Team Two Paper

Overall Strategy

The overall strategy for Jimboo the Transbot was to collect balls and place them into the field goals. We intended to utilize a unique drive system to implement a mechanical solution to what we thought would be a challenging software problem - robot alignment and obstacle avoidance. The drive system that we used we labeled a transdrive system, which consisted of three transwheels (wheels with rollers that allow for sideways translation) in such a configuration that allowed us to move forwards, backwards, side to side, and even diagonally. With this, we eliminated the need for complicated algorithms and movements in order to adjust our robot to the side a short distance, which would be useful for collecting balls and centering our robot in front of a field goal. This improved maneuverability also allowed us to navigate through tight spaces.

Mechanisms and Sensors

These wheels were rather difficult to mount since they have a 1" bore and there was no preconfigured way to attach them to the motors (as was the case for the standard wheels). So, we resolved this problem by creating a keyway 1" shaft out of aluminum, then press fitting this to a ¼" steel shaft. We then bought used bike gears, which we had to press fit to a 1.3" aluminum disk which we press fit to the same ¼" shaft. As for attaching the gear to the motor, we did something similar, except instead of press fitting, we used a setscrew. The major issues that we encountered with this drive system was wheel contact, chain alignment, and an underpowered middle wheel. Since the contact point of each of the three wheels lie in a line, there was an issue of making sure each wheel touched the ground. We resolved this by making the middle wheel slightly lower than the outer two and placing the weight of the rest of the robot on the outer edges, creating a three point binding problem.

As we kept adding components to the robot, it became so heavy that the middle wheel didn't have enough of a driving force to move sideways anymore. We tried to use a higher torque motor but force it provided was disappointing. Instead we redistributed the weight on the robot by relocating the battery and our side-to-side motion was once again operable.

Our ball collection mechanism consisted of a front roller, a conveyor belt-like lift, and a tray which would store and release the balls. The underlying principle of our design was to allow for the largest margin of error, thus we built a wide roller and a wide tray which hung two inches in front of our robot. We settled on this design because we considered it to be robust, simple, and efficient. Unlike other ball collection mechanisms we considered (paddles and claws, for instance), the roller could be operated constantly with a high probability of functionality given almost every reasonable situation.

The lift system consisted of a aluminum ramp covered in gripping tape, and was actuated by a belt

constructed out of carpet liner driven by a driven modified servo. Although it required a bit of fine adjustment, the ramp system worked reliably (albeit slowly). The main issue that we encountered with the lift mechanism dealt with our height limit. Since the rules changed mid-contest from 14"x14"x20" to the size of our bin, we encountered issues involving the transition of balls from the lift to the tray (which was resolved by redesigning our tray deposit mechanism). Another issue that we encountered was drift in the belt - after about 45sec, the belt would begin to bunch up at one end of the shaft. Although we ran out of the time necessary to fix this problem, our solution was to add a belt tensioner that would correct for the inherent imperfections in the belt and shafts. As for the issue of adjusting the ramp and belt, we constructed adjustable mounts for each shaft, which allowed is to adjust the system to create the desired distance between the belt and ramp along with the roller and the ground.

Our tray mechanism consisted of several slightly angled ramps that allowed for the balls to roll from the back of the robot (where they were deposited by the lift) to the front, where they sat until unloaded. Originally we constructed this by gluing several pieces of wood at the desired angles, but in the end we felt it would function better (and look nicer) if it was constructed out of one sheet of aluminum. Our original plan for deposit was to construct a servo (or solenoid) powered door, which could release the balls at the appropriate time. However, due to the design issues we encountered with the lift mechanism, we replaced the door with a tilting mechanism powered by a linear actuator. When tilted in one direction, the balls would rest towards the back of the tray, and when tilted the other direction, the balls would roll forwards and off the front of the ramp. The linear actuator worked quite well to deposit balls the day before impounding, but promptly blew up amidst the triumphant sound of unmeshed gears.

In terms of sensors we had very few. We opted to use the optical encoders on our middle and right wheels to aid in turning and keeping track of how far our robot traveled. We also had two mid range distance sensors in the front of our robot to perform wall avoidance. We did have a gyro connected to our robot but it served very little purpose. It was to be used in an more advanced mapper but we ran out of time before we could get that feature fully implemented. In addition to these physical sensors we also had current sensing for determining whether or not our robot was stuck on an obstacle.

Software Sub-System

The software side of our robot was fairly simple consisting of three main parts: Vision, Mapper, and State Machine. The Vision sub-system handled all of the image processing work and was in and of itself, fairly simple. Originally, it was only able to recognize balls and bar codes. The recognition of multiple balls was one of the first things completed. Several methods of gaining this recognition were toyed with but eventually a simple, albeit nonoptimal solution was adopted. To recognize multiple balls the following was done. First, the bounding box of all of the red pixels in the image was determined. Next, the area was split into two either vertically or horizontally based on the aspect ratio of the bounding box. Then, the bounding box of the two newly split boxes are computed. If a certain number of red pixels are found and the aspect ration is close to one, a ball is said to be found and no further recursion is performed. Otherwise, one of two other things happen. In the event that not enough red pixels are found, recursion on the area is terminated and that portion of the image is ignored. If enough red pixels are

found and the aspect ratio is below a certain threshold, the bounding box is once again split in two and the whole process is repeated again.

Once a ball had been found its angle and distance from the camera were computed as well. The angle to the ball was computed by using a bit of geometry based on the x coordinate of the center of the ball. We were able to do this once we estimated the focal distance of our camera. Distance to the ball was estimated using the size of the ball's bounding box. We collected a bunch of readings from different distances and fit them to a curve so we could come up with a fairly reliable estimate of the actual distance to the ball.

Mouse holes were handled in the exact same manner as balls, except the bounding box of the mouse hole was not recursed upon, since no two mouse holes would ever be in view at the same time. Size was once again used to estimate distance and utilized a similar linear curve.

The only other thing our Vision system looked for were bar codes. Initially we had decided to ignore bar codes all together but after a bit of thought, we decided that they would be useful for signaling to the robot that it's seeing the same area over and over again and should attempt to find a new part of the playing field to explore. In order to obtain accurate readings of the barcodes we did the following. First, the image was scanned for any green pixels below the blue line denoting the top of the wall and the min and max of those pixels were recorded. Next, this newly constrained area was searched vertically for the first and last black or green pixel. With these four positions we formed a bounding box of the bar code and split it into five sections.. These five sections were then randomly sampled to determine whether or not they were green. We read the bar codes from top to bottom and treated them as binary numbers, with green being one and black being zero. Once a number for the bar code was computed, it was checked again all possible bar code numbers . If the number corresponded to a valid bar code then this information was based to our Mapper subsystem, otherwise it was just ignored.

The State Machine took care of controlling the bott's actions. It utilized subclasses that implemented an abstract Behavior class and a run() method that would perform the behavior, which created something of a finite state machine written in a functional programming style, as we wrote the machine with essentially zero state variables. Before a behavior finished, it would queue the next behavior of its choosing. If it left an empty behavior stack, the Manager behavior would queue a default behavior, which in Scoring mode was an optimized exploration behavior. This exploration algorithm would set the left and right motors to speeds that correlated to the range readings of the left and right IR sensors, respectively, which on good days led to an efficient wandering behavior. The lack of state variables also prevented getting stuck in one state. Unfortunately, the state machine did not receive enough debug time due to the acidulous issues that beshrouded the middle wheel, gyro and IR sensors. I (Jason) also lost code written in the afternoon before impounding when our Eden blew, so I was perhaps the most shocked team member when I heard the news that we balled it up with three points.

The only other thing our Vision system looked for were bar codes. Initially we had decided to ignore bar codes all together but after a bit of thought, we decided that they would be useful for signaling to the

robot that it's seeing the same area over and over again and should attempt to find a new part of the playing field to explore. In order to obtain accurate readings of the barcodes we did the following. First, the image was scanned for any green pixels below the blue line denoting the top of the wall and the min and max of those pixels were recorded. Next, this newly constrained area was searched vertically for the first and last black or green pixel. With these four positions we formed a bounding box of the bar code and split it into five sections.. These five sections were then randomly sampled to determine whether or not they were green. We read the bar codes from top to bottom and treated them as binary numbers, with green being one and black being zero. Once a number for the bar code was computed, it was checked again all possible bar code numbers . If the number corresponded to a valid bar code then this information was based to our Mapper sub system, otherwise it was just ignored.

We made the Mapper to record general data about the area. It had two means of collecting data: incidental data we collected while moving, and explicitly desired information from 360 degree Scans. Specifically, we used the optical encoder readings from the middle and right wheels coupled with the nomadic gyro readings to garner a rough idea of where the bot had been, and used this map after 360-degree scans to explore areas with mouse holes or areas that we hadn't previously been to. We started these scans when the optical encoders told the state machine that the bott had been traveling for more than 10 meters without seeing a ball, when we resaw a bar code that we saw a while ago, or when we got stuck.

These Scans had the advantage that we could use them to map out a local area, centered on the scan point. Eventually, the collection of local areas could be collected into a global area. It should be noted that duplicates were not considered relevant in all of this; two local areas could very easily collide. This was not a problem, partially because we were planning to use the global map mainly to find mouseholes not in view. By recording which scan centers could see mouseholes, we could attack mouseholes more aggressively, being able to go to any visible mousehole or, barring that, a visible old scan center from which a mousehole was visible. We also developed a framework for reading and storing Wall Dots, globalized individual distance measurements from the IR sensors. However inaccurate these would be, it was clear that this data would be useful to keep Jimboo exploring new areas. Unfortunately, there were more pressing concerns that had to be addressed, and so we did not have time to implement a proper use for these, nor were we able to really implement the rest of the Mapper either, due to gyro unreliability (caused partially by a that fateful faulty OrcBoard perhaps?), among other problems.

Conclusions

Overall, our robot scored 3 points, a number that, given the fact that our computer and orc board blew up the day before the contest, we're extremely pleased with.

In conclusion, we feel we did pretty well considering all of the pitfalls we faced along the way. The robot took a little longer than expected to build, something I personally attributed to the fact that lab closed fairly early the first 2 weeks of IAP. Really, that is the time that it is most important that the lab be open for long hours, not the end. We need the staff most then, to help deal with random technical

issues that crop up, and we need the tools most then, too, to get the robot's physical body done and finalized as fast as possible, so we can get to the coding. Personally, if we had been able to get more body work done on our robot earlier, I wouldn't care if the lab closed at 9PM for the last week, since it's mostly coding and testing then anyway. So long as we have access to fields, like the one in the 6.001 lab, we're fine. Another rant: there's something funny about those IR sensors, or at least about how the OrcBoard interacts with them, considering how often they failed. Teams just cranked through those guys (and every time one broke while the lab was closed and we were unable to get a replacement we wasted a night. I doubt you want us to hoard spares, but it's tempting.) As for suggestion, I recommend that future teams come up with a vision and stick with it unless it proves to be impossible. Not a lot of people believed we could get the trans-drive working but getting it completed and performing properly was as satisfying as winning the contest.
