Team One Paper

Final Paper

Overall Strategy

Our Robot is controlled by a finite state machine structure. Each state has exit conditions which are checked periodically. When an exit condition is satisfied, the robot will change state to that specified by the satisfied exit condition. The process repeats in the next state until three minutes are up. As long as we have no balls we will always be wandering until a ball is found, when we will pick it up. If we have 4 balls within the first two minutes we will transition to searching but now for a goal in which to score. It will also transition to scoring if we have a ball, and we have a minute left to try to score the balls we have. If we have no ball with a minute left we will still look for a ball, pick it up and try to score before time runs out.

As the robot initiates, it ensures the arm is in its protected position (up) and enters the wandering state machine where it will drive forward for 5 seconds unless an exit condition is fulfilled. One exit condition is if one of the IR sensors reaches a threshold value where it then will turn slightly away from the obstacle until the IR's move away from their thresholds. If both IR sensors trip we turn more substantially to completely avoid the danger. Also inside the wandering state is a spin random state where we try to change our heading a bit to avoid going to the same places over and over again.

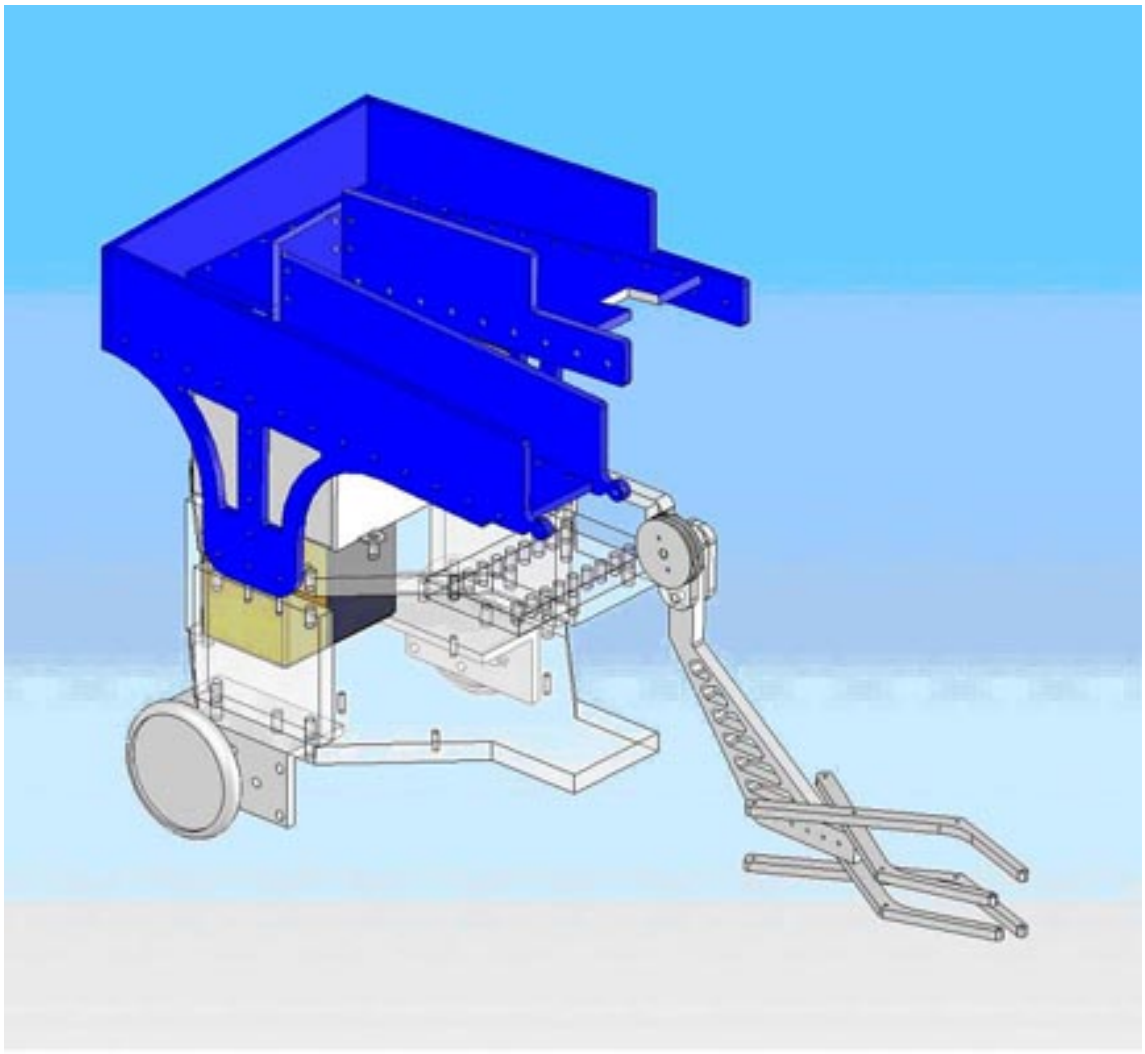
The wandering state machine will transition to the ball pickup state machine if a red ball shows up on our camera. We then use the balls horizontal position on the image and center the ball on the screen. Then move toward the ball until we are a set distance away based on the height of the camera off the ground and on screen position. As we do this we are still running a thread looking at the IR sensor reading and will avoid danger is the IR sensors trip by shimmying left or right. A shimmy is a series of movements that end up moving a robot horizontally left or right of its previous position. After a shimmy we again center the ball on screen and go in to pick it up. Once the ball pickup state has picked up a ball we then go back to wander to look for more balls or score depending on how many balls we have and time left in the round.

The other large state consists of goal scoring. When a goal has been spotted, enough balls have been collected and the time is right the robot will go after the goal. Since our arm is sensitive our only approach to goal is from an angle with the ball hopper leading forward. The robot will shimmy until the angle between the camera and the center of the goal has reached a threshold that will score and protect the arm. At which point, we ram the goal and then back up one wheel to rotate the robot and try to unload all balls. After this action the robot will return to wandering and set our ball count to zero.

Mechanical Design and Sensors

Mechanical Design

The mechanical design of the robot was divided into 3 main divisions: The chassis, the gripper arm, and the ball delivery bin.



The mechanical design proved to be somewhat insufficient. The positioning of the arm and the scoring end of the ball delivery bin proved to be such that the robot could only score if it approached the goal at a specific angle. Since it was not a simple task, much more intense code had to be written in order that the robot would be able to score in difficult situations. As it turns out, the code proved to be too difficult and the robot was unable to score when it mattered.

There were three IR sensors, two facing forward on either side and one facebackward. They were placed on the extremes of the robot (left and right) so that no walls would be able to catch parts of the robot without the IR sensors detecting them. The rear facing IR sensor was in place to make sure that the robot did not back up into any walls.

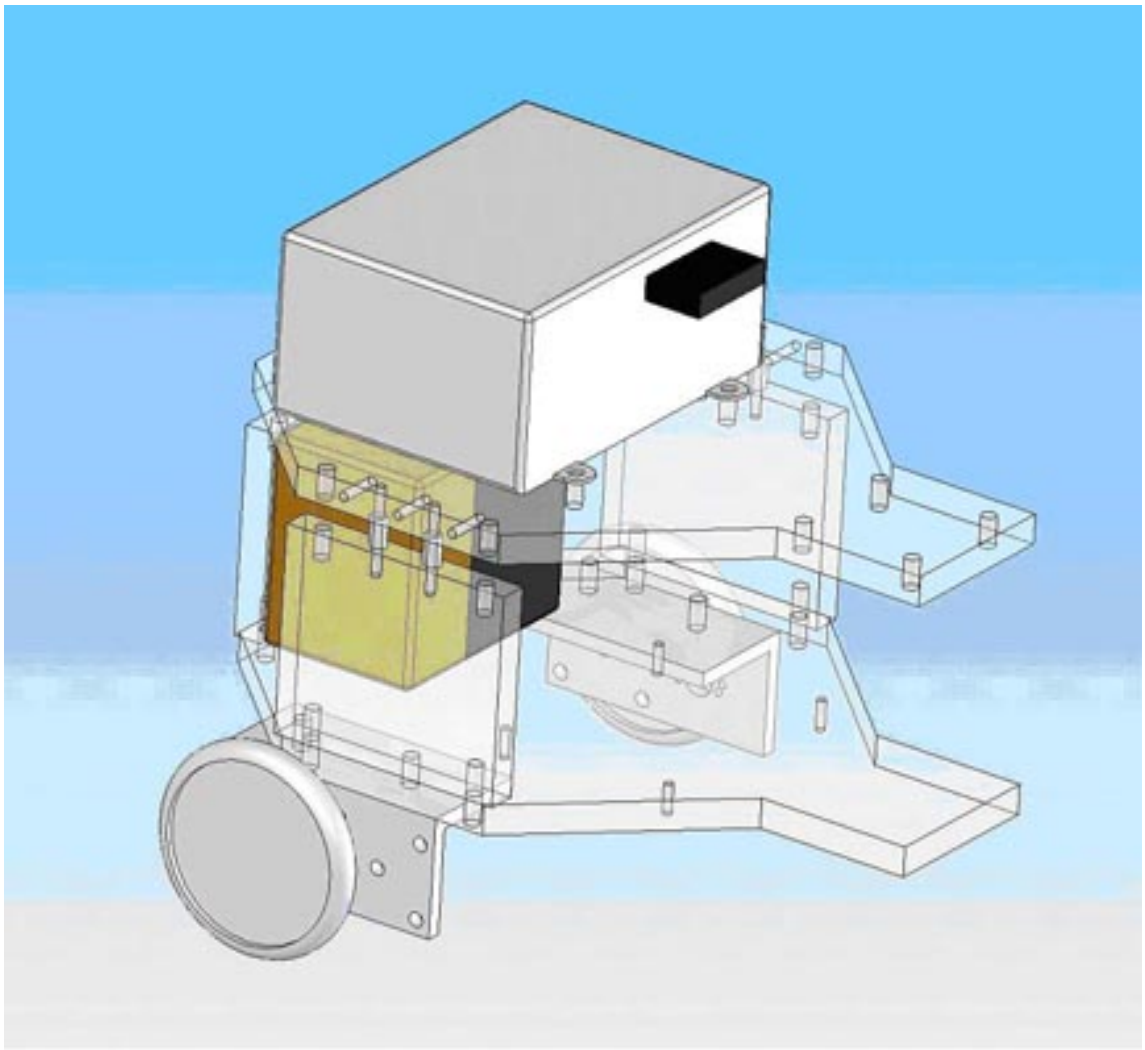
There was a touch sensor on the claw to indicate that it had fully opened since the servo used to open the

claw had been modified. The servo was modified so that it would have enough range of motion to fully open or close the claw.

Two servos were used for the arm. One to close the claw and one to move the arm up and down.

Chassis

The chassis consisted of the lower baseplate, upper baseplate, the motor mounts, and the side panels.



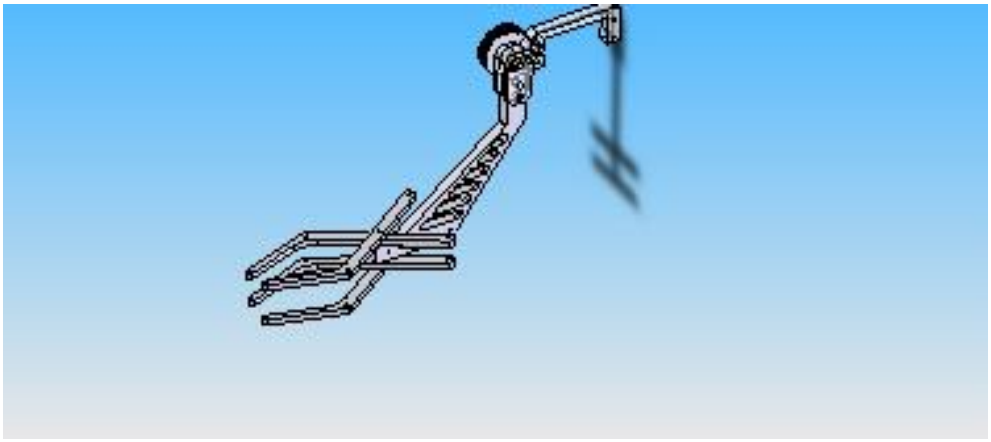
The chassis was designed almost completely around this piece of scrap plastic we found (which was some student's attempt at a perfect octagon). We ended up deciding not to use the piece but kept the design. Instead of using an imperfect octagon, however, we decided to make it a perfect octagon. The lower baseplate was originally designed to hold both the computer and the battery. It was later found that it was in the best interests of space for the computer to go on the top. The orcboard was then mounted on the lower baseplate. The sidepanels were also each slightly different, with an arrangement of holes so that the orcboard and orcpad could be mounted in an accessible and safe fashion. In the final design they were not mounted in the same orientation because they did not allow for enough clearance between the wheels and the chassis. The upper baseplate had horizontal holes in it so that the ball delivery bin could be easily

mounted. Since the ball delivery bin was laserjetted, it could only be a two dimensional shape which proved to make mounting more difficult. The motor mounts were precision ground to an outrageous tolerance because it we originally planned to mount potentiometers and to have metal gears connect the shafts of the motor and potentiometers. (Metal gears are more durable, but require more precision)

Throughout the evolution of the chassis, the original design proved to be less and less space efficient. If there were time, the chassis would be completely changed in order to improve the ball delivery capabilities.

Gripper Arm

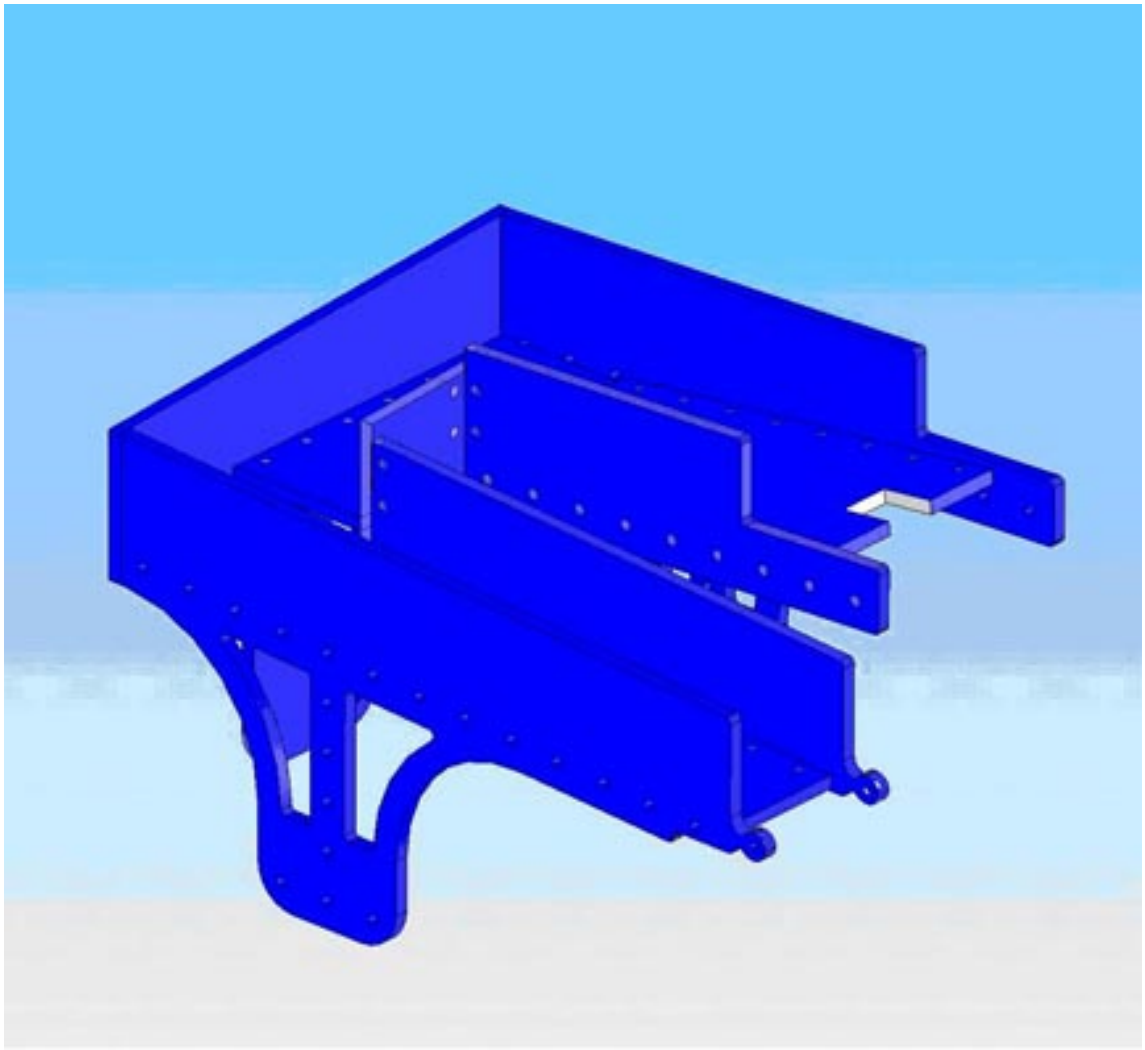
The Gripper Arm was designed to minimize the amount of precision and energy needed to pick up a ball.



The end effector of gripper arm has the general shape of a plow (when open) with two stable positions from an approaching ball. It has span of 8 inches, so if a ball was even remotely centered in front of the robot, when the robot drives forward, it would be able to center the ball and initiate the closing mechanism. As the claw closed the geometry of the device insured that any ball with its workspace would be corralled with perfect reliability. The sole link of the gripper arm was designed and constructed out of acrylic to not reduce both weight and a truss-like pattern was implemented to reduce stress concentrations. Actuation is achieved through cable driven pulleys to minimize the arms weight resulting in less energy usage. Due to the grippers actuation geometry compliance was employed to maintain cable tension. Light springs where used on the cable mechanism for opening opposed by heavy springs on the opening cable chain. After grabbing a ball the arm will rotate 180 degrees to deposit the ball in the ball delivery bin.

Ball Delivery Bin

The ball delivery bin was designed as a completely passive device.



The ball delivery bin was designed to hold 14 balls. It had three different sloped sections so that a ball could be both dropped in the front and later be ready to be delivered again from the front. The bin had a front flap and a spring system so that if the robot were to drive into a goal, it would be able to score without the need of an extra servo or drive motor.

Software Design

When the contest began, we had no idea what our robot would look like, or what capabilities. Therefore, one of the requirements of our software design was that it should be very general and extensible. This may have been a bad decision, as getting the framework down for such architecture took time away from the development of algorithms to control our robot.

The original code design was broken up into four modules: a Navigator, which would encapsulate all robot movement commands and keep track of odometry data, a Scanner, which would create a constant stream of information about visible objects (using the camera), a Mapper, which would incorporate both scanner and navigator information in order to estimate the location of the robot and other features of the environment (like the barcodes), and a Controller, which would use the other modules to make our robot act intelligently.

In the end the code design stayed relatively the same, except that some of the modules were lacking in content. For example, we never got to write any Mapper code, though the mechanism for incorporating those capabilities was there.

The Navigator

The Navigator contains the robot's motors, IR sensors, and a gyro. The original design called for two potentiometers affixed to the wheels which would allow us to tell the exact angle of the wheels, but all our potentiometers broke, leaving us with nothing but a gyro and the IR sensors for non-camera navigation. Within the Navigator, an Odometer runs a task which was to update the position using the sensor data every 20ms, passing the information to the Mapper, where it would be corrected using available camera data. This task would create OdometryData objects which would be stored in a sorted tree by time stamp, allowing fast access of data for specific blocks of time. While all of this code was written, it had to be abandoned when the potentiometers broke. Because our Navigator did not completely depend on this OdometryData we were able to hook up the gyro at the last minute so that we could at least turn a given number of degrees.

The Scanner

The Scanner contains the camera and allows access to the latest CameraData objects. Each of these is a series of arrays containing the information about the balls, goals, and barcodes found in the latest captured picture, along with the odometry data for the time the picture was taken.

The vision algorithms are based largely on the fact that our interesting objects are connected blobs of color in some range. We implemented HSVColor and HSVColorRange classes to encapsulate a single HSV color and a range of colors between two such colors. Another class, HSVMultipleColorRange, was added to create ranges which contain many disconnected ranges. This was used for the barcode, which includes both black and green. A general ImageScanner class is initialized with HSVColorRanges which it is to scan for. It can then be used to scan parts of a picture for areas of interest containing these colors, using a fill algorithm to mark related pixels once a single interesting pixel is found, and never scanning pixels which have already been marked. Each interesting area is created by initializing an AreaOfInterest object. This object performs the fill algorithm and collects a lot of data about the area, such as the leftmost, rightmost, lowest and highest point, and information necessary to compute the centroid (this was extremely useful, as sometimes the area would leak a bit under the barcodes, shifting the midpoint away from the actual barcode). Depending on what we are scanning for, these areas of interest can be analyzed to classify them as goals, balls, barcodes, etc.

Our implementation includes a BallScanner, GoalScanner, and BarcodeScanner, which each use this method to detect the GameObjects. Because our camera was mounted 5.3 inches above the ground, it was only necessary to scan only the bottom half of the picture when scanning for balls, and amazingly only the middle few rows of pixels to detect goals and barcodes (the fill algorithm takes care of everything after you detect the first pixel in the range). This allowed us to run our camera at a good 10

FPS using full 160x120 images, including some processing to produce the output debug images. GameObjects use details about the dimensions of the camera to estimate their angle and distance from the robot. The Camera Data was updated every 100ms using a new picture and stored and accessed using thread-safe methods.

The Mapper

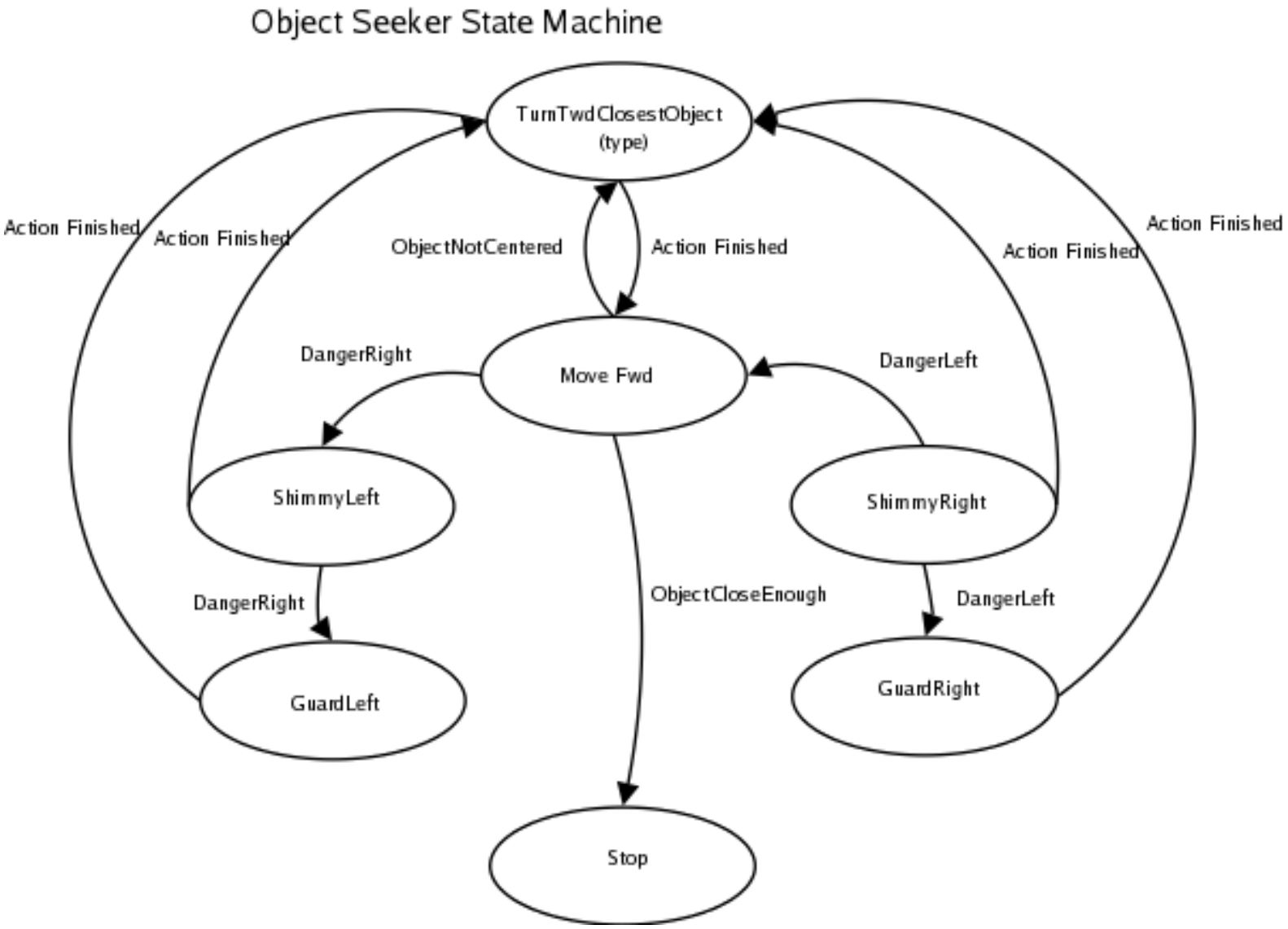
The mapper was supposed to create a map. Unfortunately our lack of odometry data and time left this totally empty. Ideally, this would have used a Kalman filter to keep estimates of the robot and barcode positions, updating these every time odometry or camera data was refreshed, thus giving a good approximation of their positions.

The Controller

The Controller is what Popular Science might call the “brain” of our robot. The final implementation consisted of a finite state machine which continuously used the latest camera and sensor data to decide when to transition between states.

The actual StateMachine class was coded to be entirely general-purpose. Action objects comprise the states, and ExitCondition objects define the transition conditions between states. The underlying structure is a DirectedGraph with Actions as node labels and ExitConditions as edge labels. When run, actions can be queried for their state, with values such as PREPARING, PERFORMING, FINISHED, ERROR, etc. ExitConditions specify their requirement for activation (which can use the state of the any action being performed) as well as the frequency with which they are checked, the default being 100ms.

When a StateMachine runs, it creates a java.awt.Timer object which checks the exit conditions from the current action at their frequency, then runs the Action in another thread. When an ExitCondition is activated, it interrupts the Action thread, kills the timer, and causes the state machine to transition to the next state and repeat the process. Because Actions are very general, it was possible to create an action which itself ran a given state machine, allowing us to have many levels of state machine with their exit conditions at the various levels checked simultaneously. Because our objects were also fairly general, we were able to write a single state machine which could be adapted to seek ball, goals, barcodes and possibly other objects with a simple change in input at initialization. This allowed for improvements in goal-seeking, for example, to also improve ball-seeking actions, improving our efficiency when developing the state machines. The same collision avoidance states never had to be copied into different state machines.



Overall Performance

Our contest round began (or rather didn't begin) with our orc pad freezing, a reoccurring problem that could only be fixed with a system reboot. But after coming back online and jumping back into the contest cue, Johnny initiated fine and began to transition beautifully through the FSM. The robust wandering code performed as expected, as Johnny bounced around the course. Unfortunately, most of our wander testing was done while tethered to a power supply, so the overall speed and -- therefore -- the area that our robot covered was less than ideal. Johnny failed to collect the first ball he encoutered due to the fact that the arm was not brought down in time, simply an error in calibration. The second ball Johnny encountered was centered with a relatively large error, seriously testing the limits of the gripper's mechanical robustness. The gripper did manage to grasp the ball, although the far reach of the right claw was only barely past the center of mass of the ball. During the round, we collected three balls and successfully deposited them into our bin. At this point, Johnny began to look for a goal. The goal scoring state found a goal within a short time and attempted to center and shimmy to the proper orientation. This unfortunately was the weakest part of our implementation, since it was only tested a few times on three goal configurations. As we had feared at the start of the contest, the alignment was slightly off and our passive trigger hit the field goal upright. Overall, we were extremely pleased with Johnny's performance and

everything performed (or didn't perform) as it was expected to, which is a miracle in engineering implementation.

Conclusions/Suggestions for future teams

Suggestions

- organize formal team design reviews, the key to your robots implementation is in the group dynamics
- Be modular and build something that works before going into super complicated mapping and odometry
- Make sure you are making progress towards your goal each day, don't waste time on stuff you cant use
- Do not underestimate the difficulty of things that may seem simple on the surface.
- Buy encoders you cheap bastard
- Make sure to test everything as you go along… make sure each function you write works before you move on.
- Plan out your software structure before you start coding randomly
- When designing your mechanical structure make decisions based on how long it will take to make because you need a working machine early that you can test.
- Make sure those making software talk to those doing mechanical things because something that may make sense mechanical may make the software 10x harder to write.

Conclusion

Johnny 5 made a good showing and impressed a number of people as they walked by but ultimately could have been much more than it was. The mechanical portion of our robot took over as things began to take longer and longer to come in and built. The arm ended up being a beautiful thing but it did not take the single week we expected to have it in working condition. Our software, although functional, could have been better if we had gone all out at the beginning of the month. We were unsure as to what kind of odometry we would be able to utilize and spent much time planning a mapping round that never came about. Potentiometers that would have been to used to measure our distance and be used to keep track of our location ended up failing as the pots broke after only a few days of use. In the end we scraped together an impressive machine using only a camera, gyro and IR sensors to navigate through space. I believe we did extremely well under the pressures, setbacks and inevitable frustrations that came up along the way. Working up to the last minute, we worked as a team utilizing the strengths of each individual to produce a robot that did perform well. I think we will forgive Johnny 5 for becoming stuck on the left goal post on its scoring run and call him a success. We all got a great deal out of this experience and for some of us it is the beginning of many more endeavors into robotics.

- Team ONE (Nu Delta)