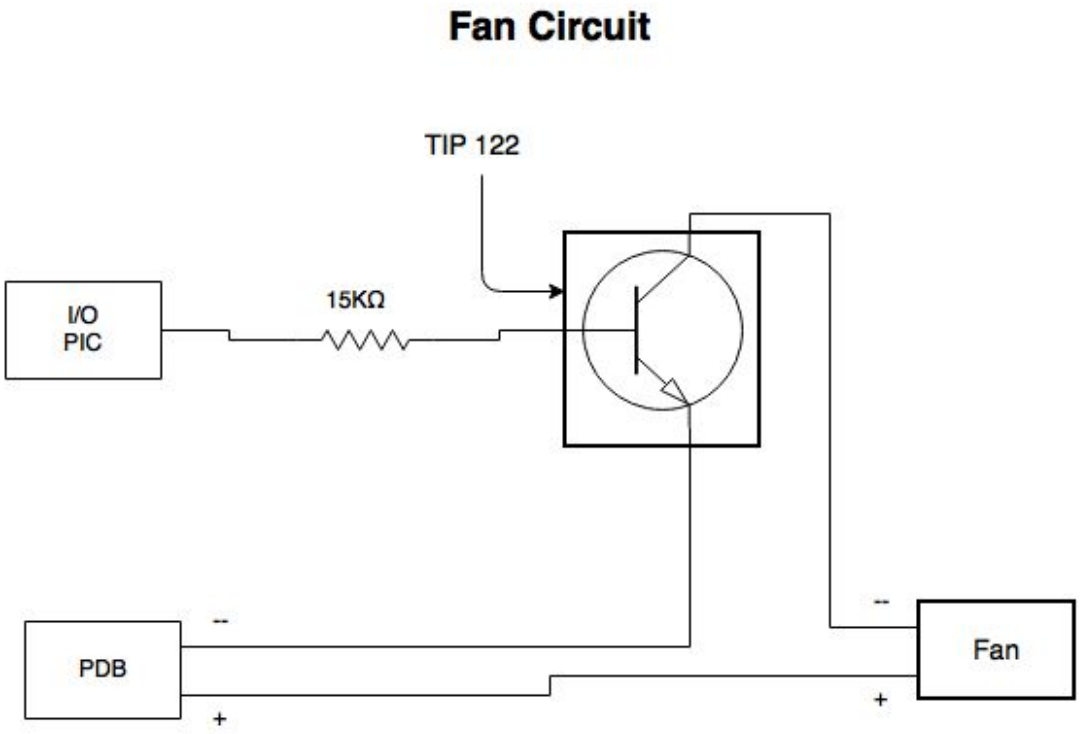


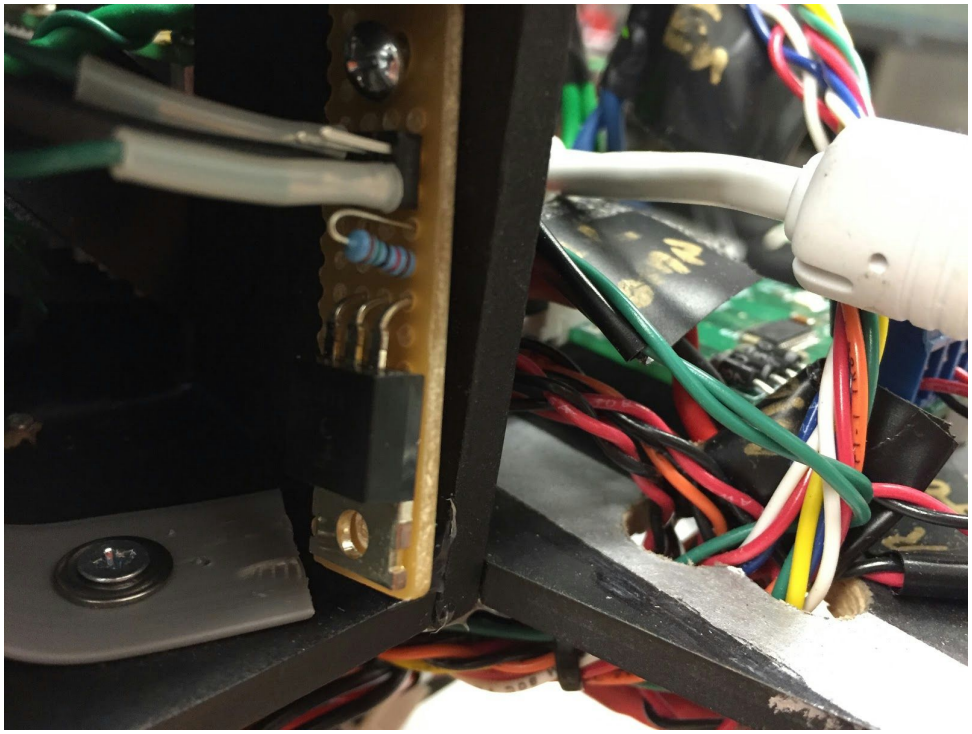




WIth TIP 122 for Synchronous Sampling

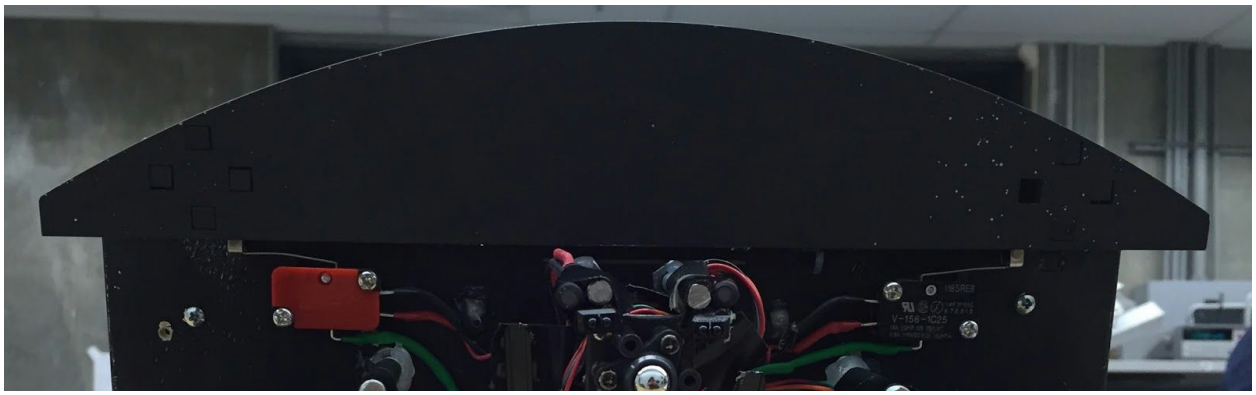
Fan



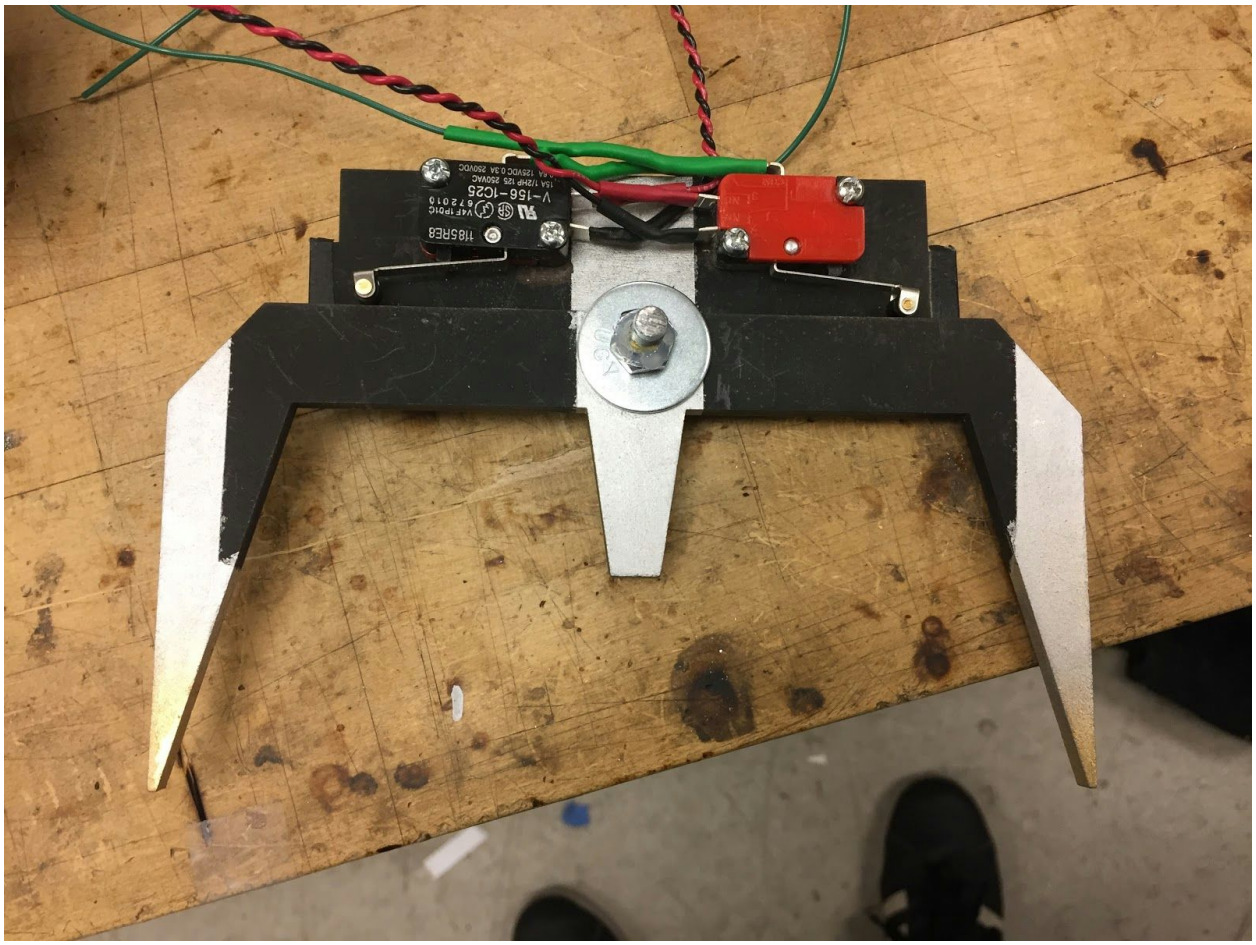


Bump Sensors

The green output wire has the 220Ω resistor.



Underneath view of our front bumper (using two bump sensors)



Back bumper with Gold Tips
(using two bump sensors)

Other Electronic Issues:

We began this project with a different H-Bridge to drive our motors. We had the SN7544 which works great, except it has a lower cutoff current. So when we stalled our motors, we would reach a current of about 1.2A at which point everything would die. Even with the H-Bridge on a separate fuse on the PDB from all the other sensors, we would have no luck. It helped to run our motors at a lower PWM, but we still ran into the issue every once in awhile. Luckily another team in our lab had an extra 8814 that has a 2.5A cutoff. Once swapped out, we did not run into the problem again. But we did spend a few days coming up with alternative solutions to avoid the cutoff current.

Another big issue we ran into was in respect to our PDB. The times that we used in our state machine to tell the robot how far to back up, turn, and wall follow we all greatly impacted by a small drop in our battery voltage. Basically speaking, if our battery was at a very specific voltage, these timers would be way off causing the robot to back up and turn for much longer than it should. THis caused a lot of frustration when debugging out state machine, trying to figure out why our robot was performing something different each time we ran it, and not staying consistent. We would adjust our timers to compensate, but when we put a new battery in, the timing would be completely off again. After triple checking our motors, our code and our robot, one suggestion came about about the PDB. We checked it out, and got a new PDB with two good fuses. The NEXT run we did with the new board we got checked off with the oversight of Max L. The second time after that we ran the robot, it completed the entire task (checkoff worthy) at the competition. Because of this we are lead to believe we had a faulty PDB, and continued to have no problems after.

Additionally, this is not so much of an electrical issue but it relates to our electronics. Every once in awhile our robot would not detect tape when it should. Causing it to overturn the corners and miss the T. AFter checking the code as well as the sensors themselves to make sure they were all on and working we could not find the issue. This led us to how we mounted the tape sensors on long threaded screws. We held the tape sensors to these screws that extended to the ground with electrical tape. We figured out because of the cold nights, the electrical tape shrunk and raised the tape sensors off the ground to a height in which they did not work WE could not believe this, and was able to fix the issue by pulling the tape sensors back down and applying extra tape to hold them in place.

Mechanical:

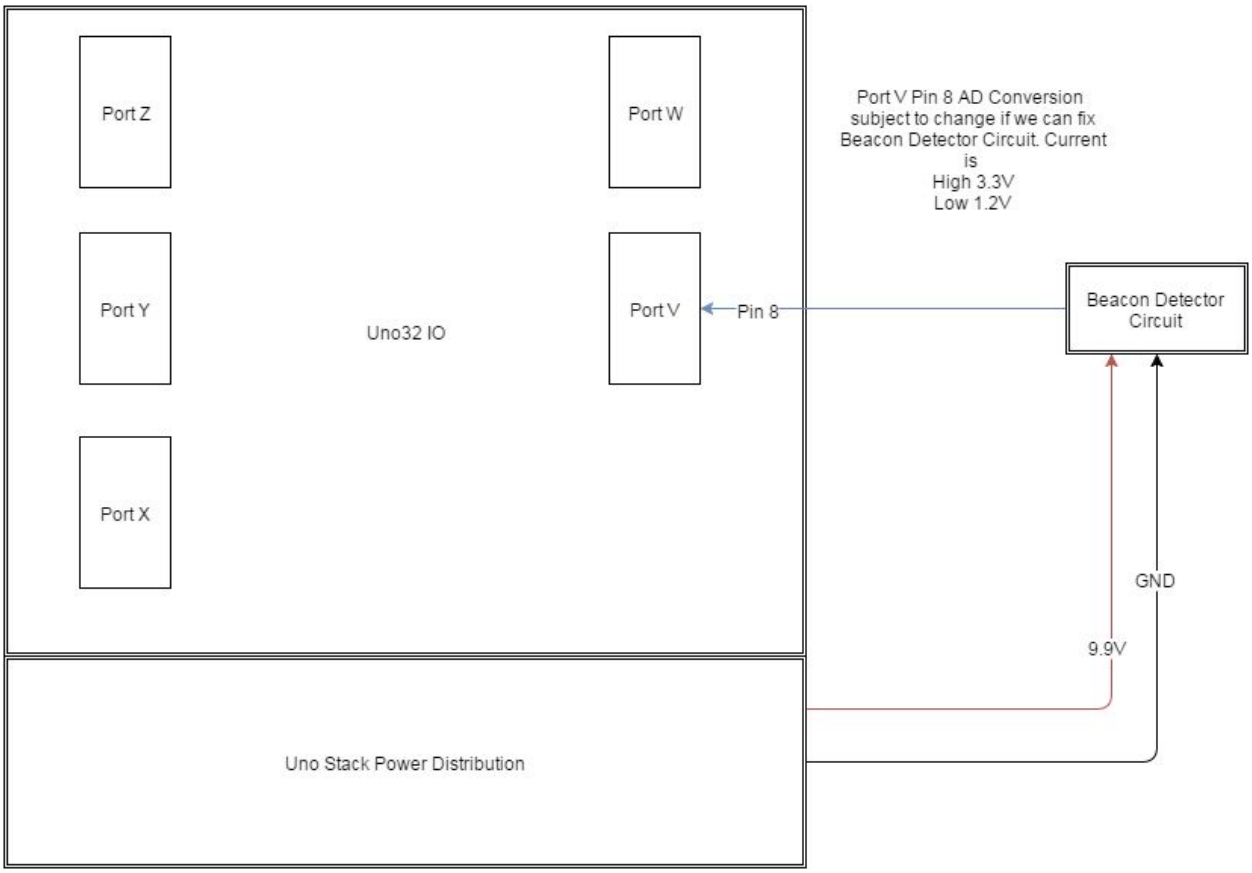
Hardware Drivers:

Pin Choices:

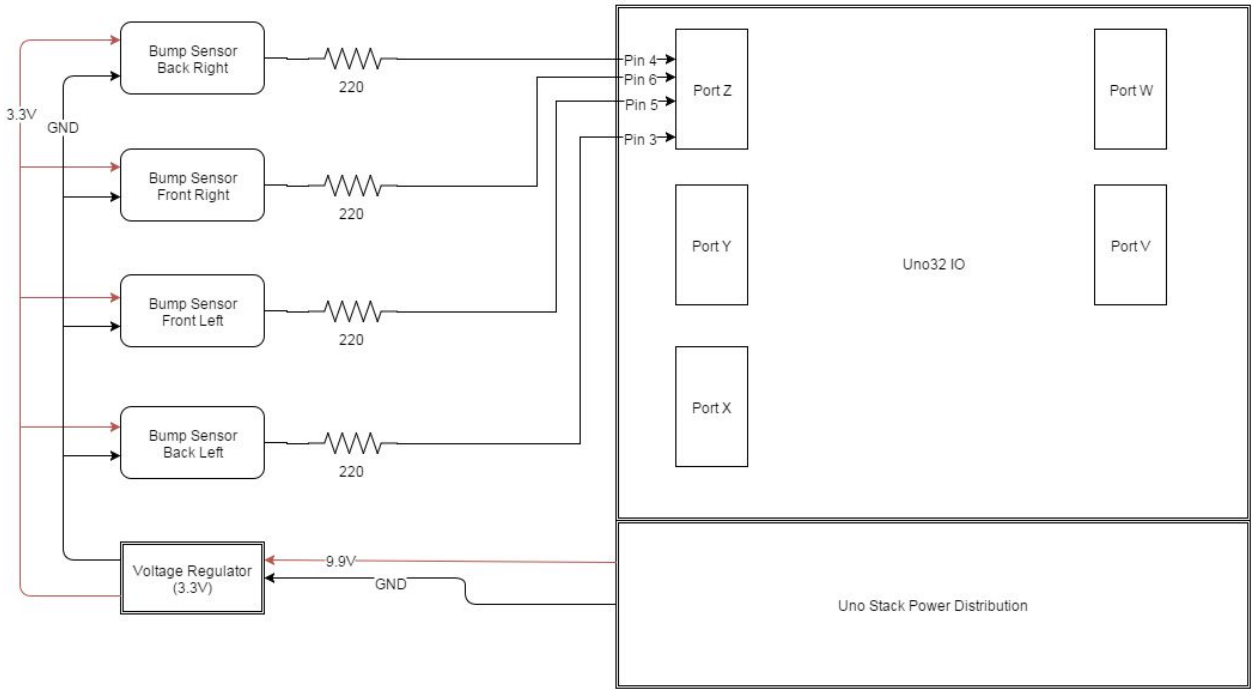
When making the decisions of which I/O pins to connect to which components we needed to think about the needs of the component (i.e. each motor needed a PWM, and each sensor needed an ADC reader). So we had to take those aspects into consideration with this table to help us:

Component	Pin needs
Drive Motor (x2 through H-Bridge)	2 PWMs for enable 2 digital output for direction
Fan	1 digital output for enable
Servo	1 RC servo output
Beacon Detector	1 ADC input pin
Bump sensors (x4)	4 digital input pins

Block Diagrams:

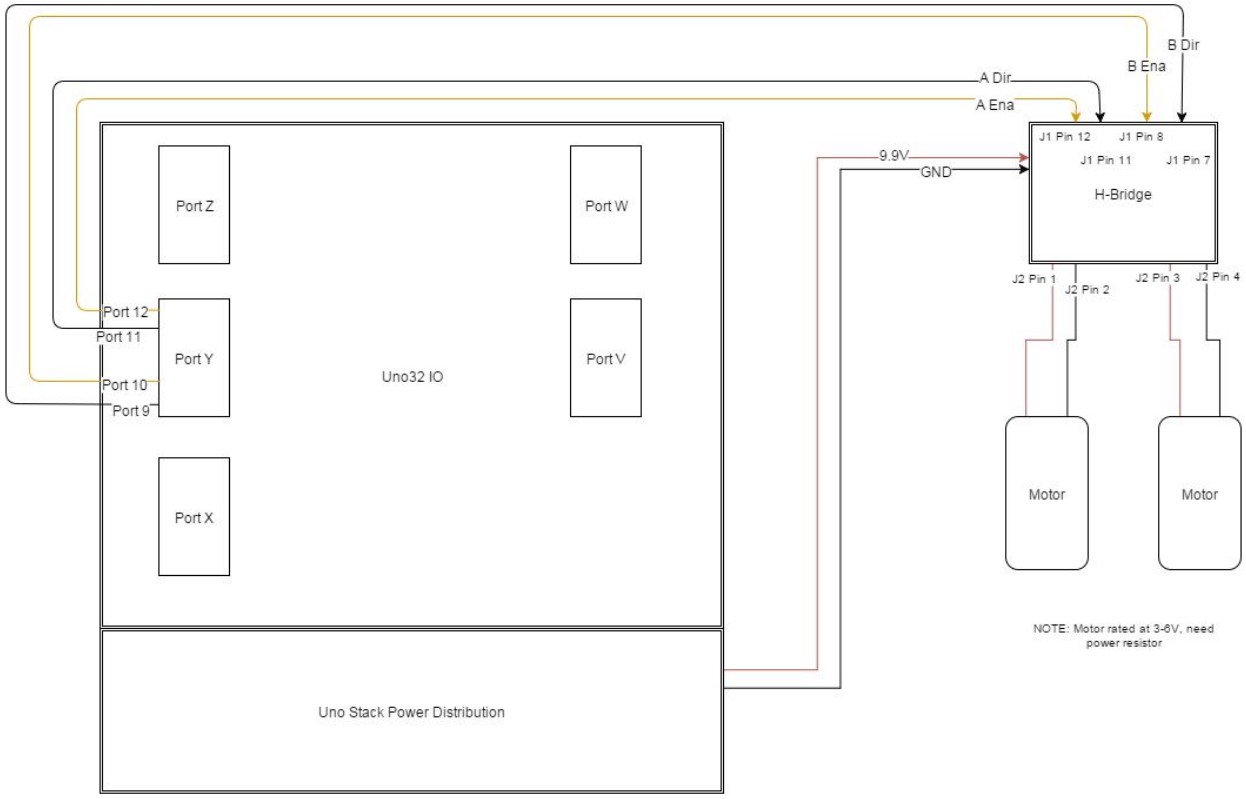


All Pins are Digital Input

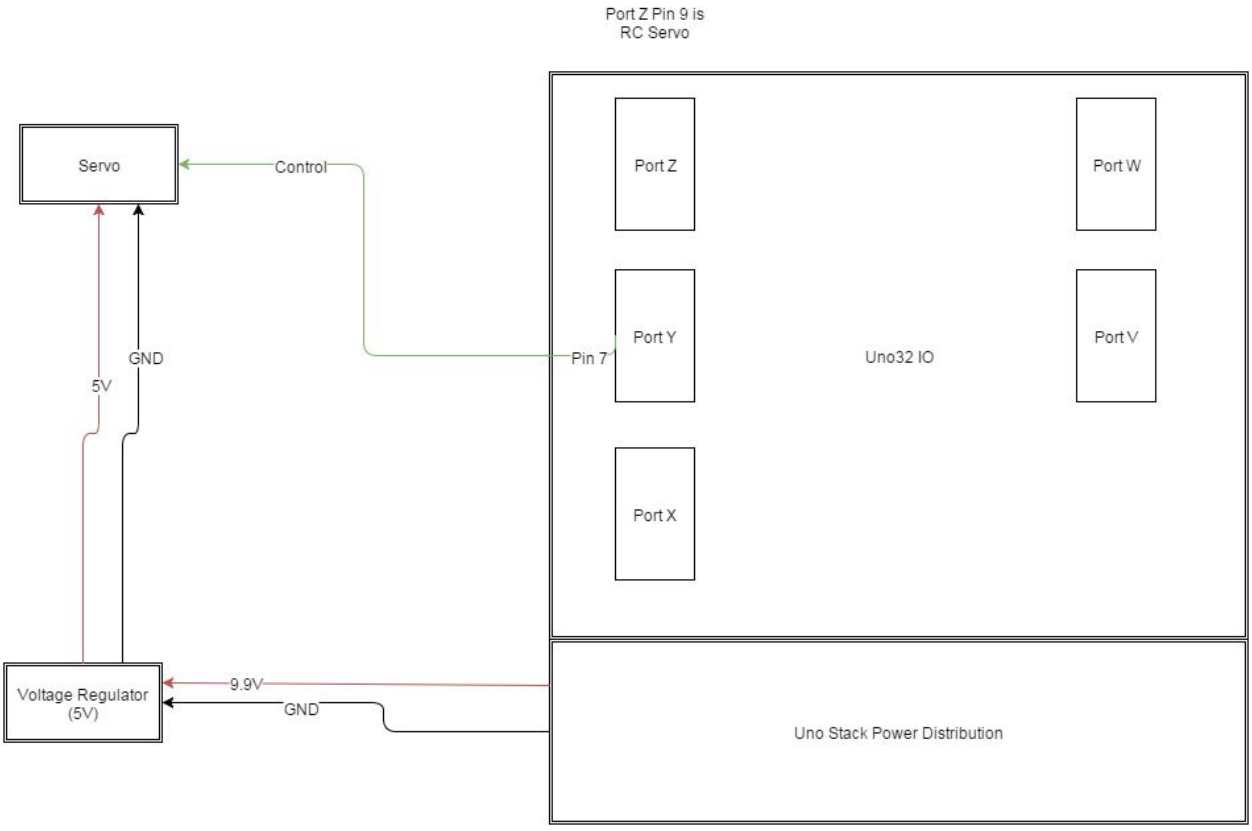


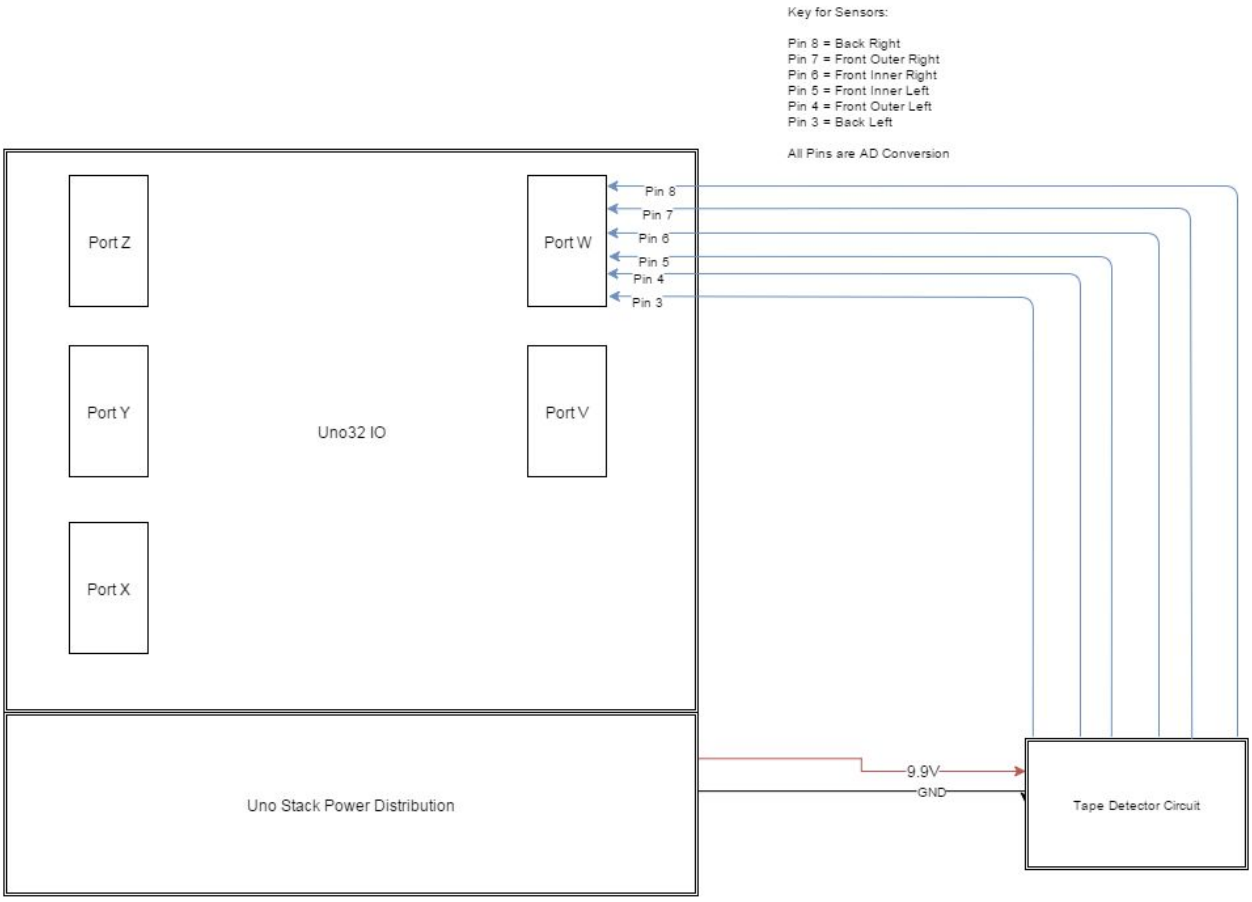
Port Y Key:

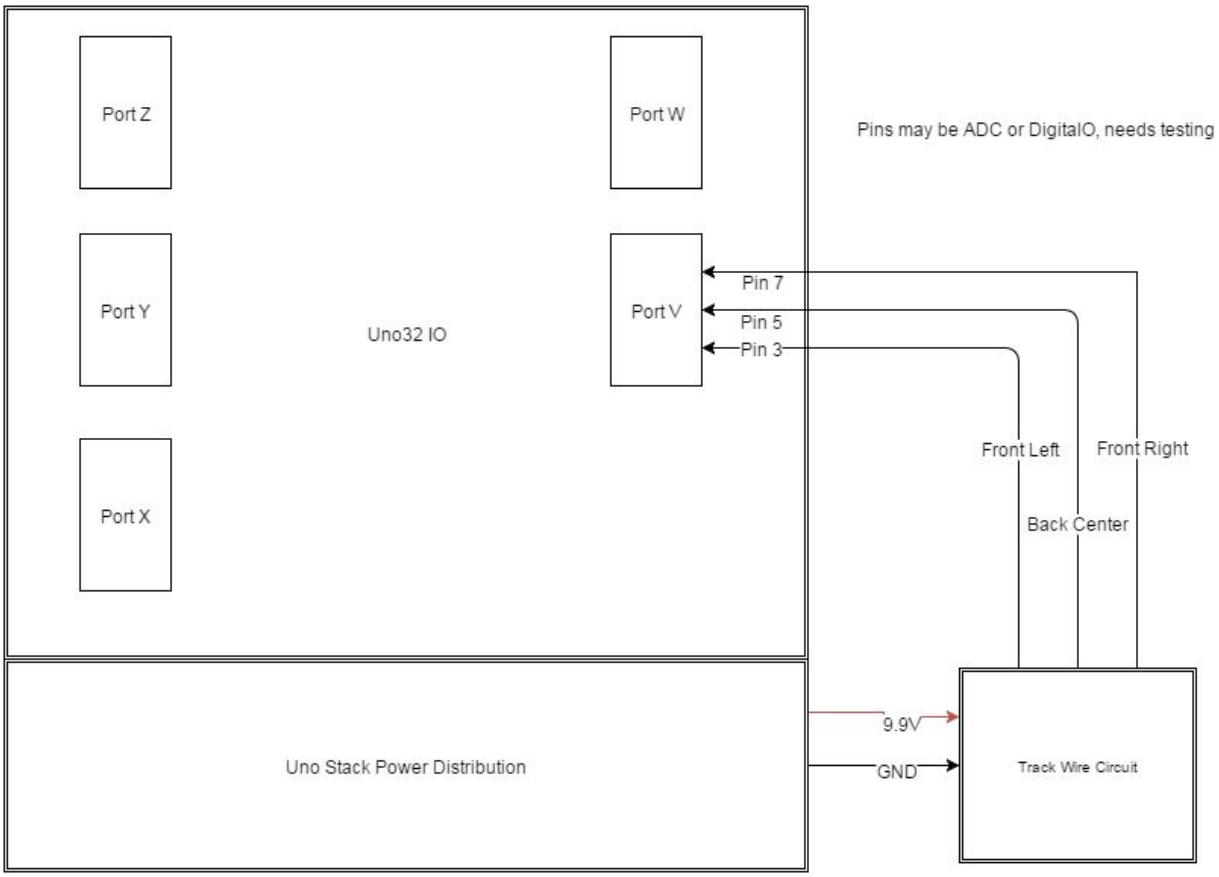
Port 12 = PWM
Port 11 = Digital Out
Port 10 = PWM
Port 9 = Digital Out



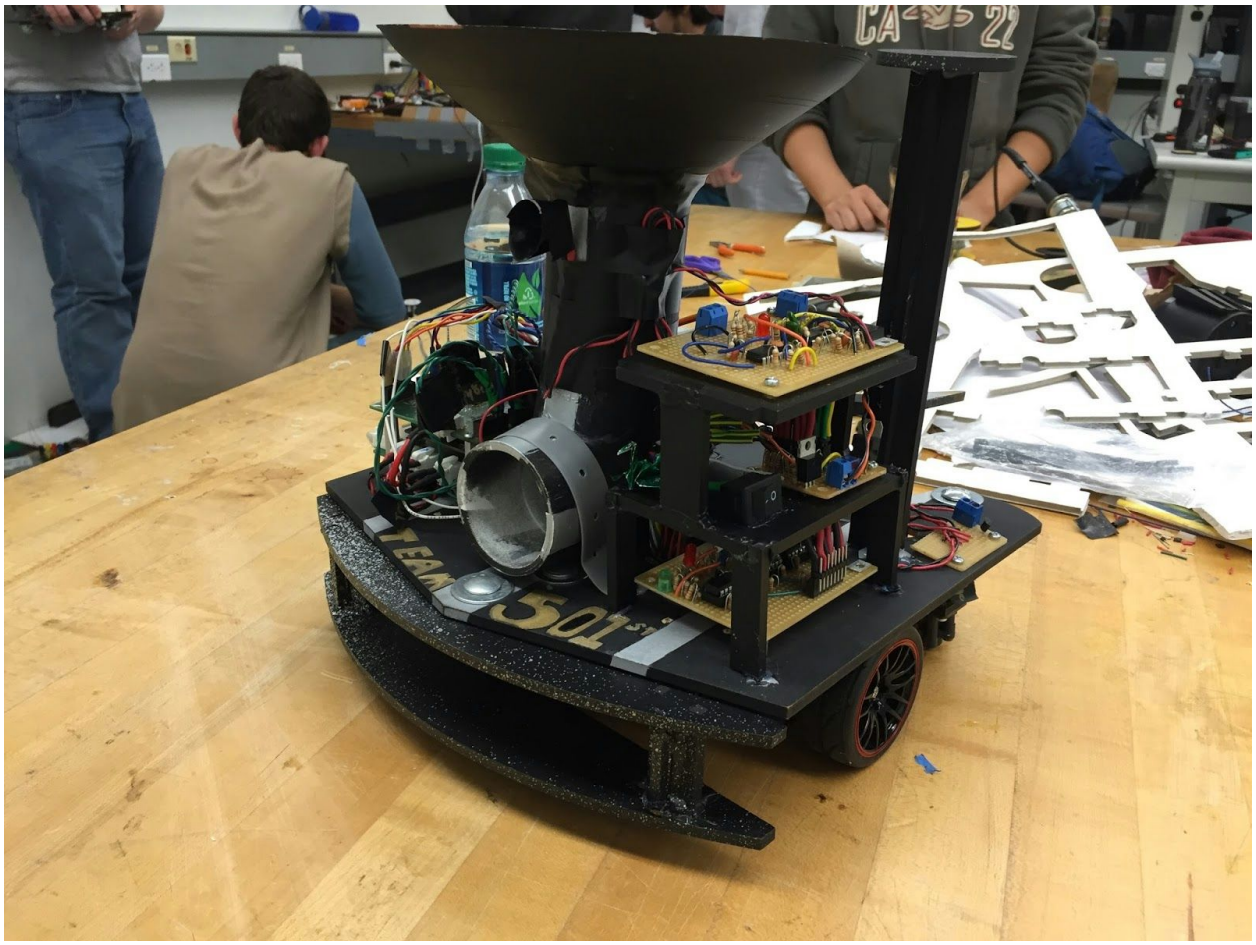
NOTE: Motor rated at 3-6V, need power resistor


Port Z Pin 9 is
RC Servo
Servo
Control
Port Z
Port W
Port Y
Port V
Pin 7
Uno32 IO
GND
5V
Port X
Voltage Regulator
(5V)
9.9V
GND
Uno Stack Power Distribution

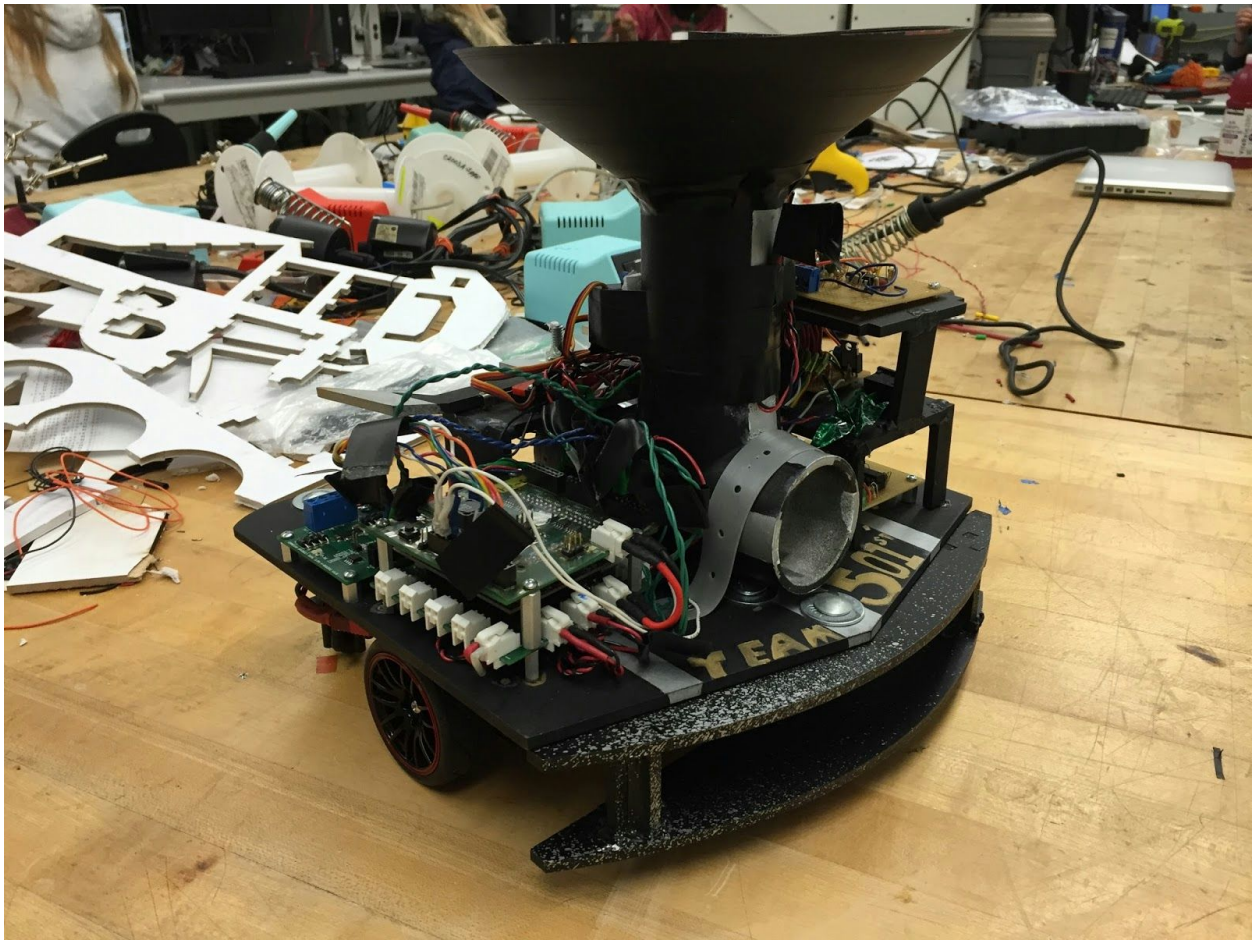

Key for Sensors:
Pin 8 = Back Right
Pin 7 = Front Outer Right
Pin 6 = Front Inner Right
Pin 5 = Front Inner Left
Pin 4 = Front Outer Left
Pin 3 = Back Left
All Pins are AD Conversion
Port Z
Port W
Port Y
Uno32 IO
Port V
Port X
Pin 8
Pin 7
Pin 6
Pin 5
Pin 4
Pin 3
Uno Stack Power Distribution
9.9V
GND
Tape Detector Circuit


Port Z
Port W
Pins may be ADC or DigitalIO, needs testing
Port Y
Uno32 IO
Port V
Pin 7
Pin 5
Pin 3
Front Left
Front Right
Port X
Back Center
9.9V
GND
Uno Stack Power Distribution
Track Wire Circuit

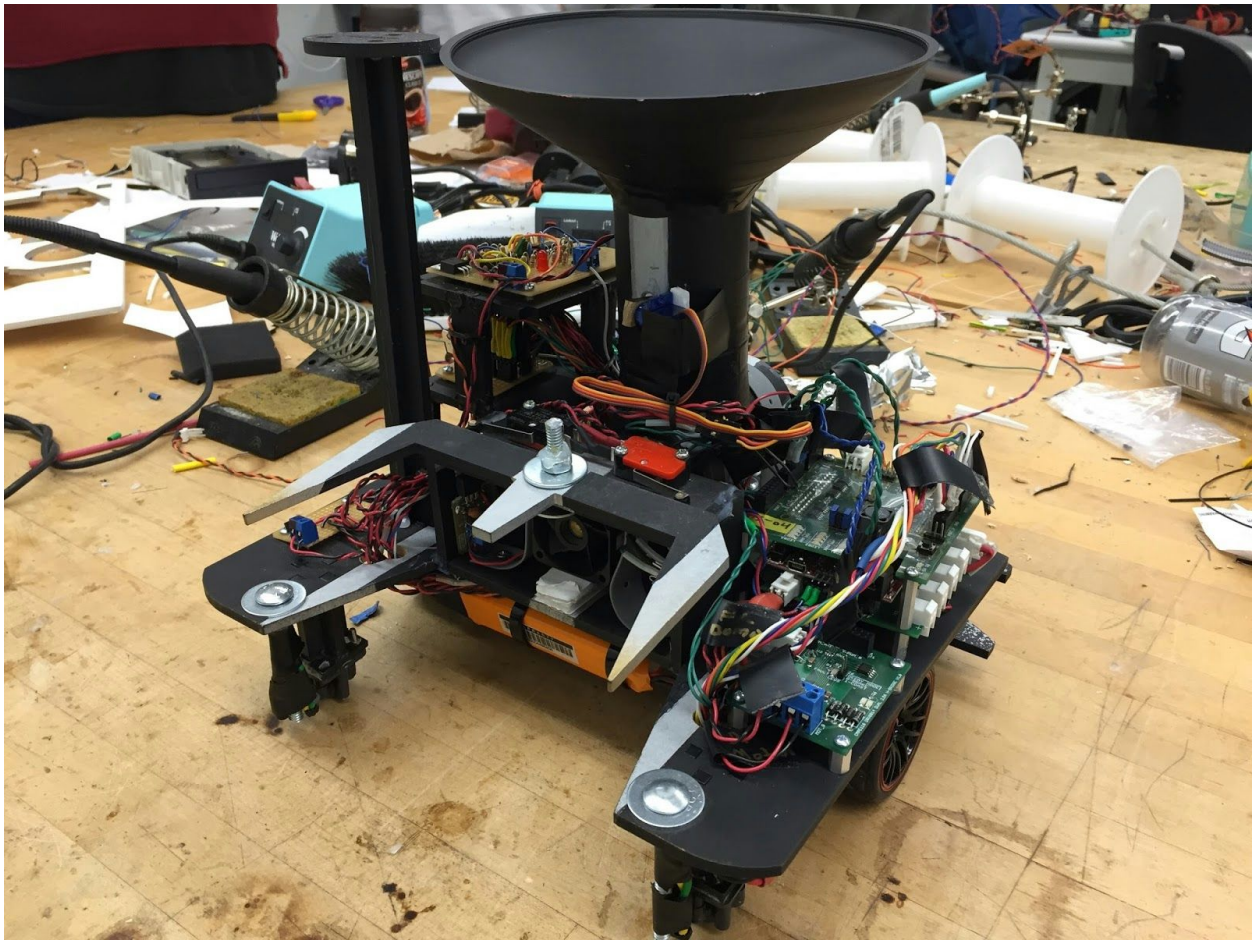
Final Design + Assembly

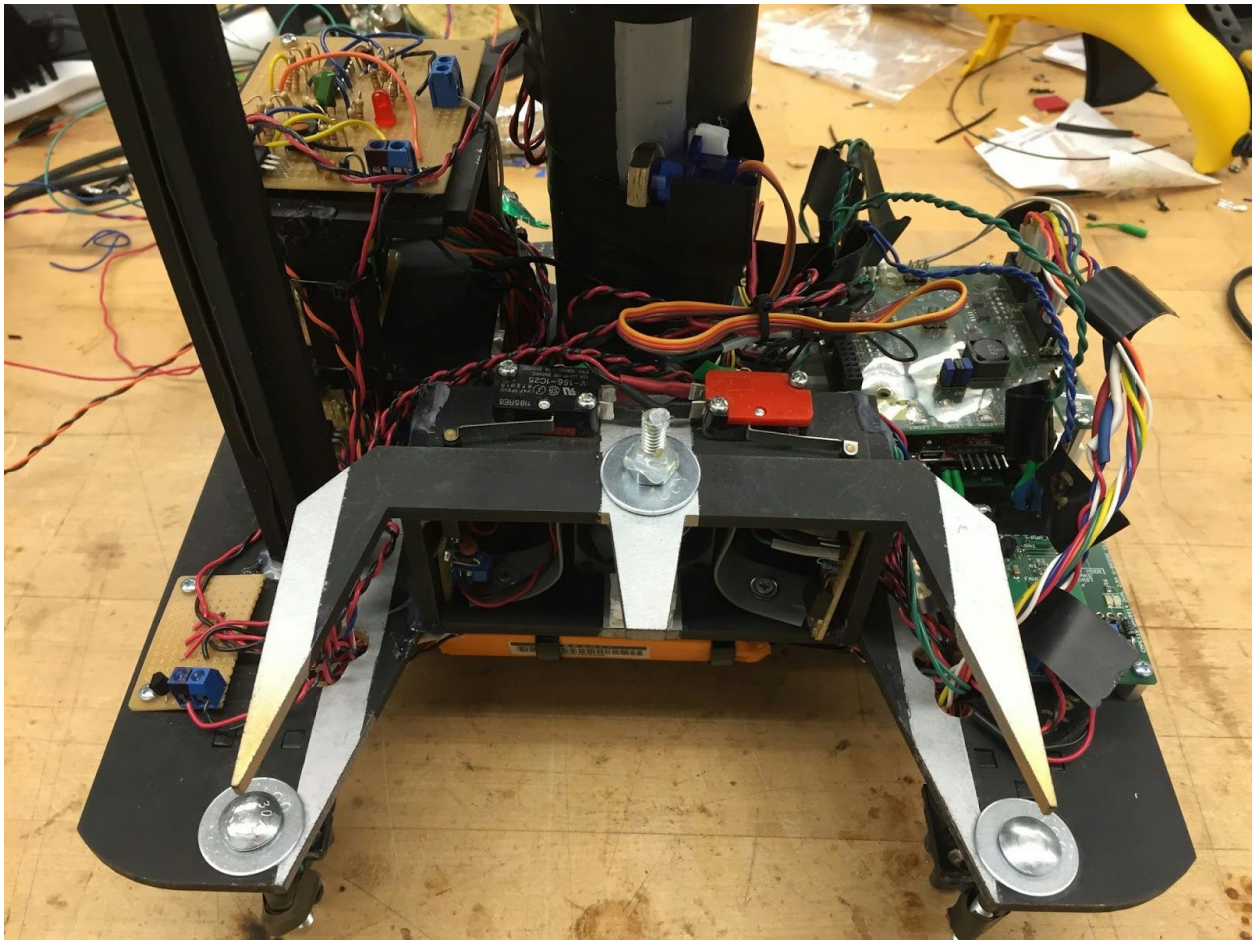


Fully constructed, check off ready robot. Appropriately named TLAR Binks.



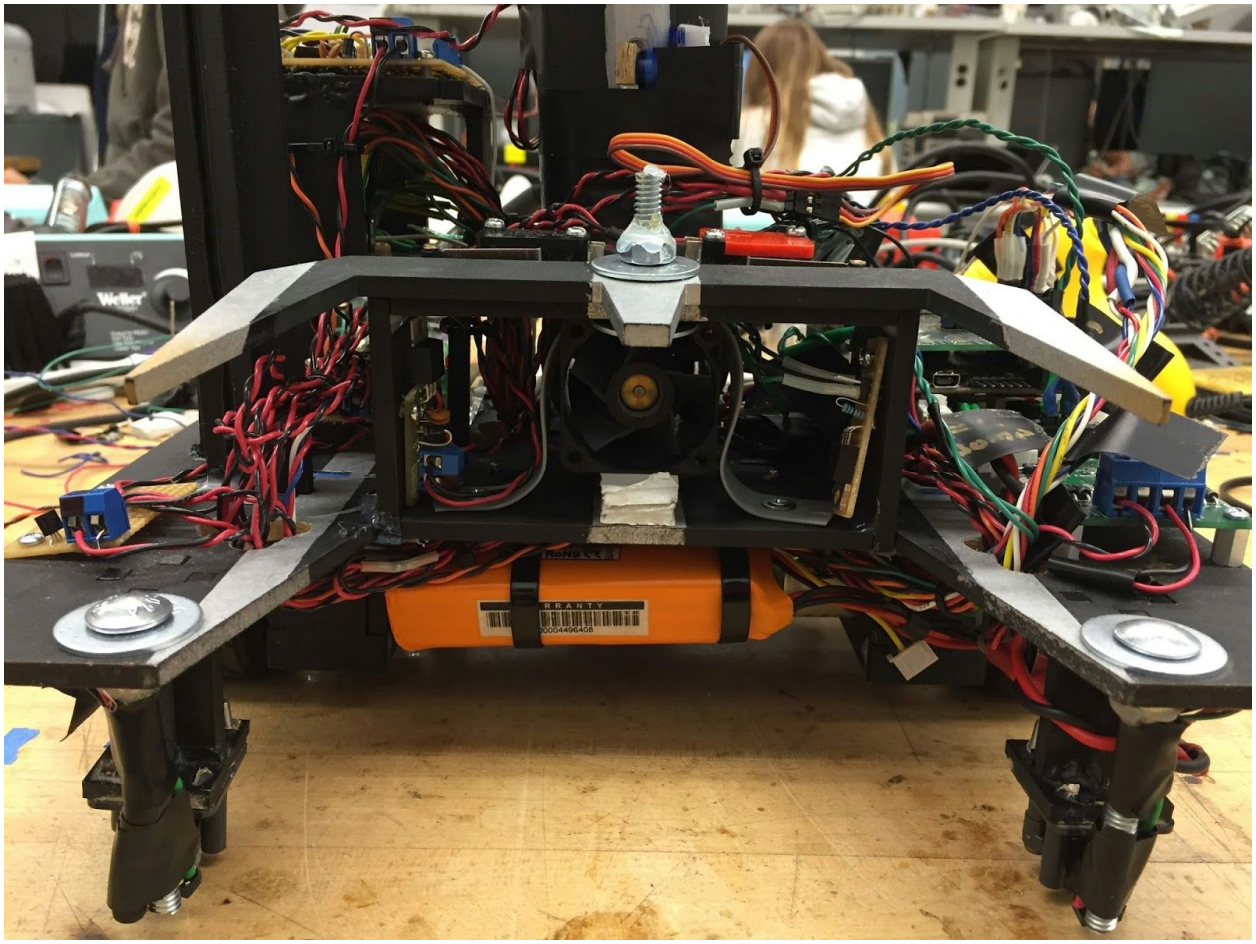
Our front bumper is being held on by one bolt with two bump sensors on each side. This allows us to easily detect front and angled bumps, with very good feedback from the bump sensors.





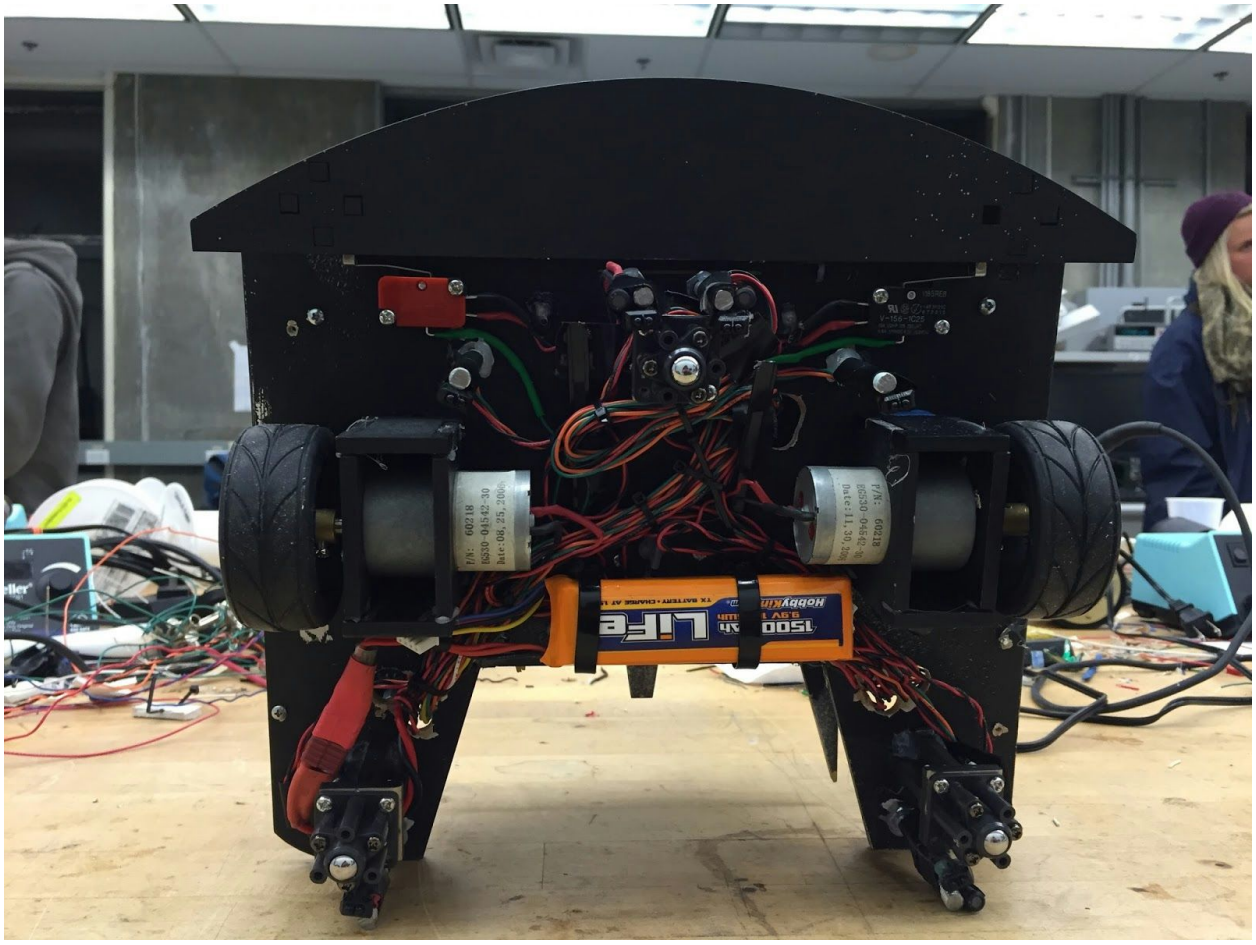
We are very proud of our back bumper, because of how well it functions on the robot. It not only acts as the plunger to get the ammo from the tower, it also work in conjunction with our base and helps align the robot to the ammo tower when we are backing up onto it. This design gives us the greatest amount of freedom (margin of error) to where even though we do not square up with the ammo dump perfectly, the back bumper catches the ammo tower and force aligns itself every time, with the help of our powerful motors. We use the two bump sensors on the back, so we know when we fully line up and ran into the ammo tower. If both bump sensors does not trip when we reverse into the tower, we know we did not hit the tower square on and may not have gotten all the ammo. We then have the bot go forward a little and reverse back into the tower in another attempt to get the ammo. This saved us many times on the ammo tower that lied on the joint of the two sides of the field.

Check out the video below in the links section, in which our robot properly adjusts to acquire all the ammo.

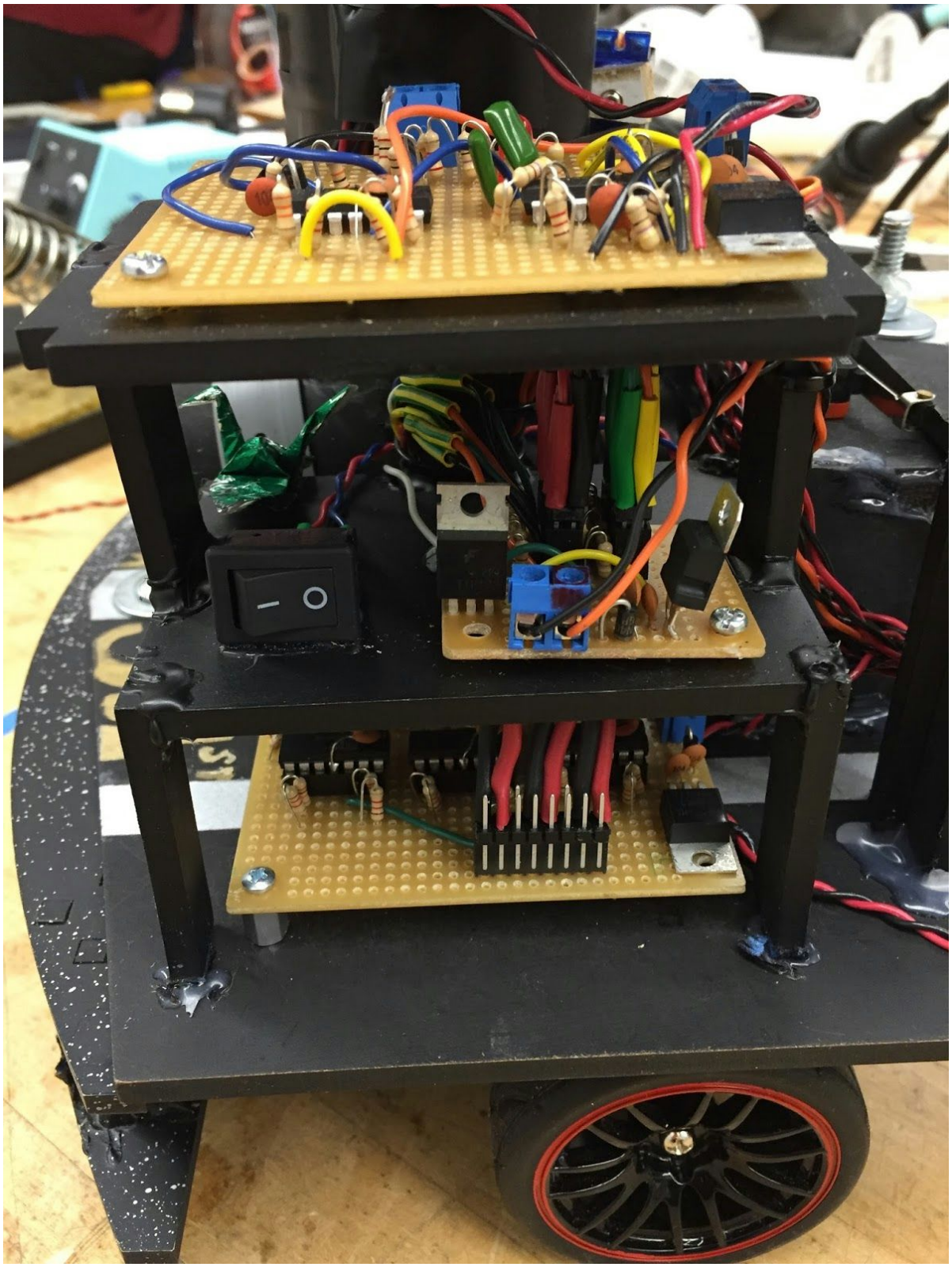


We put the battery as low as we could to keep the center of gravity low on the bot.

We use a servo motor with an arm to drop the ping pong balls to have the fan blow them out. A soccer pylon worked perfectly because it is wide and short, covering the most area while not taking up a lot of vertical room.



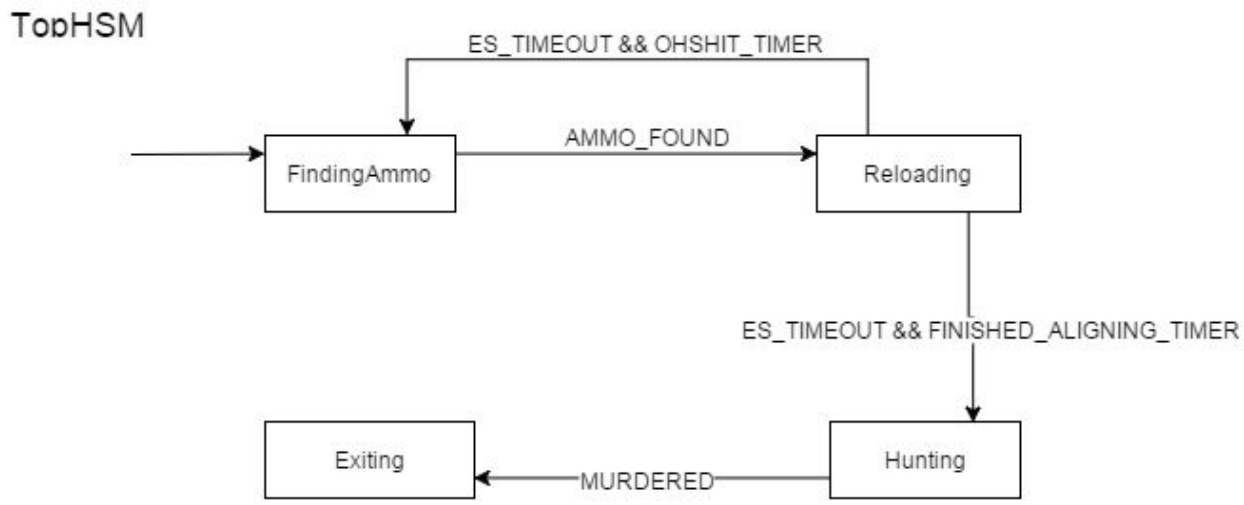
Because we had such a wide bot, we used three ball casters to keep it sturdy. Along with keeping the wheels as far apart from each other as we could. The track wire detector we used is in the bottom right of the picture on the threaded screw.



Our electronics shelf, starting from the top going down: beacon detector, tape detectors (robot on/off switch next to it), track wire detectors.

State Machine:

TopHSM



State:

FindingAmmo - INIT

SubHSM:

FindingAmmoSubHSM

Previous State(s):

Reloading

Next State(s):

Reloading

Entry:

Exit:

Events/Actions:

AMMO_FOUND

The ammo tower was found, transition to Reloading.

State:

Reloading

SubHSM:

ReloadingSubHSM

Previous State(s):

FindingAmmo

Next State(s):

FindingAmmo
Hunting

Entry:

Stop OHSHIT_TIMER started from state within FindingAmmo

Exit:

Events/Actions:

ES_TIMEOUT

EventParam == OHSHIT_TIMER

Transition back to FindingAmmo, something went wrong while trying to reload.

EventParam == FINISHED_ALIGNING_TIMER

Balls are confirmed loaded, transition to Hunting.

State:

Hunting

SubHSM:

HuntingSubHSM

Previous State(s):

Reloading

Next State(s):

Exiting

Entry:

Exit:

Events/Actions:

MURDERED

All balls have been fired, the bot assumes hits, and begins to search for the exit by transitioning to Exiting.

State:

Exiting

SubHSM

ExitingSubHSM

State:

Exiting

SubHSM:

ExitingSubHSM

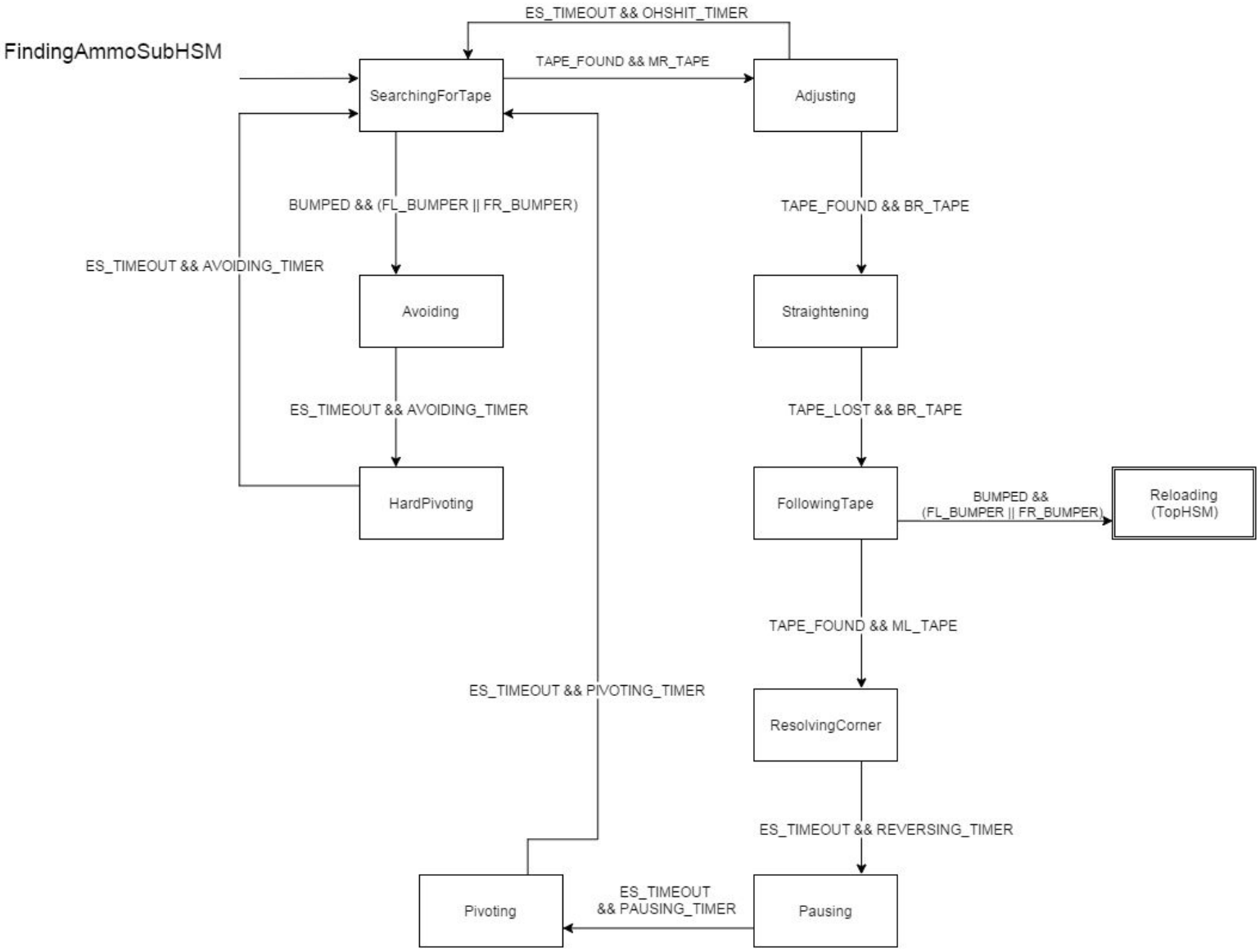
Previous State(s):

Hunting

Next State(s):

Entry:
Exit:
Events/Actions:

FindingAmmoSubHSM



State:
SearchingForTape - INIT
SubHSM:
Previous State(s):

Pivoting
Adjusting

Next State(s):

Adjusting
Avoiding

Entry:

Start Spiral Timer, used to increase the radius of our spiral.

Exit:

Stop the motors.
Reset the spiral counter.

Events/Actions:

ES_TIMEOUT

EventParam == SPIRAL_TIMER

Every time the spiral timer finishes, the bot increases the speed of the left wheel in order to create a growing spiral with the path.

TAPE_FOUND

EventParam == MR_TAPE

Assume the bot has found edge tape with the middle right tape sensor, and try to straighten up on it.
Transition to Adjusting.

BUMPED

EventParam == FL_BUMPER || FR_BUMPER

Assume the bot has collided with a roach or the center obstacle, and get away from it.
Transition to Avoiding.

State:

Avoiding

SubHSM:

Previous State(s):

SearchingForTape

Next State(s):

HardPivoting

Entry:

Drive backwards to get away from whatever caused bump event in SearchingForTape.

Exit:

Stop the motors.

Events/Actions:

ES_TIMEOUT

EventParam == AVOIDING_TIMER

After driving back for X amount of time, the bot stops and transitions to HardPivoting.

State:

HardPivoting

SubHSM:

Previous State(s):

Avoiding

Next State(s):

SearchingForTape

Entry:

Tank turn left so that when the bot re-enters SearchingForTape, it is on a new path that should avoid what caused the previous bump.

Exit:

Stop the motors.

Events/Actions:

ES_TIMEOUT

EventParam == AVOIDING_TIMER

After tank turning left for X amount of time, the bot will begin its spiral course in SearchingForTape. Transition to SearchingForTape.

State:

Adjusting

SubHSM:

Previous State(s):

SearchingForTape

Next State(s):

Straightening

SearchingForTape

Entry:

The bot will tank turn left to better align on the edge tape before following.

Begin the ohshit timer in the event something goes wrong.

Exit:

Stop the motors.

Stop the ohshit timer.

Events/Actions:

TAPE_FOUND

EventParam == BR_TAPE

Once the Back Right tape finds tape we are roughly aligned with the tape for tape following. Transition to straightening for final adjustments.

ES_TIMEOUT

EventParam == OHSHIT_TIMER

In the event of the back right tape sensor never finding tape, such as mistaking a T for edge tape, the ohshit timer will kick us back to our SearchingForTape spiral. Transition to SearchingForTape.

State:

Straightening

SubHSM:

Previous State(s):

Adjusting

Next State(s):

FollowingTape

Entry:

Tank turn left at a slightly slower turn rate than that of Adjusting.

Exit:

Stop the motors.

Events/Actions:

TAPE_LOST

EventParam == BR_TAPE

When the back right tape sensor loses tape, the bot is straightened up on the tape and ready to follow it. Transition to FollowingTape.

State:

FollowingTape

SubHSM:

Previous State(s):

Straightening

Next State(s):

ResolvingCorner

SearchingForTape

Entry:

From straightening, we assume we are straight on the tape (which was a safe assumption), and drive straight forward.

Exit:

Stop the motors.

Events/Actions:

TAPE_FOUND

EventParam == ML_TAPE

If the middle left tape sensors finds tape, we assume we have hit a corner. Transition to ResolvingCorner.

BUMPED

EventParam == FL_BUMPER || FR_BUMPER

If the bot were to encounter a front bump event, we assume we have found an ammo tower (note: even if we found a roach, we have ways out of that situation). Transition back to SearchingForTape just to reset the FindingAmmoSubHSM init state. Post an AMMO_FOUND event to the TopHSM, which will send the bot to the Reloading top state.

State:

ResolvingCorner

SubHSM:

Previous State(s):

FollowingTape

Next State(s):

Pausing

Entry:

Set the course of the bot on a curved path backwards such that the bot's rear is pointing towards the center obstacle. Start the timer for how long the bot will reverse for.

Exit:

Stop the motors.

Events/Actions:

ES_TIMEOUT

EventParam == REVERSING_TIMER

Once the bot has reversed for X amount of time, we pause for a brief moment in the Pausing state. Transition to Pausing.

State:

Pausing

SubHSM:

Previous State(s):

ResolvingCorner

Next State(s):

Pivoting

Entry:

Start the timer for how long the bot will be paused for.

Exit:

Events/Actions:

ES_TIMEOUT

EventParam == PAUSING_TIMER

Once the pausing timer expires, we want to pivot out our nose to get it facing the correct direction. The reason for the pause comes from our H-Bridge occasionally shutting down when we would transition to quickly from reversing to forward driving. Transition to Pivoting.

State:

Pivoting

SubHSM:

Previous State(s):

Pausing

Next State(s):

SearchingForTape

Entry:

Pivot on the left wheel forward. Start the timer for how long the bot pivots.

Exit:

Stop the motors.

Events/Actions:

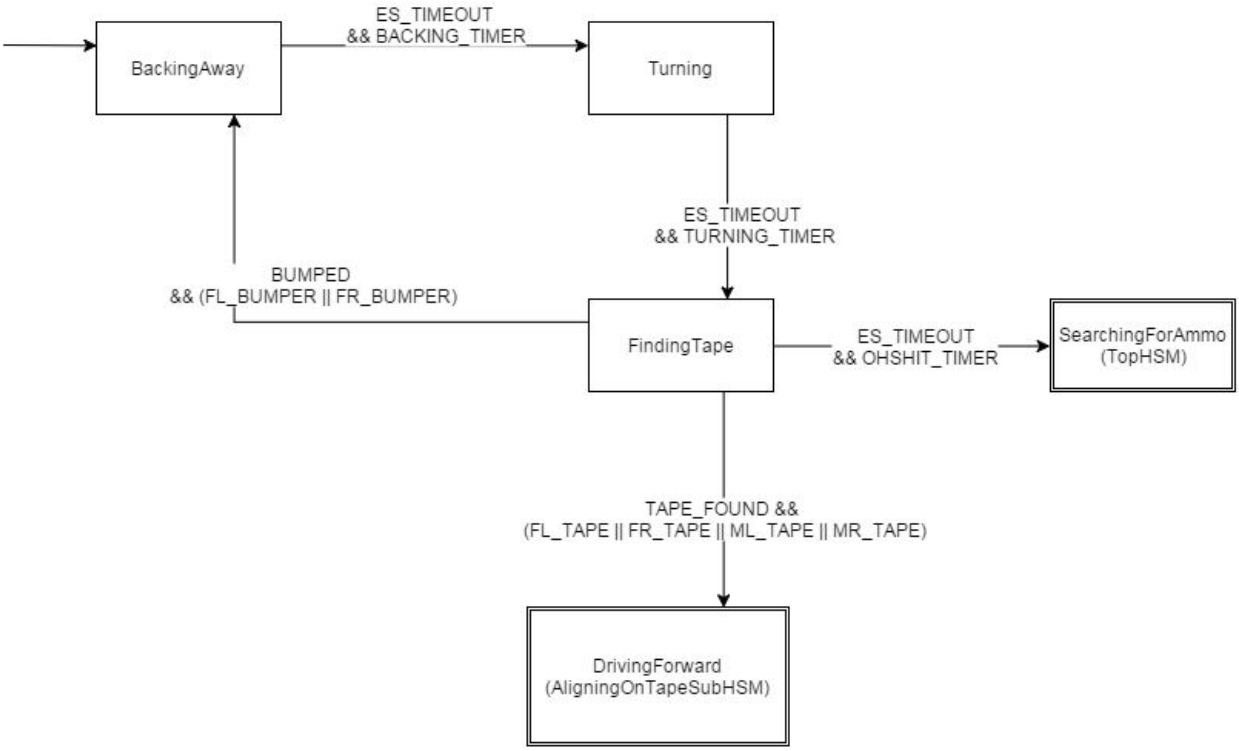
ES_TIMEOUT

EventParam == PIVOTING_TIMER

Once the bot finishes pivoting, we return to SearchingForTape. The logic behind this is that our bot will be at an angle where when it tries to find tape again, it will find the other edge's tape (not the tape that brought it to the corner), align on it, and drive forward until a tower is found.

ReloadingSubHSM

ReloadingSubHSM



State:

BackingAway - INIT

SubHSM:

Previous State(s):

FindingTape

Next State(s):

Turning

Entry:

Start the bot reversing from the tower that caused the bump. Start the timer for how long the bot will reverse.

Exit:

Stop the motors.

Events/Actions:

ES_TIMEOUT

EventParam == BACKING_TIMER

Once the bot has backed away from the tower, it will turn out so it may re approach the tower on a curved path. Transition to Turning.

State:

Turning

SubHSM:

Previous State(s):

BackingAway

Next State(s):

FindingTape

Entry:

Start tank turning to the left. Start the timer to tell it when the bot has turned enough.

Exit:

Stop the motors.

Events/Actions:

ES_TIMEOUT

EventParam == TURNING_TIMER

Once the bot has turned out roughly 90 degrees to the left, it will drive on a curved path to find the tape on a short tower or another wall for the long wall tower. Transition to FindingTape.

State:

FindingTape

SubHSM:

Previous State(s):

Turning

Next State(s):

BackingAway

AligningOnTape

Entry:

The bot will begin to drive forward on a curved right path. Based on how far back we reversed, the curve will allow us to hit the T or another wall almost perpendicular. Start the ohshit timer in the event of not finding a bump nor tape.

Exit:

Stop the motors. Stop the ohshit timer.

Events/Actions:

BUMPED

EventParam == (FL_BUMPER || FR_BUMPER)

If the bot receives another bump, the bot is at a long wall ammo tower, and will restart the process of backing up, turning, and coming at it again. Transition to BackingAway.

TAPE_FOUND

EventParam == (FL_TAPE || FR_TAPE || ML_TAPE || MR_TAPE)

If the bot finds tape with any of the front tape sensors, it will attempt to align with the tape. Transition to AligningOnTape.

State:

AligningOnTape

SubHSM:

AligningOnTapeSubHSM

Previous State(s):

FindingTape

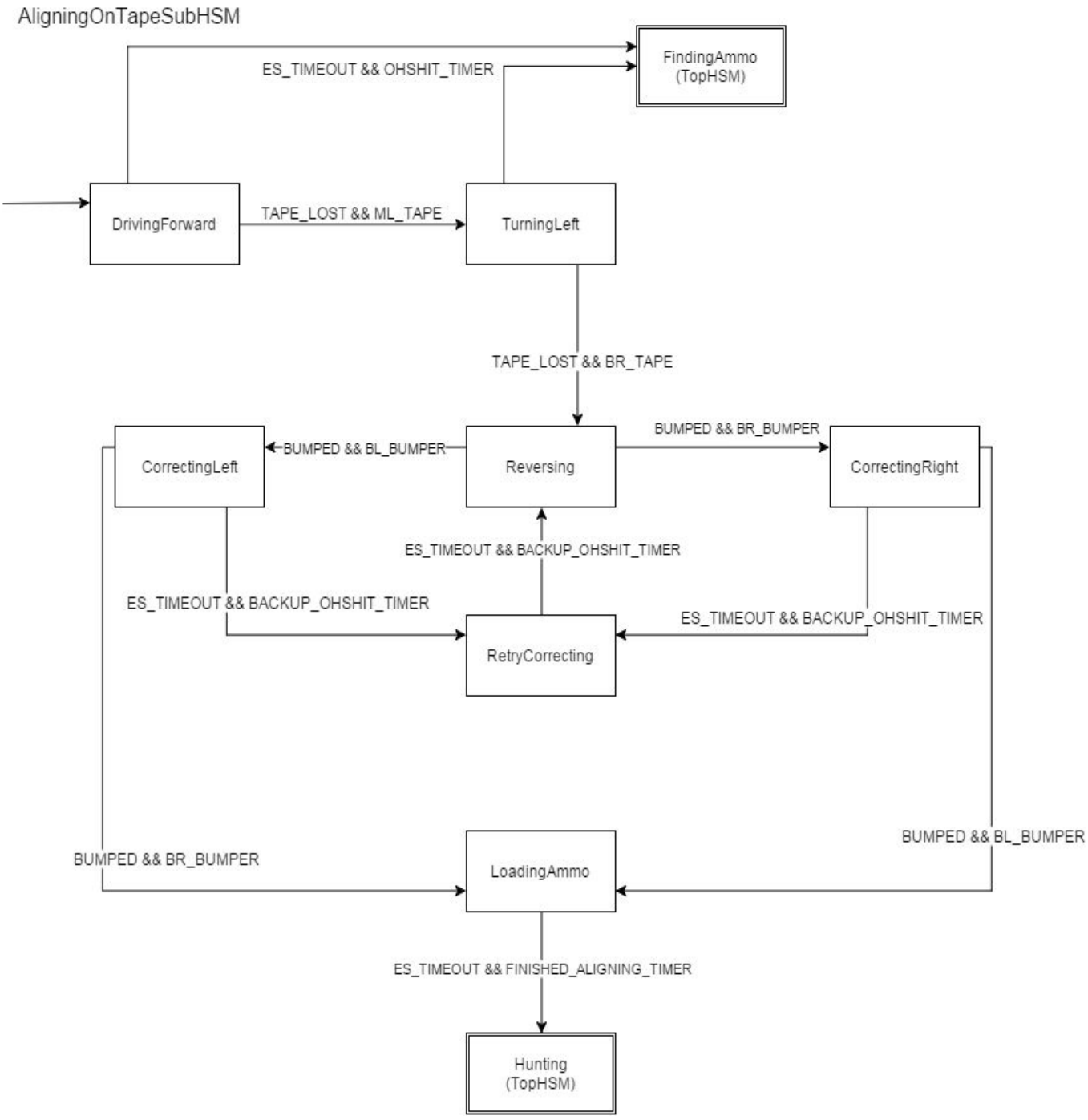
Next State(s):

Entry:

Exit:

Events/Actions:

AligningOnTapeSubHSM



State:

DrivingForward - INIT

SubHSM:

Previous State(s):

Next State(s):

TurningLeft

Entry:

The bot will drive slowly forward as it attempts to lose tape with one of its rear sensors. Start the ohshit timer in the event of it neven finding the tape with the back sensor to begin with. If the ohshit timer goes off, we go all the way back to SearchingForAmmo on the top level.

Exit:

Stop the motors. Stop the ohshit timer.

Events/Actions:

TAPE_LOST

EventParam == ML_TAPE

When the bot loses the middle left tape, it roughly has its central pivot point above the T. Transition to TurningLeft.

State:

TurningLeft

SubHSM:

Previous State(s):

DrivingForward

Next State(s):

Reversing

Entry:

Tank turn left slowly as the bot attempts to straighten up on the T. Start the ohshit timer

in the event of never losing the back right tape sensor, such as the bot is up against a corner that will not allow the bot to move.

Exit:

Stop the motors. Stop the ohshit timer.

Events/Actions:

TAPE_LOST

EventParam == BR_TAPE

Similar to finding edge tape and straightening up on it to drive forward, the bot uses its back right tape sensor to detect when it loses tape. Upon losing it, it will attempt to reverse to collect balls. Transition to Reversing.

State:

Reversing

SubHSM:

Previous State(s):

TurningLeft

Next State(s):

CorrectingLeft

CorrectingRight

Entry:

Drive straight backwards to collect balls.

Exit:

Stop the motors.

Events/Actions:

BUMPED

EventParam == BL_BUMPER

If the back left bumper is triggered, the bot came in at an angle and needs to straighten up on the tower before confirms that ammo is loaded. Transition to CorrectingLeft

EventParam == BR_BUMPER

If the back right bumper is triggered, the bot came in at an angle and needs to correct before advancing in the state machine. Transition to CorrectingRight.

State:

CorrectingLeft

SubHSM:

Previous State(s):

Reversing

Next State(s):

RetryCorrecting

LoadingAmmo

Entry:

Drive backwards, pivoting on the left wheel to properly align on the tower. Start the ohshit backup timer.

Exit:

Stop the motors.

Events/Actions:

ES_TIMEOUT

EventParam == BACKUP_OHSHIT_TIMER

If the backup ohshit timer goes off, it means the bot has yet to align on the tower and something is wrong. Transition to RetryCorrecting.

BUMPED

EventParam == BR_BUMPER

If the back right bumper is triggered, the bot has successfully aligned on the tower. Transition to LoadingAmmo.

State:

CorrectingRight

SubHSM:

Previous State(s):

Reversing

Next State(s):

RetryCorrecting

LoadingAmmo

Entry:

Drive backwards, pivoting on the right wheel to properly align on the tower. Start the ohshit backup timer.

Exit:

Stop the motors.

Events/Actions:

ES_TIMEOUT

EventParam == BACKUP_OHSHIT_TIMER

If the backup ohshit timer goes off, it means the bot has yet to align on the tower and something is wrong. Transition to RetyCorrecting.

BUMPED

EventParam == BL_BUMPER

If the back left bumper is triggered, the bot has successfully aligned on the tower. Transition to LoadingAmmo.

State:

RetryCorrecting

SubHSM:

Previous State(s):

CorrectingLeft

CorrectingRight

Next State(s):

Reversing

Entry:

Drive forward for a short amount of time before reversing onto the tower. Start the realigning timer.

Exit:

Stop the motors.

Events/Actions:

ES_TIMEOUT

EventParam == REALIGNING_TIMER

After driving forward a bit, go back to reversing. Transition to Reversing.

State:

LoadingAmmo

SubHSM:

Previous State(s):

CorrectingLeft

CorrectingRight

Next State(s):

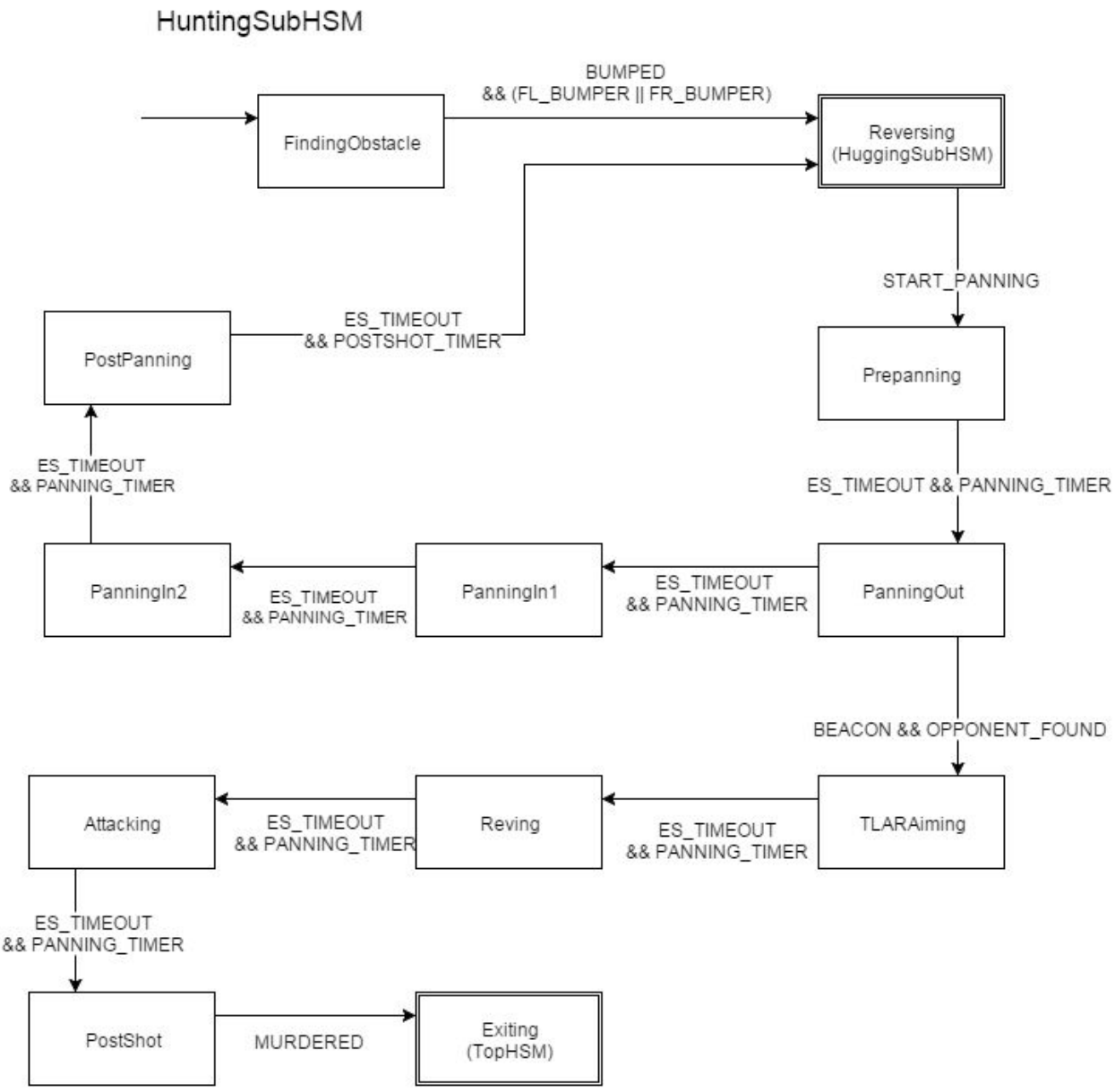
Entry:

Start the very short timer FINISHED_ALIGNING_TIMER that is pushed to the top level and changes the states to Hunting.

Exit:

Events/Actions:

HuntingSubHSM



State:

FindingObstacle - INIT

SubHSM:

Previous State(s):

Next State(s):

Hugging

Entry:

Drive forward.

Exit:

Stop the motors.

Events/Actions:

BUMPED

EventParam == (FL_BUMPER || FR_BUMPER)

Upon receiving a front bump event, the bot has hit the center obstacle. Transition to Hugging.

State:

Hugging

SubHSM:

HuggingSubHSM

Previous State(s):

FindingObstacle

Next State(s):

Prepanning

Entry:

Start the panning timer.

Exit:

Stop the motors.

Events/Actions:

START_PANNING

After X bump events within hugging, the bot receives a START_PANNING event. Transition to Prepanning.

State:

Prepanning

SubHSM:

Previous State(s):

Hugging

Next State(s):

PanningOut

Entry:

Start the panning timer. Pivot backwards on the right wheel.

Exit:

Stop the motors.

Events/Actions:

ES_TIMEOUT

EventParam == PANNING_TIMER

After pivoting away from the wall a bit, the bot will pan out to search for the opponent. Transition to PanningOut.

State:

PanningOut

SubHSM:

Previous State(s):

Prepanning

Next State(s):

PanningIn1

TLARAiming

Entry:

Tank turn left in search of the opponent. Start the panning timer.

Exit:

Stop the motors.

Events/Actions:

ES_TIMEOUT

EventParam == PANNING_TIMER

If no enemy was found, undo the panning that was done. Transition to PanningIn1.

BEACON

EventParam == OPPONENT_FOUND

If the bot detects the opponent, the bot slightly adjusts its aiming. Transition to TLARAiming.

State:

PanningIn1

SubHSM:

Previous State(s):

PanningOut

Next State(s):

PanningIn2

Entry:

Tank turn right. Start the panning timer.

Exit:

Stop the motors.

Events/Actions:

ES_TIMEOUT

EventParam == PANNING_TIMER

After undoing the majority of the panning out, pan in just a bit more. Transition to PanningIn2.

State:

PanningIn2

SubHSM:

Previous State(s):

PanningIn1

Next State(s):

PostPanning

Entry:

Pivot on the right wheel forward. Start the panning timer.

Exit:

Stop the motors.

Events/Actions:

ES_TIMEOUT

EventParam == PANNING_TIMER

To reverse the pre-panning process, the bot pans in on the right wheel pivot. Transition to PostPanning.

State:

PostPanning

SubHSM:

Previous State(s):

PanningIn2

Next State(s):

Hugging

Entry:

Tank turn right. Star the postpanning timer.

Exit:

Stop the motors.

Events/Actions:

ES_TIMEOUT

EventParam == POSTPANNING_TIMER

The bot does a sharp tank turn right to adjust for the slight pivot left that occurs when entering Hugging. Transition to Hugging.

State:

TLARAiming

SubHSM:

Previous State(s):

PanningOut

Next State(s):

Reving

Entry:

Tank turn right. Start the panning timer to tell the bot when to stop.

Exit:

Stop the motors.

Events/Actions:

ES_TIMEOUT

EventParam == PANNING_TIMER

The bot would often overshoot the opponent, so to counter that the bot will do a quick and sharp pivot to the right to score a direct hit. Transition to Reving.

State:

Reving

SubHSM:

Previous State(s):

TLARAiming

Next State(s):

Attacking

Entry:

Run the fans. Start the Revving timer.

Exit:

Events/Actions:

ES_TIMEOUT

EventParam == REVVING_TIMER

After revving the fan for a short amount of time to get it up to speed, the bot is ready to attack. Transition to Attacking.

State:

Attacking

SubHSM:

Previous State(s):

Reving

Next State(s):

Entry:

Drop the servo arm holding the ping pong balls. Start the shot timer.

Exit:

Stop the fan. Reset the servo arm.

Events/Actions:

ES_TIMEOUT

EventParam == SHOT_TIMER

After waiting a short time for shooting the balls, the bot needs to adjust its direction to properly hit the center object.

State:

PostShot

SubHSM:

Previous State(s):

Attacking

Next State(s):

Entry:

Tank turn right. Start the post shot timer.

Exit:

Stop the motors.

Events/Actions:

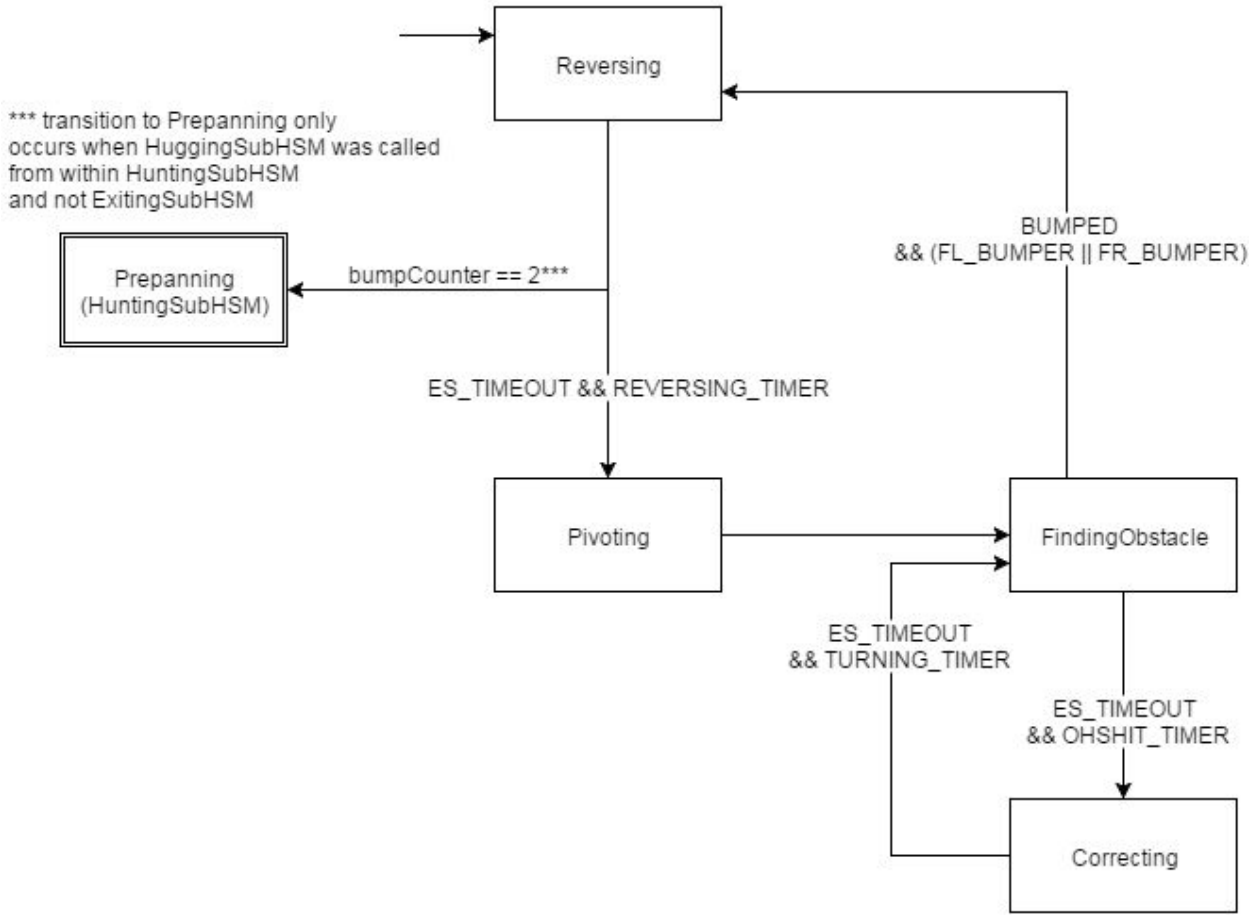
ES_TIMEOUT

EventParam == POSTSHOT_TIMER

After adjusting slightly post shooting, post a MURDERED event to the top level to signal the balls have been fired and the opponent is dead.

HuggingSubHSM

HuggingSubHSM



State:

Reversing - INIT

SubHSM:

Previous State(s):

FindingObstacle

Next State(s):

Pivoting

Entry:

Drive backwards on a curve. Start the reversing timer.

Exit:

Stop the motors. Stop the reversing timer.

If the number of bumps from FindingObstacle is equal to two, the bot posts a START_PANNING event for the HuntingSubHSM.

Events/Actions:

ES_TIMEOUT

EventParam == REVERSING_TIMER

After reversing for a short while to get away from the wall and ensure the bot will not clip part of it while it tries to turn, the pivots out hard. Transition to Pivoting.

State:

Pivoting

SubHSM:

Previous State(s):

Reversing

Next State(s):

FindingObstacle

Entry:

Tank turn left. Start the turning timer.

Exit:

Stop the motors. Stop the turning timer.

Events/Actions:

ES_TIMEOUT

EventParam == TURNING_TIMER

After turning out, the bot returns to trying to find the center obstacle. Transition to FindingObstacle.

State:

FindingObstacle

SubHSM:

Previous State(s):

Pivoting

Correcting

Next State(s):

Reversing

Correcting

Entry:

Drive forward on a curved path in search of center obstacle. Start ohshit timer.

Exit:

Stop the motors.

Events/Actions:

BUMPED

EventParam == (FL_BUMPER || FR_BUMPER)

If a bump is detected, the bot assumes it found the center obstacle and will reverse away from it. The bump counter is incremented. After 2 bumps, we assume the bot has made it to a different side of the center obstacle and it should pan again for the enemy. Transition to Reversing.

ES_TIMEOUT

EventParam == OHSHIT_TIMER

The ohshit timer will only trigger is a bump event has not occurred for a significant amount of time. The bot has most likely encountered a sharp corner of the center obstacle. Transition to Correcting.

State:

Correcting

SubHSM:

Previous State(s):

FindingObstacle

Next State(s):

FindingObstacle

Entry:

Tank turn right. Start the turning timer.

Exit:

Stop the motors.

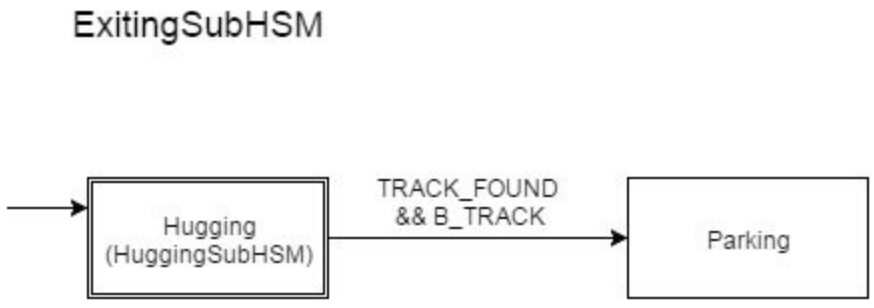
Events/Actions:

ES_TIMEOUT

EventParam == TURNING_TIMER

After a short sharp pivot to the right, the bot can assume it will drive forward and encounter the center obstacle again, instead of missing it. Transition to FindingObstacle.

ExitingSubHSM



State:

Hugging - INIT

SubHSM:

HuggingSubHSM

Previous State(s):

Next State(s):

Parking

Entry:

Exit:

Stop the motors.

Events/Actions:

TRACK_FOUND

EventParam == B_TRACK

If the back track wire sensor is tripped, the bot assumes it is inside the exit portal. Transition to Parking.

State:

Parking

SubHSM:

Previous State(s):

Next State(s):

Entry:

Exit:

Events/Actions:

Reflection

Each member of the 501st can agree that the Mechatronics final project was one of the most stressful projects any of us have ever worked on. Initially, the team started out with very high morale and dreams of hitting Beer Checkoff. The team worked long hours from day one, planning and designing the perfect robot. After about a week of long hours and tough planning, Josh and Ian began to get sick. For a solid week and a half, 2/3rds of the group was sick. The illness lead to constantly being tired, physically and emotionally towards the project and each other. Morale quickly dropped off as we were sick. The group project log Ian started in the begining stopped being updated, and the group was ready to snap each other's' necks and had given up on Beer Checkoff. Once people started to get better, we sat down and reevaluated how the project was going. We all agreed that things needed to change in order to finish the project before Thanksgiving. The group then worked hard and diligently, putting in many long hours and little sleep. During this time the bot was finishing up being assembled and the code was in the process of being written. The 501st were unable to meet Beer Challenge, but were able to take a solid 4-5 days off for Thanksgiving break. Upon returning, we realized that victory

may have been called a bit early, as we were required to put in a solid 3 days of long hours of work in order to check off. Finally, with only 30 minutes remaining of the last day for “On Time” checkoff, the 501st were finally checked off by Max L. The total amount of hours averaged 8-10 a day per person, start November 2nd to December 1st; totaling 232-290 hours a person.

Links

Code
Solidworks
Bill of Materials
Gantt Chart
Bot Reloading
Check Off Run
Competition Run