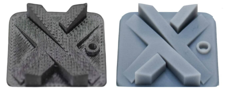
Autonomous Soccer Robot

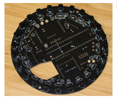
By: Naveen Enock

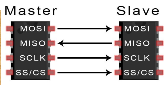
Abstract

These Lightweight robots are equipped with omnidirectional camera, kicker, dual layered omni wheels, and fmy Maxon motors. Utilizing custom 3-D printed parts, the robot achieves both a lightweight and secure framework. Data from line sensors on the custom line PCB are used to ensure the robot stays within the defined boundaries of the field. Using infrared sensors the robot is able to detect and follow the ball using the optimal path, made possible by path optimization software. Finally, a refined bluetooth connection between both robots allows for smooth transitions between offensive and defensive roles, allowing for tactical play.



Main PCB
The main PCB contains 24 ball sensors placed at 15 degree increments. Each sensor is also paired with its own low pass filter to convert the digital signal from the ir receiver to analog values that the adc can read. It also contains my main microcontroller the teensy 4.1, openMV cam, motor driver ports and the xbee for remote communication.

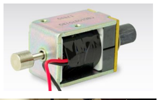


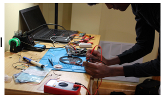


Manufacturing and Design

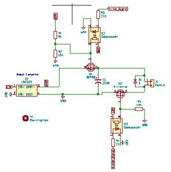
PCBs were designed using KiCad software. Because some of my parts were uncommon I created my own symbols and footprints within kiCad to be able to use these components with my design. I would then test different parts of these boards to make sure everything works correctly. In The first iteration of my board I used 3.5mm terminal blocks to connect the battery cable to my main PCB. But I found that after constant unplugging and replugging of the battery connector the cable would slowly pull out of the terminal blocks and sometimes create shorts. In my next version I made sure to have a metal contact where I could directly solder the wires for more secure attachment

All of my pcbs are placed in a very easily accessible location on both the top and bottom of the pcb. These pcb's are removable by simply unscrewing 4 screws and fully detachable as both the pcbs are connected through a very easy to remove idc cable.





Both my robots have a kick functionality to make scoring goals against enemy defenders easier. my kick is formed through a open frame solenoid that requires very high voltage for strong kicks. I found that 55 volts seemed optimal for staying within the kicker limit. To be able to control this voltage with my teensy 4.1 I used a series of mosfets that were connected to optocouplers to be able to control the kicker. I even have a protection mosfet that disconnects the rest of the robot from the kicker when the kicker is enables. This ensures that the solenoid does not draw too much current from the battery.



For my custom designed hyperbolic mirror I used mirror foil that was bendable to any shape when heated to a certain temperature. I printed a mould using SLA 3d Printing and then used high grit sandpaper to smoothen the mould out. Holding the mirror sheet over the stove I would heat it up until the mirror surface became a water like surface and very rapidly placed the mirror over my mould. Unfortunately this did not work very well as even after sanding there were many small bumps and curves that corrupted the reflection. For a second try I used NylonX. I went through the same process of creating a mirror as aforementioned.

To design the custom pieces and chassis of my robot I used FDM 3-D printing using PLA material as it proved to be light and strong. Since using printing services of a company is very expensive, I used my own printer to print all my parts. First I uploaded STL files to splicer and then tuned settings for printing. Sometimes printing proved to be very difficult when I had overhangs or support dependent structures. For my final chassis I attempted to use NylonX as my printing material as it is significantly stronger and only has a density of 1g/cm^3 compared to 1.25g/cm^3 of PLA. However I found that nylonX had many warping problems and after many attempts I decided to stick with PLA.

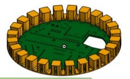
This year the primary cause of hardware problems with my robots proved to be electrical faults. Because all of the soldering was done by us, human errors were easy to make and finding these errors proved to be very difficult. The primary process for finding these errors was using a multimeter to test continuity between different pins. Sometimes a wrong placement of one part of the board would result in burning of traces and damaged components. To bypass these burned traces I would solder jumper wires to mimic the job of the trace.

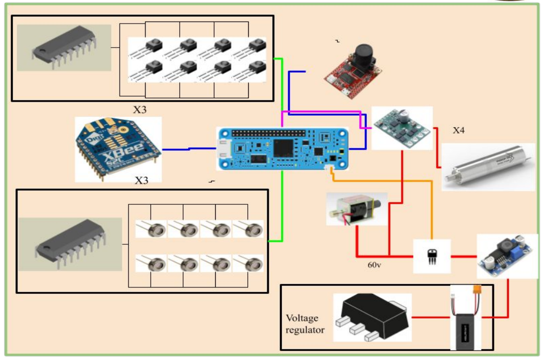
Electrical

Communication

Use of ADC
Because of the large amounts of sensor that my robot uses I decided to use the MCP3008 adc which takes in 8 analog values and the teensy is able to ask for any of these values at anytime through spi communications. MISO, MOSI, SCK and CS wires are connected between each adc. The teensy then communicates with each of these adc's to get the necessary sensor values.

Ball Sensor Shield
Through inspiration from other great teams I realized that I could make my ball sensor values even more accurate by adding a sensor shield. This shield would have to cover every ball sensor shield and add a small opening to let IR light in. The material of the covering had to be fully closed so that it would not let any light in.



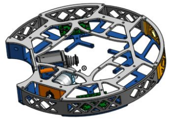


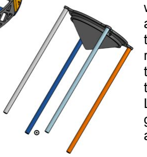
Chassis

my chassis is not only designed to hold all of my components it is also designed to take impacts and make the robot structurally very strong and at the same time keep it within weight limit.

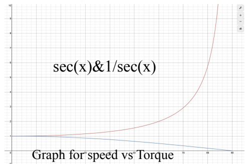
Walls
I surround the entire perimeter of the robot even around the wheels to ensure that the impacts of other robots colliding will not be taken into the wheels or motors The walls are also curved which work to distribute the shock from impacts.

Improvements
After US nationals I decided to make many large improvements to my robot for worlds. The biggest change I made was using maxon motors to replace the old pololu motors. Maxon motors. Each maxon motor is 40-45g heavier than my previous motor so I had to make my chassi thinner and remove unnecessary parts. One place I reduced weight was by using 4 poles made of standoffs instead of the heavy acrylic tube to hold up the mirror. After previous testing I also found that the line PCB was too close to the carpet and as a result gave values that were hard to work with. This led to a lot of problems with the robot being unable to stay in the box. In my improved design I made overhangs within my chassis to push the line pcb more to the inside of my robot.



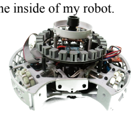


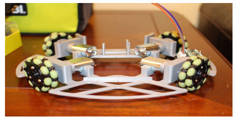
While designing my chassis one of the biggest decisions I had to make was the angle of fmy wheels in respect to the center. At first one would assume that a 45 degree angle for all the wheels would make the best Xdrive. But after doing research I realized that the angle of the x-drive is directly proportional to the speed/torque ration of the robot. Learning this I tuned my robot for greater speed by creating 40 degree angles wheels.



Graph for speed vs Torque

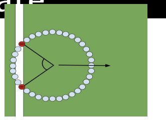
One of the biggest problems I faced was finding a way to attach plastic omni wheels to a metal d-shaped shaft. I experimented with many different methods using 3d printing adapters with their own set screws and nuts. However all of my tests concluded the same result: plastic is very flimsy and is not strong enough to hold the set screw against the motor shaft. After further research I found aluminum adapters that fit into my motors with its own set screw system. I simply created omni wheels that would be able to connect to this aluminum adapter.





Software

One of the primary criteria that the robot has to meet is staying within the boundaries of the field, which are delineated by white lines. I achieved this task mainly with the help of my custom line PCB. Positioned at the bottom of my robot, the PCB is equipped with 24 LEDs and photoresistors arranged at 15 degree increments. Using the threshold values for the respective colors, and

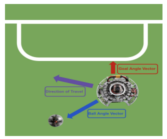


Role Switching
I use two Xbee S2C radios configured using zigbee protocol to communicate between the two robots locally. The robots exchanged their roles and switched between offense and defense depending on which robot was closer to the ball.

Orbit and Dampen Functions
Although I were able to capture the ball using my ball detection software, the orbit and dampen functions allow us to do this using the most efficient and optimal path, which may not always be a straight line. This is mainly because of a lack of a way to keep the ball in my control, as the momentum of the robot would inadvertently bump the ball out of my control. The orbit function finds the radius of the orbit around the ball, and using the current ball angle, outputs the angle needed to properly orbit the ball. When directly behind the ball, the orbit function outputs 0, as a straight path is optimal. However, in the other cases an angle will be added to the initial ball angle. This path is then further optimized using the dampen function. This function reduces the amount that the robot's path alters from its original straight path. This is important as it prevents the robot from following an unnecessarily long path. These functions together create a path that is mostly straight other than a slight smooth curve as the robot approaches the ball to maintain control.

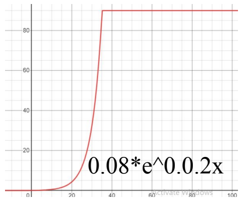
Defense
Programming defense proved to be a great challenge as there were many new constraints with the new field design. my main algorithm surrounded vector addition once again. I calculated both the ball angle and the goal angle and used the two angles to place the robot between the two. I noticed that when playing the role of defense, at times, the robot would chase the ball as opposed to following its direction. To combat this issue, I designed an algorithm which depending upon time away from line, magnitude of the ball signals (signifying distance from the robot), and vector projections to prevent excessive forward movement, I are able to have my robot reliably defend the goal.

calculation. First I read the sensor values of each ball sensor. Then I multiplied the read values with the sin(Θ) and cos(Θ). Θ being the angle of the ball sensor. After doing this I had vectors of each ball sensor. The sensor that the ball was closer to would have a larger vector pointing in that direction. To get one vector pointing to the ball I simply added all of the vectors. To convert x and y coordinates of the vectors to an angle I used the atan2() function which took in x and y as parameters and output an angle in degrees.



Camera
Color detection with the openMV was relatively easy as I were given very great example codes in the application. All I had to do was set color thresholds which are able to detect specific colors in the image. After I detected the goal I compared the x and y coordinates of the goal to the coordinates of the center of the image. Finally I compared these coordinates and once again used the atan2() function to find an angle to the goal. I were also able to find the distance the camera was relative to the goal. I found this by using focal length and comparing the size of the goal as the robot moves closer or farther away. Theoretically this works great however I found that this data was not very reliable. For this reason I only used the goal distance for very limited uses.

Goal Kicker:
The kicker of my robot was controlled based on many different parameters. The most important of these involved the data from the light-gate sensor that is placed in the ball capture zone of my robots. If the sensor was within a certain threshold it would mean that the robot had the ball. In addition the kicker would only kick after a certain amount of seconds to ensure avoidance of continuous kicks


80
60
40
20
0
20
40
60
80
100
0.08*e^0.0.2x