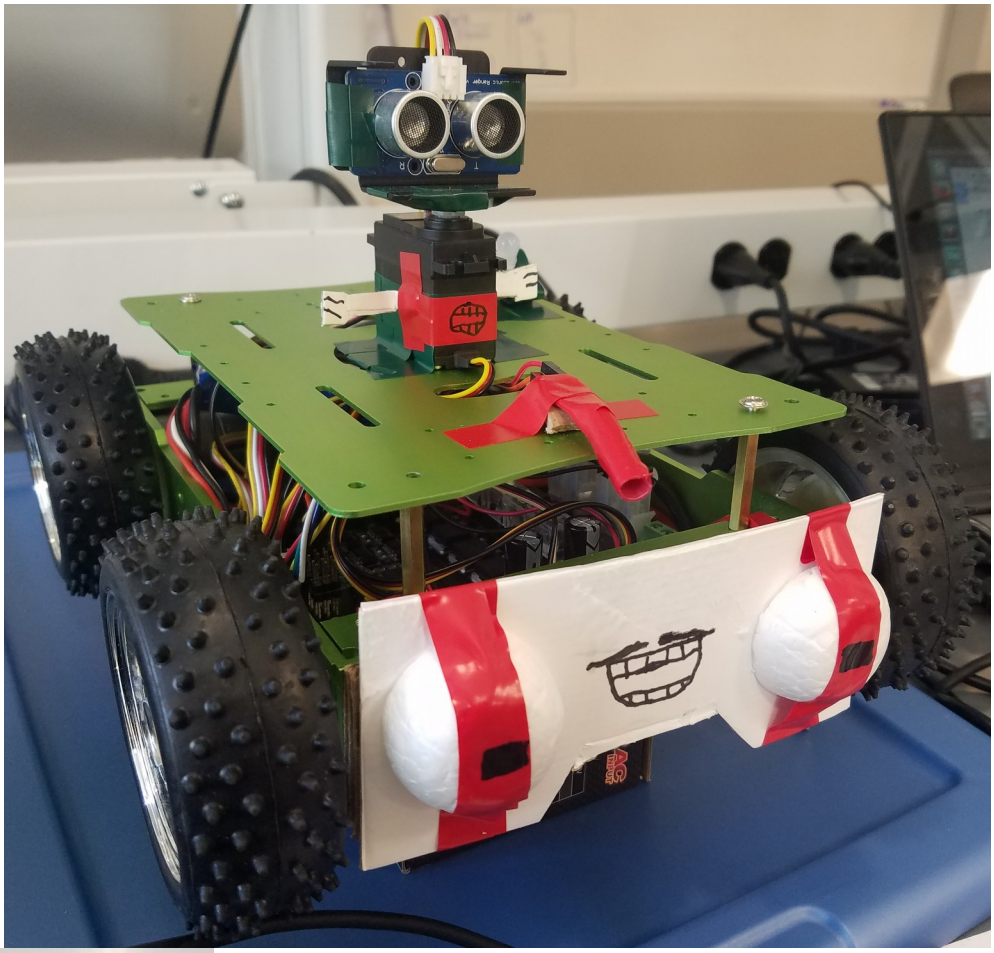
Nicolas Fredrickson

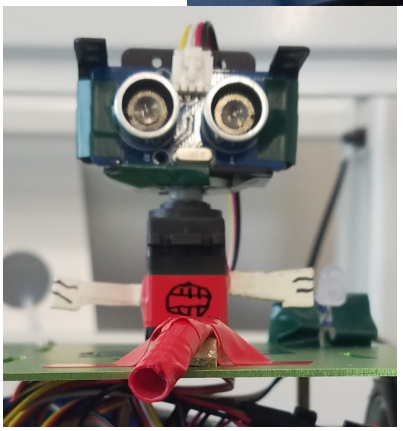
May 16th, 2017

Robotics Final Report

Our goal with this robot was to have it autonomously navigate to a certain goal point, then navigate back to its starting position, and then climb an incline using a light sensor. The robot was navigating through a 5x5 grid, with obstacles placed to occupy entire sectors. The incline was placed on the edge of one sector.

The starting position, our goal position, and our incline position were all deterministic – that is, they were determined ahead of time. The positions of the obstacles, however, were not. Thus, my robot – named **Slick** – had to navigate the grid while determining where our obstacles were.





Above: A picture of Slick, showing off the HS-422 Servo, its mounted Ultrasonic Ranger, the light-sensing Diode, the decorative bumper (which conceals the button) and the body of the robot. Also in view (albeit hard to see) is the RGB LED, just to the right of the servo, above the Master Blaster arm.

Left: A frontal view of the Servo, Ultrasonic Ranger, Diode, and RGB LED (just right of the Master Blaster arm).

The Robot

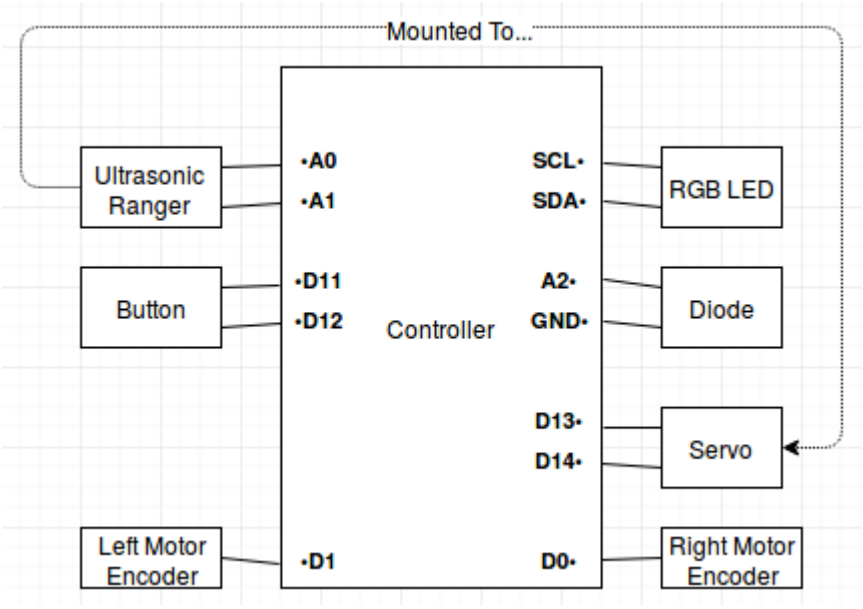
Our robot was composed of multiple hardware components, which were broadly grouped together in the code as different subsystems. When discussing the robot, I tend to break up the subsystems as the Motors, the Button, the Servo, the Ultrasonic Ranger, the Diode, and the RGB LED. The motors were simply the motors that came with the robot; they were attached to interrupt pins in order to count the number of encoder ticks, and don't merit much discussion beyond that.

The Button was placed on the front of the robot in order to prevent damage to the motors. If the robot was told to go forward and were to collide with an object, it would still attempt to spin the motors and ultimately wear down their machinery. However, since the button didn't extend past the wheels when mounted on the front, I created a bumper. The bumper sat over the button and was held loosely in place with electrical tape, to allow for the bumper to be compressed. In the end, the bumper was used not only to stop the robot after a collision, but to use that occurrence as a method of detection – transforming Slick into a bumpbot.

The Servo and the Ultrasonic Ranger, while separate subsystems, were very closely related. The Ultrasonic Ranger's purpose was to detect obstacles from a distance. In order to get the best coverage, the Ranger would need to face different directions; however, turning the robot would result in accumulating error. The Ranger was mounted onto the Servo so that it could face to the left, straight ahead, and to the right without the robot having to change it's orientation. Unfortunately, the Servo's range was constricted to roughly 180 degrees, which means we couldn't check behind the robot – not a problem though, as the only time we have to back up is after going forward. Also, the Ranger was meant to check for obstacles ahead of the robot while it was moving, and to stop the robot if a obstacle was detected. This proved to be wildly inaccurate and led to strange behavior, so it was removed.

The Diode was used to detect light during the Incline section of the robot's behavior. In order to give the diode a focus in one specific direction, it was placed in a custom sheath, composed of a straw and several layers of electrical tape. The Diode was also pointed downward, in order to track the guiding lights on the incline.

The RGB LED was placed on top of the robot purely for feedback. It was often difficult to gauge the robot's state when testing, so I needed someway for the robot to visually communicate its state. It had rather inconsistent behavior, owing to it typically being more of a debug tool than a necessary component of the robot.



Behaviors

The machine's behaviors were collected as a set of states – Wander, Wander Out, Go Home, Incline, and Done. It also kept track of it's movement state – Forward, Reverse, Stopped, Turn Left, Turn Right. In practice, only the first three movement states were relevant. The machine's default state was Stopped, in the Wander state.

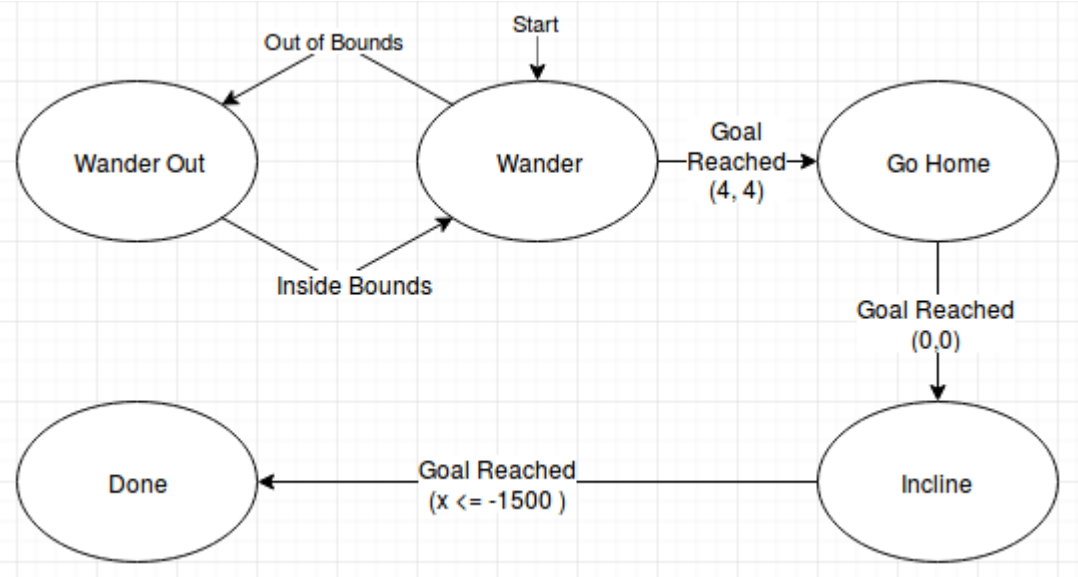
Whenever the robot was stopped, the Wander state would use the Ranger to investigate left, forward, and right. It would then try and plot a path to the goal, using a (mostly working) version of the A* algorithm. The algorithm selected the cheapest path, scoring each path using a heuristic – paths that introduced more error (i.e. turning) cost more. While we did find a complete path to the goal, we were actually only interested in the first action on the path, and the first subsequent action where we would stop the robot (either turning or straight-up stopping). We would take the first action (maybe going forward, going reverse, going left, or going right) and would consider that 'subsequent action' as our sub-target (for when we want to stop).

If we were going forward or reverse, and we went outside the boundary, we would enter the Wander Out state. We would stop, the immediately switch to the opposite direction. When we had re-entered the boundary, the vehicle would stop and re-enter Wander mode.

When going forward, if we collided with an obstacle, we would: stop, register the appropriate sector as blocked, set our sub-target to the current sector, and then reverse. If we were going forward or reverse, and we reached our sub-target, we would stop. Determining where to stop in a particular sector changed with our direction (forward or backward) and our heading.

Now, once we had reached our target we transition to the Go Home state. This state is literally just the Wander state with a several changes. First, the goal has been changed from (4, 4) to (0, 0). Second, we no longer check for hazards with our Ranger, or our Button, our even checking if we've gone over the boundry. Third, the robot now moves at a faster speed going forward or reversing. Finally, the path algorithm now gives a higher cost to sectors we haven't passed through before.

After we reach Go Home's goal, the robot turns to align itself with the ramp and pauses, to give me a chance to properly place it on the ramp. The robot the progresses up the ramp by alternating between going slightly left and going slightly right. Whatever state it is in, it flips when it detects that it has gone over the LED strip found on the incline. This keeps it roughly centered until it reaches its goal – having an x value of less than (or equal to) -1500, a measured value that typically stopped the robot as soon as it left the incline.



The Results

Wander Time	Go Home Time	Incline	Style, Verve
1:07 1 Mark	1:19 1 Mark	1:32 No marks	5 + 1

During the Wander phase, my robot ended up perceiving a blockage where there was none. I thought that would be the end of things, but to my surprise, my robot actually navigated out of that perceived dead end and proceeded to reach the goal. I found it quite impressive – even if it netted me the absolute longest time, even without the marks.

The error – which was an error I had seen before – seemed to happen when the Ranger was investigating a sector between two boxes. Typically when the robot ended up in these situations, it would end with an infinite loop of moving forward and backwards – a victim of bad path finding. Seemed that didn't happen this time – though why, I'm not too sure.

Still, after the Wander state, it only took 25 extra seconds to finish. My run was one of those with the least marked, and I had the most style and verve. Honestly, I feel like that's reflective of my overall design philosophy concerning the robot. I was trying to aim for consistency with the turning and navigation. It was exactly what I expected, but maybe just a bit slower than I wanted.