Team Ten Paper

Mechanical Design

One week into MASLab, our entire team met together for the first time and brainstormed ideas for our robot's design. We agreed that we wanted to try to score field goals, but weren't exactly sure how. After discarding many other ideas, we decided that we wanted a simple design with no lower holding area so that we could capture balls just by running over them. We also wanted a continuously running lifting device, which would avoid the need to know when we captured a ball.

Early in week two, we began experimenting with prototypes. We started by remounting the wheels and castors on our pegbot to create the clearance necessary to allow balls under our robot. We also built a funnel of sorts to guide balls to a specific area under our robot and feed them into a lifting mechanism. This initial prototype was successful at directing balls that we ran over. When we built a similar funnel on our actual chassis, we changed the angles slightly to make room for other components and found that it was less effective because the balls bounced off of it more. However, this never proved to be a real problem because our eventual lifting mechanism also helped direct balls.

Once we knew that we could guide balls in a given direction, we started planning a one-way gate for the front of our robot, to prevent loss of balls if we were to travel backwards soon after capture (before the balls reached the lifting mechanism). The main challenge here was to create a gate that was sturdy, but had low enough resistance to always allow a ball to push it open. Knowing that the gate could be our Achilles heel if it failed to open, we thoroughly investigated hinges and materials before building it. This lower gate (so named to avoid confusion with the higher, servo-operated release gate) likely saw more iterations than any other mechanical element of our robot. Initially, it was constructed from a thin horizontal strip of polycarbonate attached to hinges on the front of our robot, with short brackets extending down to prevent it from moving forward. This design was probably not inherently flawed, but it caused us much pain. The main source of distress was that when measuring and designing the gate by hand, we had limited accuracy in the alignment of screws. As a result, when we tightened the screws on the hinges, the gate would twist slightly, causing resistance that a ball couldn't necessarily overcome. We tried tweaking this many times, and eventually concluded that we would have to leave the screws just slightly loosened. Another problem was that it was hard to make brackets that were long enough to effectively keep the gate from moving forward, yet short enough to provide clearance for balls to move under them. The final gate was actually similar to the initial gate, with the exception that it featured a metal bar for reinforcement on the polycarbonate strip.

The most daunting element of our robot was the lifting mechanism. Initially, we hoped to build an Archimedes screw to turn continuously and lift balls. We prototyped a screw and found that we could certainly create one (although creating one that actually worked would be more of a challenge). However, our desire to have no lower holding area made it hard to find a way to coerce balls into the

screw. We also considered a chain or belt with discs or rings to lift balls, but we still couldn't figure out how to get balls onto it. Finally, we came up with the idea of a triangular belt that would initially contact balls while traveling from the front of the bot to the back and would then push the balls up a chute in the back. This seemed like a solution to our problem because it would already be pushing the balls when it started lifting them.

The first task in constructing a belt, which proved harder than we expected, was finding a suitable material. Ideally, we wanted a preconstructed belt or chain that already had flights to move the balls. After an afternoon of searching at the lab, a conveyor belt warehouse, Hubba-Hubba, and Cambridge Bicycle, we ended up with a bike chain and a plan to weld flights onto it. When our welder broke, we decided to instead remove the pins that hold the chain together, make flights from steel wire, and attach the flights by inserting the wire where the pins would normally be. The chain was powered by a gear at the back of the robot, which was attached to a drive motor by a metal tube (held in place with both screws and epoxy). At the front and top corners, the chain traveled over fixed rollers (also constructed from metal tubes).

The two main problems with the chain were that it tended to slip off the gear and the flights got caught on other parts of the bot. The problem with slippage was so bad that we almost decided to punt the chain entirely in the final week. However, we fixed it reasonably well when we realigned the gears and rollers. The catching problem seemed less serious at first but turned out to be more persistent. We could adjust the flights and file down the parts of the bot where the chain generally caught, but since the chain was never entirely taut, the catching was inconsistent and hard to predict. After tweaking the chain many times over, we took care of remaining catching problems by implementing a function that used current sensing to tell when the chain was caught, reversed the chain for a short time, and then moved the chain forward again. This worked well because the chain's path was so inconsistent that it rarely caught in the same place twice in a row.

Under the chassis, the chain and funnel guided balls onto a polycarbonate ramp, which was reinforced with sheet metal. The ramp was about 1/4" off the ground so it wouldn't catch on the carpet of the playing field, but was flexible enough that a ball could push it down to floor level and roll onto it. The ramp was attached to a vertical section of PVC pipe. The pipe served many purposes, including guiding balls as they traveled upward and providing a mount for both the upper roller and top ball-collecting tray. Approximately one third of the pipe was cut away along the entire length to let the flights fit into it.

Above the pipe was a tray made from polycarbonate with short walls from thin wood. The tray was also shaped like a funnel, closely resembling the funnel under the robot, and was angled down slightly. (The top of the pipe and tray were about 13" off the ground, while the bottom of the tray was around 12.5" to fit over a field goal.) Because the chain tended to launch balls out of the pipe, we had some trouble with balls bouncing out of the tray. In order to keep balls in, we made a "hat" for the tray from a piece of plastic cut from a 2 liter Mountain Dew bottle. The hat was attached to the tray with screws and cable ties, and could be easily compressed to fit in our tub. Because the hat was a late addition, it wasn't tweaked as much other elements of the bot, and in the competition a few balls got caught next to the hat and did not travel down the tray. While this was never a real problem because we didn't actually

approach a goal and attempt to empty the balls, it showed that we should probably have put more thought into the design of the hat.

Near the front end of the tray, a gate made from sheet metal kept balls from falling out. Upon docking at a goal, a servomotor was to lift the gate and release the balls. The servo was he only electronic part of the robot that worked consistently. While I don't know the true origin of the word servo, I like to think that it comes from the Latin word servo, meaning "I serve" or "I save", because it certainly served us well.

Overall, the ball collecting mechanism worked quite efficiently (once it worked). The robot needed only to run over a ball. The ball would travel through the lower gate, catch on the chain, get pushed onto the ramp and up the pipe, pop out of the pipe and (hopefully) land under the hat, travel down the tray, and rest behind the gate until the robot found a goal. In retrospect, the only serious problem with the system was that it took so long to construct that we had little time to learn how to wander around and find balls and goals.

Sensors and Strategy

The robot uses its sensors to behave differently depending on its environment. For instance, it uses short-range IR sensors on its sides for wall following. The sensors are mounted above the goal line to allow wall following even when the wall turns into a goal. The robot moves forward while attempting to maintain a certain short distance from the wall. To do this we set an outer and inner bound on how far away the robot is allowed to be from the wall. When it is driving exactly in the center of these bounds, both wheels will be driving forward at 100% speed, as gets farther away from the wall, the wheel closest to the wall gradually slows down and as it gets closer to the wall, the wheel farthest from the wall will slow down. This proportional feedback keeps the robot an appropriate distance from the wall. The only problem occurs when robot encounters the inside of a sharp angle. This means that it will collide head on with the wall and the IR sensor values will not change.

To solve this problem, three bump sensors are positioned on the front of the robot. These let the robot know when it has hit a wall from the front. When the left bumper is triggered, the robot turns right. When the right bump sensor is triggered, the robot turns left. The amount of turn is typically 45 degrees. This would help the robot if it were moving forward either by wall following or random movement, and it hit a wall at an angle, it would turn away from the wall. With a 45 degree turn, we still have the problem of the robot going into the inside of a sharp angle. Imagine the situation in which the right bump sensor triggers on the right wall, and it turns left only to have the left bump sensor triggered on the left wall, and the cycle repeats. A simple solution to this problem is to make the robot turn more than 45 degrees sometimes. We chose one third of the time chosen at random to turn 225 degrees to avoid getting stuck in acute angles. Another intended purpose of the bump sensors was to detect when the robot was lined up to score field goals. Since the middle bump sensor was mounted slightly farther forward from the other two sensors, we would know that we could release the balls when we were lining up with a goal and the side sensors but not the middle were triggered. As it turns out, it was very

difficult to line up so precisely that this occurred, so we modified our check into making sure that one of the sides was triggered but not the middle. This was not as good and did not always work.

We decided that wall following alone was not a good way to explore most of the field so we incorporated random motion into our robot as well. It chooses between options such as moving forward, turning, backing up or switching into wall following again. We had global time-outs so that our robot could not be in any one of these modes for too long. The random behavior and wall following can only be disrupted by seeing a ball at any point, or by seeing a goal in the last minute of the game. Moving towards goals takes priority over moving towards balls in the final minute. When our robot moves towards an object, it will stop, take a picture, turn a small set angle, and then stop to take another picture. It only moves forward when the object is in a small window in the center of the screen. After moving forward, it stops again to reevaluate the situation. It stops being able to see a ball when it is within a few inches of the robot, so when this occurs, the robot drives straight forward, plowing over the ball, hopefully. The robot made all of its specific-angle turns by stopping, calibrating its gyro for 200 milliseconds, turning, and stopping once it the gyro has changed enough (or it times out). We chose not to trust the gyro for global angle readings, and only for these just-in-time calibrated local readings.

Vision and general software issues

We tried several approaches to our robot's visual processing. Our code was based loosely off of the "image tutorial" we were given over winter break; unfortunately, MASLab made major changes to the version of Java used and its image libraries before IAP started, and we ended up wasting a lot of the first week rewriting already functional code to work with the new library. We also got inspired to use all sorts of fancy and new-fangled features of Java, from generics to abstract classes and interfaces and so on. While our initial code was very clean in some sort of theoretical sense, it was also nearly impossible to write anything that worked in it, and it was pretty slow. As the month went on, our code evolved from abstractly nice to actually working if somewhat incomprehensible.

We initially had a separate thread that continuously captured and analyzed images, presenting its results in a few synchronized variables like a list of "ball ScreenLocation" objects. We found that this was too inefficient, mostly because the camera thread had to search for every type of object that we wanted to recognize even if at certain times we would have preferred to only look for balls, or to only check if there are a bunch of red pixels somewhere on the screen. We probably could have fixed this by giving the camera thread some flags to read to decide what to do with its time, but instead we moved to make our bot capture and analyze pictures on demand in the same thread. This made the capturing itself a bit more efficient, so that once the bot had actually seen a ball and stopped to look at it it was very good at aiming itself precisely at it; however, it did mean that it often was too slow to notice balls as it rotated past them until too late.

Our image scanner searched the image (usually downsampling by only looking at one out of every two or three pixels) for pixels of the color being searched for (usually either red or yellow), doing blue-line filtering as it went. The picture's data was first converted into three arrays of floats called hues,

saturations, and brightnesses. The scanner could either return that it had "enough colored pixels", or if asked to could try to conglomerate them into connected components. It was also able to discard images of balls that lie under images of goals, since we would not want our robot to try to get balls that have already scored (even though we weren't going to aim for the low goals).

While we tried to put in as many optimizations as possible, we had to remove some of them close to the competition. For example, we originally had it so that the trigger to switch from random or wall-follow mode into ball-seeking mode would be just to see enough red pixels in the image, instead of actually doing the time consuming connected-components analysis. At the same time, in the ball-seeking code, if no balls were found, it would switch into random mode. Unfortunately, this meant that if the bot was seeing enough red pixels that there could hypothetically be a ball, but not actually recognizing a ball (say, a bunch of red pixels scattered over the field), it would switch back and forth continuously between random mode and ball-seeking mode without actually doing anything. Thus, we had to add the more expensive connected-components check to the code that decided whether or not to switch out of random or wall-follow mode.

We initially planned to have our own sophisticated time-sharing system that would allow multiple strategy objects to be active at the same time (but still executing sequentially in one thread). Unfortunately, the emergent behavior of such a system ended up being a bit too confusing and we eventually ripped out that entire system and replaced it with a single strategy which was more or less a finite state machine.

General Concerns

One problem we had to deal with was the fact that for the first week, we were never all four there at the same time. (In fact, Dave and Ali literally passed each other at the Kendall T on the way to and from the airport respectively.) This gave us a late start on overall brainstorming. We spent so much of the first week and a half focused on short-term goals like fixing our OrcPad and getting our pegbot through Checkpoint One that we didn't really sit down and think about strategies and design until midway through the second week. In addition, our triangular chain design, while effective and pretty damn cool, put such space constraints on the placement of everything from the sensors to the battery that we didn't really know where everything was going to be placed until the chain was completed in the final week. We could have gotten around this if we had pre-planned our robot in CAD and created it with a waterjet, but we chose to build our robot by hand. The biggest problem came from the fact that every time we were able to test the strategies of the robot, the camera and sensors were mounted in different places. In retrospect, we underestimated the difficulty of creating the chain. That said, it was still worthwhile to have the chain, which is the element of the robot that we are most proud of.

Another problem involved the division of effort. While we all put in roughly equal work on the robot, and we're proud to say that we had no real personal squabbles, we ended up dividing the work in ways we didn't expect. We originally wanted everyone to work at least a little on the code, with Dave and Ali taking leading roles there. However, we ended up having only Dave actually write code, though much of

the content of the code ("if the robot senses that the wall is too close on the right side, it backs up by only turning its left wheel", etc) was supplied by Ali and other team members. Had we actually been able to collaborate fully on the code, we may have been able to get a much more sophisticated system in place. In addition, having Dave focused mostly on code left him somewhat in the dark when it came to the robot's mechanical systems, and so during the many days where the robot wasn't in good enough shape to test code, he mostly sat around doing nothing.

Conclusions and Suggestions

So come Judgment Day, sorry competition day, what were our thoughts? Were we ready for this portentous event? A mere three days prior, as our robot languished in non-extant conditions, our expectations were somewhat limited.

However, on the day, thanks to a somewhat loose (read lenient) interpretation of the game rules, our hearts filled with pride as Swiss Cheese Bot's first public steps were taken. Accompanying the joy in our hearts was the solid sound of our clanking chain-mechanism. Five balls were captured in this glorious fashion.

Removing the rose-tinted glasses, it was pleasing to see a successful mechanical design, and a competent ball-finding/aligning/collecting system. However, the goal-finding approach could certainly do with improvement. The one minute time period selected for goal-search was somewhat short on reflection. Further, goal-approach appeared poor on the day.

So, from our somewhat unelevated pedestal, what advice can we give to future MASlab-ers? Firstly, a well-planned (and executed!) design would be helpful. We erred too much on physical trial-and-error on the balance between planning and doing. This resulted in severely restricted programming access to the robot. It cannot be emphasized enough the importance of the software design aspect in MASlab.

To prospective robot-handlers, if you have read this far, then you have surely recognised just how valuable past experience is: make full use of the staffers next year, listen to (if not always follow) their advice. Read the other final reports.

Lastly, hope is a vital element in the build-process. As the deadline hurtles towards you, do not lose faith: everything will surely come together in the end. At least know this -- others have gone through the same distraught and trepidation, and have emerged bathed in the warm pride gained only through honest effort.