MASLAB Reflection: Team 12

Mechanical:

Our robot:

- We used a rubber band spindle to pick up balls, which works very well. That pulled the balls into a hopper that then fed our lift. We built a spiral lift, I recommend this as it is proven and simple. However it is not trivial to build so watch out for this. We used two pacman cut outs and a maze-like assembly to sort balls. We also used an ir sensor as a breaker beam to tell when the ball got into the pacman sorter. Sometimes the balls would get stuck if there wasn't enough time for it to settle. We would then dispense them using the second of the two pacman sorters by pushing the balls into two different delivery canals. One for the top goal and the other that was fashioned out of sheet metal for the red balls and the lower goal for the greens.

TIps:

- Get the base done as quickly as possible. I mean by the first friday you should have something that moves and has a camera mount.
- Design and build everything in solid works, use assemblies, but make sure you have a proof of concept prior to doing that. Waste as little time as possible.
- Use already developed technologies instead of developing your own if possible.
- Robust and simple, those should be your biggest design features.
- Design something that doesn't need constant attention, you don't want to have to be continually bending, grinding, or sanding parts to something to get it to work.
- Use as few moving parts as possible and make sure you have room for whatever you are picking up to be moved through your robot, but not too much room.
- Make sure you're robot won't get in its own way, meaning make it so it won't have things hanging off it. You don't want it to get stuck, because if you get stuck, you will lose.

Code: See github for the full code (https://github.com/jpmccabe/maslab_robot)

Strategy:

- When scoring, our robot's strategy was broken up by a one-minute strategy and a remaining two-minute strategy. In the first minute of a round, the robot will only score in a reactor if it has at least two green balls. This way the robot is able to score in the top and bottom port of unique reactors, giving us more points. In the remaining two minutes of the round the robot is allowed to score just one green ball in the top port of a reactor.
- The robot will score over the interface wall whenever it sees it and has red balls.
- The robot will only attempt to extract balls from the energy silo in the last half of the round. We did this because we were not sure how reliable the mechanism would be.
- Scoring in the reactor had top priority, followed by scoring over the interface wall, collecting from the energy silo, avoiding obstacles, and lastly collecting balls.

Ball Sorting:

- Our robot did not use a color sensor on the sorter, just a photogate to detect when a ball entered the sorting mechanism.
- So when the robot tries to collect a ball, it adds what we call a TimedBall to a queue. The TimedBall contains information about the color of the ball and the time it was picked up.
- This allows the sorter to know the color of the next ball and use a timeout to realize when a ball was not actually picked up.
- Any balls that are accidently picked up, which means the color is unknown, are assumed to be red. So the worst case is we accidently dump green balls over the interface wall.

Aligning With Field Objects:

- To score in a reactor or over an interface wall, or to extract a ball from the energy silo, we must align perpendicular to the field object.
- The first thing the robot does is center the object in the camera frame. This is necessary because our later angle calculations are based off of the object being centered in the screen. Centering is done with a constant motor speed and threshold.
- Once centered, if the angle between the robot direction and the perpendicular to the field object is small then the robot uses a proportional controller to just drive towards the field object.
- If the angle is large then the robot will do a Manhattan routine. This involves calculating the angle needed to turn parallel to the field object and the distance it needs to drive to be directly in front of the object. It executes these moves by turning the motors for an amount of time proportional to the angle or distance. Lastly the robot turns ninety degrees to face the field object and then uses a proportional controller to drive towards it.
- When the robot is six inches away from the a field object it goes into an insert routine which just involves driving forward for a few seconds.

Electronics:

- We only had two sensors on our robot, the webcam and a photogate sensor. The photogate sensor was used to detect balls in our sorter mechanism. The photogate sensor required a circuit with an op-amp and voltage divider to get the analog output down to the 3.3V level.
- We used three Cytron motor controllers - two for the drive motors and one for the ball lift. One Vex motor controller was used for the rubber band roller.

Machine vision:

- We had seven threads for machine vision. The main one takes frames from the camera and gives to the remaining six threads.
- These six threads process the image they are given to detect the six main objects of the competition: red balls, green balls, blue walls, yellow walls, teal walls, and purple walls.
- First we transformed the image given by the camera from BGR to HSV. Note that I said

BGR and not RGB. We spent two full days debugging my code not thinking that this could be the problem at all.

- We then tested my code several times until we were able to find the perfect HSV range of values for each of the objects' colors.
- We also used the erode and dilate opencv functions to eliminate noises. the erode basically finds the small amount of white pixels that are surrounded by black pixels and turns them black. The dilate function does the opposite
- We used the opencv functions contours and canny edges to find the edges of these blocks of white pixels that could represent the balls or the colored stripes on the walls.
- We also ignored any blocks of white pixels that could not possibly be a ball. For example the objects whom bounding rectangle width is twice the height.
- We placed the robot at some distance from the wall and then measured the height in pixels between the upper and the lower edges of the stripe. We repeated that for 5 different distances and put them on wolfram alpha and got a linear equation in which you enter pixels and it outputs a distance.
- We did the same for the balls but used the diameter in pixels instead of height.
- For Angle it's not linear so you can't just use the x and y of where the object is in the image to find the angle using trigonometry. What we did was pretty similar for what we did for finding distances. We got a bunch of data points of where the object was on the x axis and the measured angle. Then, we put this data on wolframalpha and asked for the power equation. It was pretty accurate.

Things to think about:

- Don't leave the programming to the last few days, or even the last week. It takes a lot longer than you think to build software robust enough to actually do something.
- Plan the mechanical design all at once, not a week at a time.
- NEATLY WIRE YOUR ROBOT
- Don't try to make things too sophisticated, but at least build something that is capable of doing everything.
- Use math in your code, not janky constants to calibrate things.
- Don't break your micro controllers by wiring them incorrectly. You will most likely do this however.
- You want to use things that are reliable. If the sonars are not working well, don't use them etc.
- Use timeouts in every state in your code.
- Do everything you can to prevent your robot from getting stuck on field objects. Even after all of that planning your robot will still get stuck so write code to get it unstuck.

What we learned:

- How to use Solidworks better.

- That scotch tape is very low friction.
- Debugging code takes a ridiculous amount of time.
- The webcam is the only necessary sensor.