Autonomous Crown-Fetching Robot

Patrick Ayers & Robert Hoffman

Introduction:

For the final project of Mechatronics we were tasked to create an autonomous machine, capable of navigating a course specified by the professor in a time limit of 2 minutes. This was to be done in a 30 day period using both purchased materials and those supplied to us by the faculty. It is a demonstration of the various skills we had learned in the quarter leading up, and a challenge in implementing our own designs.

Description:

The challenge this year was called “Game of Drones:” the field was modeled into four castles in the styling of a popular tv series, and our robot had to navigate from one to another. Of the four castles, the robot randomly starts in one, and another has a crown that the robot must retrieve and return to its own castle. Each castle has a throne in it where either the crown rests or can be placed. The castle that the robot starts in has an active track wire along one of the walls, and the castle holding the crown has an IR beacon on the throne beneath the crown. Lastly, there is a randomly placed obstacle represented by an 11” cube in the field that the robot must evade. For more information, visit this page: https://classes.soe.ucsc.edu/cmpe118/Fall14/Project/CE118_2014_ProjectOverview.pdf

Methods:

Bill of Materials:

-MDF $5
-Assorted Hardware (bolts, nuts) $20
-Motors provided by Max Lichtenstein ($40 approx.)
-UNO32 and H-Bridge provided by Gabriel Elkaim
-Servomotor recovered from electronic scrap ($15 value)
-Limit Switches ($0.89 x 6)
-Tape Sensors
-Assorted Electronics ~$5
-5V Switching Regulator $5

Track Wire Detection:

A wire which radiates a 25kHz signal is placed on the back wall of the home castle. This is useful for orienting to the exit of your starting castle, so a detector was built so that the robot could detect proximity to this wire. The first stage of this circuit is an LC tank circuit. The inductor and capacitor are chosen to provide a center frequency near 25kHz. It was necessary to add a trimpot to the amplifier stage of this circuit so that the range of our sensor could be adjusted to fit its placement in our robot. It is placed in the front and triggers a 0 signal to the PIC32 if the wire is within 2 inches of the sensor coil.

IR Beacon Detection:

The ruling castle which contains the crown has a throne which flashes IR LEDs at 2kHz with 50% duty cycle. This modulation helps to distinguish the beacon from external IR noise from the environment. The detector developed uses a phototransistor fed through a tran-resistive amplifier stage with a 2.5v DC bias. This initial signal is then AC coupled and amplified before being passed through a bandpass filter designed to center on the 2kHz frequency of the IR beacon. This filtered signal is then AC coupled and amplified before going through a peak detector and inverting comparator with hysteresis. This comparator will output a 0 if the beacon is within 8 feet of the phototransistor, and a 1 if IR is undetected. The signal is 5v, so it must also pass through a voltage divider at 3.3v before it is used by the PIC32.

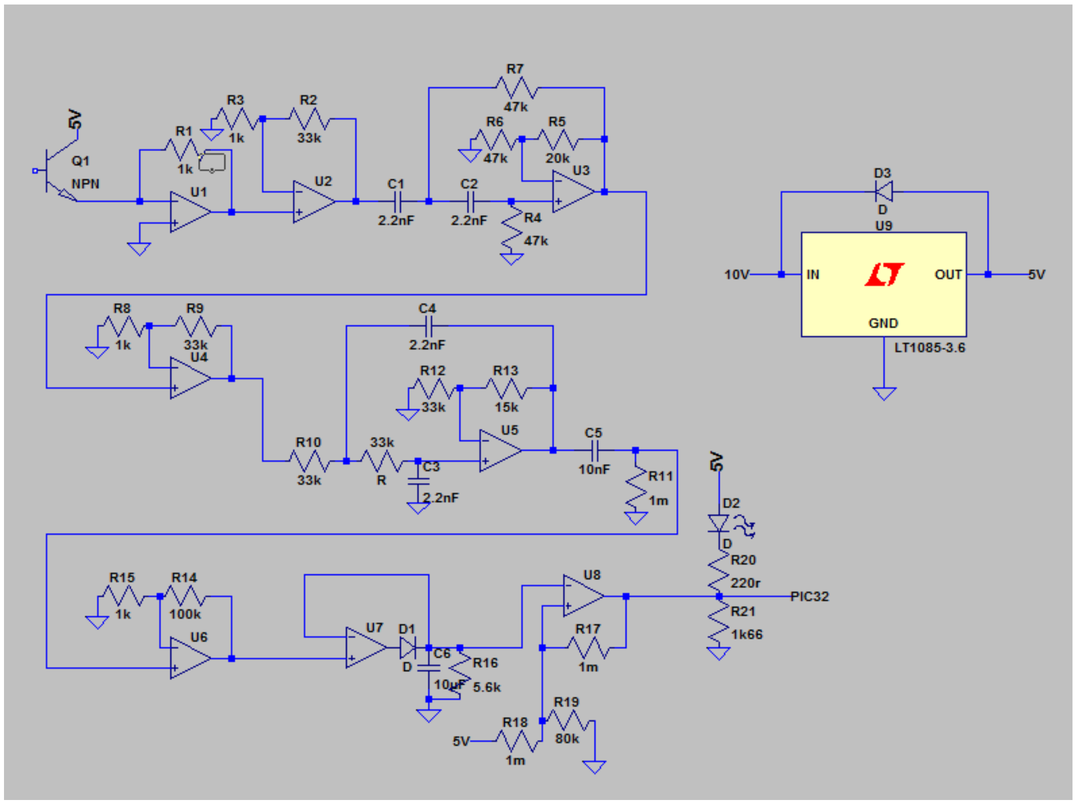

Figure 1: IR Beacon Detector Schematic

Tape Detection:

In several areas of the board space, black electrical tape was used to indicate various objects and points of interest. Our design was primarily interested in the fact that each of the moats had a tape border. The tape sensors comprised of an IR LED and phototransistor. The analog signal produced by the phototransistor had different values depending on various factors, but had the most change when the surface that the robot was over changed from white MDF to black tape. All IR LEDs were powered at all times by 3.3v through 100 ohm resistors. This made each one fairly bright (about 30mA each) in an attempt to overwhelm external IR noise. Connecting each phototransistor from 3.3v to ground using a 10k ohm resistor and reading the value into the ADC pin of a PIC32 gave reading from around 900 on white MDF to 100 on black tape. Each phototransistors reading was slightly different based on its unique placement on the robot so separate low and high hysteresis thresholds were measured and chosen for each of the 5 sensors used. As the arena got more scratched up and dirty from usage, the robot would sense small scuffs in the floor as tape. To lower the sensitivity, we implemented software which takes many ADC samples before taking the average of these readings. Averaging out a few hundred readings effectively lowered the sensitivity to small imperfections in the floor

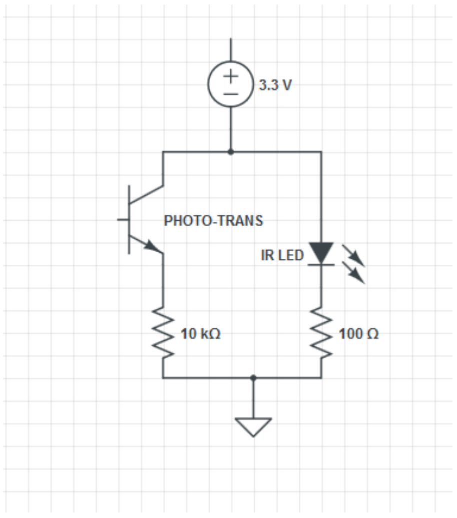

Figure 2: Tape Sensor Circuit

Navigation:

A few different ideas were proposed for the task of determining the robot’s position and the best course of action in programming. Initially we sought to look over the walls of each castle to determine which had the crown as quickly as possible. For navigation this meant choosing one hallway to go back and forth the full distance on. This was scrapped in favor of driving from the entrance of one castle to the next, checking the beacon detector continuously. This was chosen in part because the robot would only have to move a short distance before realigning and could keep track of itself more easily that way.

The first order of business was to find a way for the robot to accurately track its position. We determined that the easiest way to do that given our hardware was to push against walls with a flat edge or bumper. The first state in our design had the robot driving in a square slightly larger than the castles, bumping off of walls, until it located the trackwire. Because 90 degree turns are achieved using timers, slight variation in the surface cause an accumulation of error in the robot’s steering. Squaring up against the front bumper periodically removes this error and gets the robot back on course.

The most difficult task was to get the robot to drive to the center of the arena even after multiple collisions along the way. There are no walls to determine if the distance has been travelled, and a simple timer does not work as driving time changes if bumps are encountered. To get this part of the navigation to work, when the robot first heads down the hall to the center, a timer is started for the estimated time it would take if no collisions occur. If the robot hits something along the way, this timer is stopped and evasive maneuvers are done. When the robot is ready to continue heading to the center, the timer is resumed but with some added time to compensate for the collision. After some tweaking of the timer and compensation values, the robot could successfully get to the center despite many wall collisions on the way.

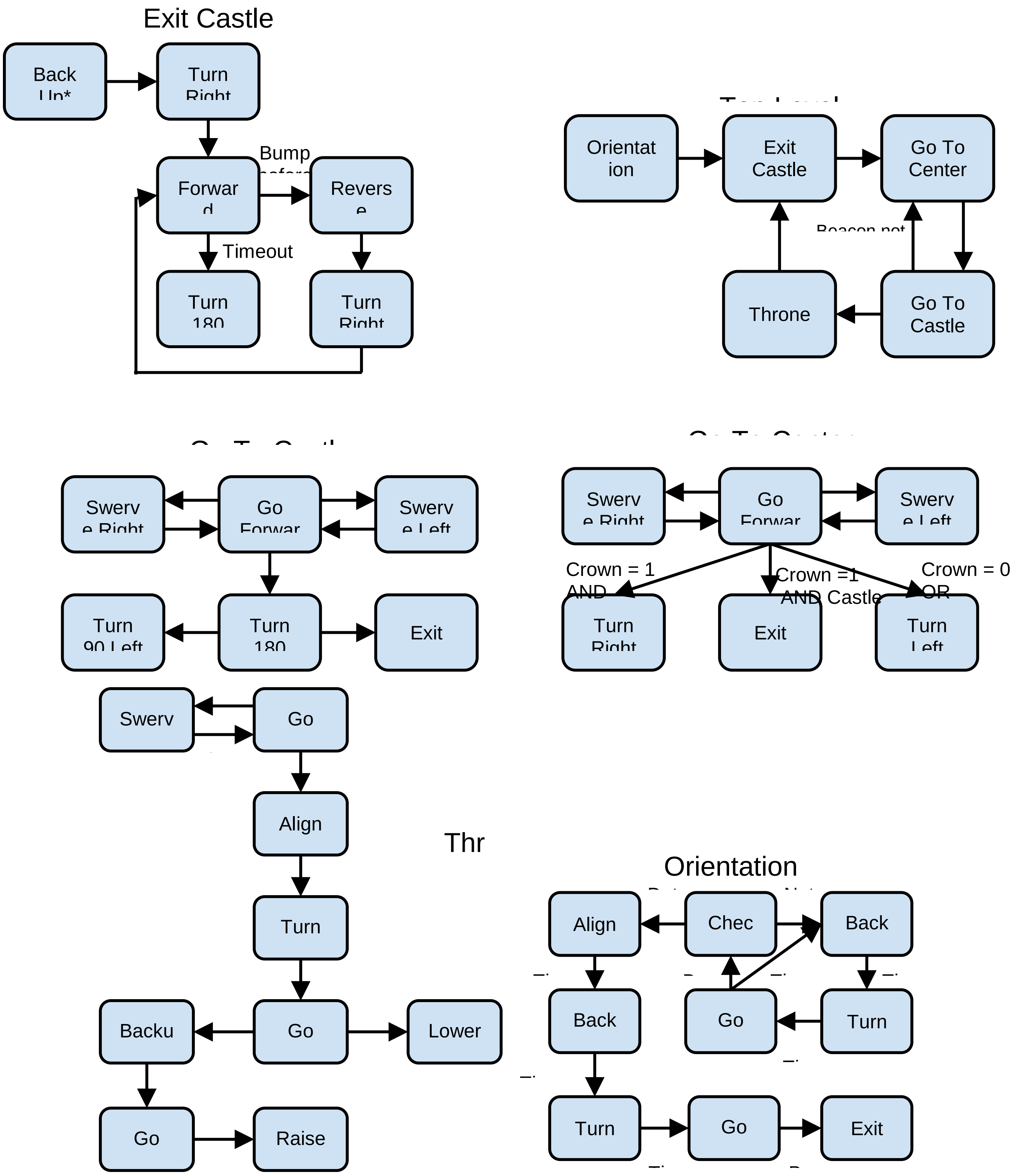

Figure 3: State Diagrams

Mechanical Design:

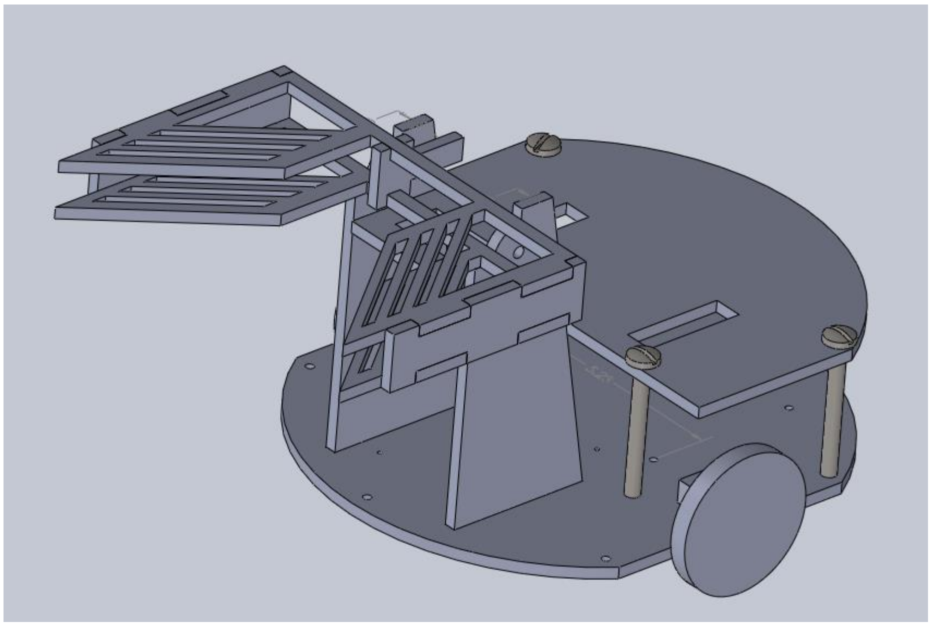

Figure 4: Mechanical Design (excluding bumpers)

We had two design philosophies that we kept consistent throughout our development process: avoid hard corners whenever possible, and keep the robot as compact as possible. All of our base designs used an 8.5” diameter circle as the basis for sizing, and trimmed it so that we had a flat edge on two opposite sides for wheel placement. This aspect of the design never changed, but we went through several iterations altering what mounting holes the base had in it. Rectangular grooves were cut into the base so that the tape sensors could be recessed into these holes. This was done because small wheels and mounting the motors on top of the base gave our robot low ground clearance. When the reflective sensors are too close to the ground, they do not give stable ADC readings.

To provide extra room for electronic components, we made a platform that resembled half of our base, which was mounted 3.5” above it on four 0.25” bolts. Also, instead of mounting holes for motors and sensors, this platform had two slats for wire routing and holes for the bolts to hold itself up with. For wiring convenience, the platform was directly above where our UNO32 board was placed, and the platform itself held the H-bridge circuit and the battery.

The other half of the robot was devoted to sensors mostly. We used two vertical pieces of MDF that held several platforms between them to hold our track wire sensor, beacon detector, and servomotor with the attached lifting mechanism.The IR detector was tilted in a way so that it would point directly at at the beacon.

The arm mechanism for lifting the crown was a simple design, each side was slanted towards the center so that if it hit the crown at an angle the robot would slide into position. The actuation of this was done by a servo motor, which rotated the arm between a lowered and upright position for picking up and placing the crown.

The front side of the robot had three bumpers for navigation. One center facing bumper and two at about 45 degree angles to the side, with no space between them. The side bumpers also flared about an inch off the side of the bot so that they would cover the wheels. Each of these were held in place by limit switches glued to the based. Each switch was also reinforced with a slotted piece of MDF, and the bumpers were constructed with pieces of MDF as well.

Programming:

Our programming process worked in stages: first we wrote a common file of functions useful to our robot specifically, and then worked on each low level state machine individually. In addition, we also constructed a test harness that would serially print out the various values of each sensor given user input, allowing us to debug sensors painlessly for the most part. Having the test harness also helped us make fine tuned calibrations for the robot's movement around the field, which was important given the amount of space that it must travel outside the castle without relying on any sensors. It also helped us assign reasonable values to all the sensor thresholds. In fact, due to the finicky nature of our tape sensors, each individual sensor had its own thresholds for triggering events.

We implemented three event checking functions: beacon detection, tape detection, and bumpers. Since the bumpers already sent digital signals their event parameter was encoded as a bitfield with each bit representing which bumper had been triggered. Additionally, we incorporated bumper debouncing in our state machine with each state that's triggered by a bump event not having an exit condition based on a bump event. With the tape sensors the parameter became more complicated as it became necessary to determine whether or not each sensor had just gone over tape or left a taped area. Again we used a bit field, but this time with each even number bit representing whether each sensor was currently over tape and the adjacent odd bits representing whether or not that sensors value had just changed. The beacon event was stored as a binary value, passing whether or not it was currently receiving IR light.

We had encountered mechanical difficulties with our motors being unequal and alleviated this by having each motors duty cycle be modified proportionally so would drive at near equal speeds. As the project progressed so did mechanical degradation, so having the value be a proportion allowed us to approximate and adjust on the fly. In addition to being proportional to each other, the motors were driven proportional to the amount of voltage still left in the battery. This eliminated concern of our calibrations failing since now the wheels would spin at the same speed regardless of the amount of power left.

The most important part of keeping our program simple was reusing states: rather than have different states for returning or exploring, our robot would loop back to an early point in its navigation when it retrieved the crown. There are only three places in our code where the robot checks to see if it is returning: when it reaches the center of the field and needs to decide where to turn, when it is in front of a castle and deciding to go in or not, and when it is deciding whether to pick up or put down the crown. The data needed to make these decisions was stored in two static variables: which castle the robot finds the crown in and whether or not the robot was holding the crown.

Conclusion:

If there was one thing to take away from this project, it would be the frailty of mechanical implements and the need to have a contingency plan. Our motors failed late into the project and replacing them was a serious setback in finishing the robot. Regardless of the quality of the motors purchased, we should have invested into having extras or an alternate design to fall back on. It was possible for us to recover from that issue but it is not something I would like to assume I have the ability to do at any point in the future.

As for other parts of the robot, our design was fluid and went through several changes in every aspect as we progressed. Initially we each wanted to focus on our own specific areas but it became that both of us had to involve ourselves in every aspect of construction and design. This turned out to be a good thing as many ideas were pondered and vetoed, saving time, resources, and frustrations on our behalf.

For the actual process and scheduling, we kept a fairly organized system in that anything built had to be tested within the week it was started. This let us move through prototypes quickly (which in turn meant we had a large number of prototypes) and refine our design decisions quickly. It also meant that we became quick at assembling and testing new designs, which would help us in the final hour when we had a massive mechanical failure.

Running the test harness many times throughout the production of our robot proved to be vital to solving underlying issues. We struggled with faulty, seemingly random tape sensor errors until the test harness revealed to us that it was a faulty ADC pin on the UNO32 which had to be disconnected. Without running specific tests, the hardware error would have never been located.

Link to our files: http://tinyurl.com/RHofPayersLab