**Description**

As described in the Lab instructions, the first thing we did was install the sonar sensor support, the sonar sensors, modified and installed the sensor library and attempted to obtain range measurements. We immediately ran into issues as the sensors were reporting accurately only up to 5cm and would spike their measurements to upwards of 300cm when the true distance was lower than 1cm. After we removed the sensor from the support we realized that the sensor data was becoming more accurate, and that anytime we remounted the sensor it would give the incorrect results. We eventually realized that the support was pushing the sensor components together and whether the issue was that the receiver was pushed into a bad angle or that the movement disconnected a bad solder joint was never discovered as we received a metal scraping tool and removed excess material from the sonar sensor support, hopefully making the issue moot.

While we were measuring the sonar sensor, the sensor was normally accurate between 0 and 1.5 cm of the true distance away, with the only exception being 100cm with a delta of 2.79 cm. After using a best-fit linear line algorithm (least squares), we had a line with a slope of 0.9855 with a perfect slope being 1.0 that only left us with an error of 0.0145.

While actually using our calibrated sensor to drive in a maze, we used many different algorithms. We tried using the naive approach, which was to have the robot move forward until it reached a certain threshold with the value returned from the sonar sensor, and then rotate 90 degrees clockwise then loop. This led to us realizing just how bad Brian’s difference in left and right speeds were and we attempted to correct the issue with basic changes to the duty cycle argument of the software implemented PWM system. This did not work well enough. Brian consistently showed that there was no discrete setting to set Brian to that would not cause him to drive on an arc and ultimately run into a wall.

We then tried a “wiggle” tactic, which would hopefully reposition Brian into a more appropriate position in the lane. This was promising and could have worked, but felt like a band-aid on the issue rather than a solution.

Finally we tried a full circle relocation that would rotate 90 degrees, determine the distance from the wall, rotate 180 degrees, determine the distance from that wall and then determine if he needed to backup or move forward and finally rotate 90 degrees and continue forward. Unfortunately without reliable turning this ended in failure.

Finally, we tried tweaking the frequency of the PWM. The duty cycle was just a number taken between 0-99 to indicate percentages, but the frequency was a constant at 10,000. This gave us significantly more values to choose from and allowed us to use the same duty cycle for the left and right motors while the frequency accounted for the difference in power input to power output of the motors. This worked quite well and Brian was able to drive in a relatively straight line, but because of the fact that the library we were using was using #defined, hardcoded constant as their frequency (which in my opinion ought to have been a template argument), we were forced into reimplementing most of the methods in the library under the namespace “brian”.

Once Brian could drive in a straight line, we tried him on the track and found that he’d randomly decide to turn. After some debugging with the serial connection, we found that Brian randomly spit out sonar values in the 10s or 300s, so as a “fix”, we clamped the read-in value to 100 and took only the average of 5 values over 250 ms to make a more stable output of the sensor. Brian still did sometimes “see things that weren’t there” which wasted one of our attempts during the competition, but this is probably due to one of the issues already listed above.

Our final issue was that, as we tested Brian again, and again, we became aware that the power of the battery was also a major influence on the accuracy of Brian moving straight or turning 90 degrees appropriately. Making the function of applying equal force to the left and right motors a function of frequency, duty cycle, power, and battery power. Given that there was no immediate way to determine the battery power we simply got Brian to go around the track once using the naive algorithm and put in the towel as there was no more time left to attempt different methods.

Our final discovery however was that Brian’s rear, left (if looking on top) motor was considerably slower than both the front left and front/rear right motors. This was a deciding factor that caused brian to rotate off center quite dramatically. Knowing this we may have been able to perform better in the actual competition as Brian drove incredibly straight but over or under rotated seemingly randomly.

**Summary**

Given that we’d know the dimensions of the maze and did not have the difficulties of reliably rotating that we have, Brian should be able to race considerably better without considering the sonar sensor data, because without it, Brian could simply rely on compiled-in constants and ambiguities in performance would be solely based on hardware.

Some simple things that would improve Brian’s performance would include: a new set of rear wheels, hopefully spinning at significantly closer speeds; a battery sensor to indicate battery power, so that we could estimate the function of driving straight and turning; measuring the (x, y) offset of rotation and simply further inquisition of how to make Brian turn tighter; a sensor system that didn’t arbitrarily send bogus values so regularly and/or a better algorithm that disambiguated real data from false data.