James Lee, Navjot Singh, Andrew Michelsen, Carlos Flores, Diane Wang, Diana Rodriguez, Abha Pandey

Oak Grove Eaglebots Team 11-0170

romanoffj@esuhsd.org

New Code? Oh Boy!

Introduction

When the new version of KISS-C came around, our team had a very difficult time deducing the cause of the myriad of errors that were present in our driving and turning routines. After examining the changes and updates that KIPR implemented in KISS-C compiler, the problem originated from our extensive use of global sensor update function calls, which are now replaced by the "get_create_insert_command_here (float lag)." With only 7 days remaining until the Northern California regional competition, our team had to fix the problem—fast.

Driving Forward

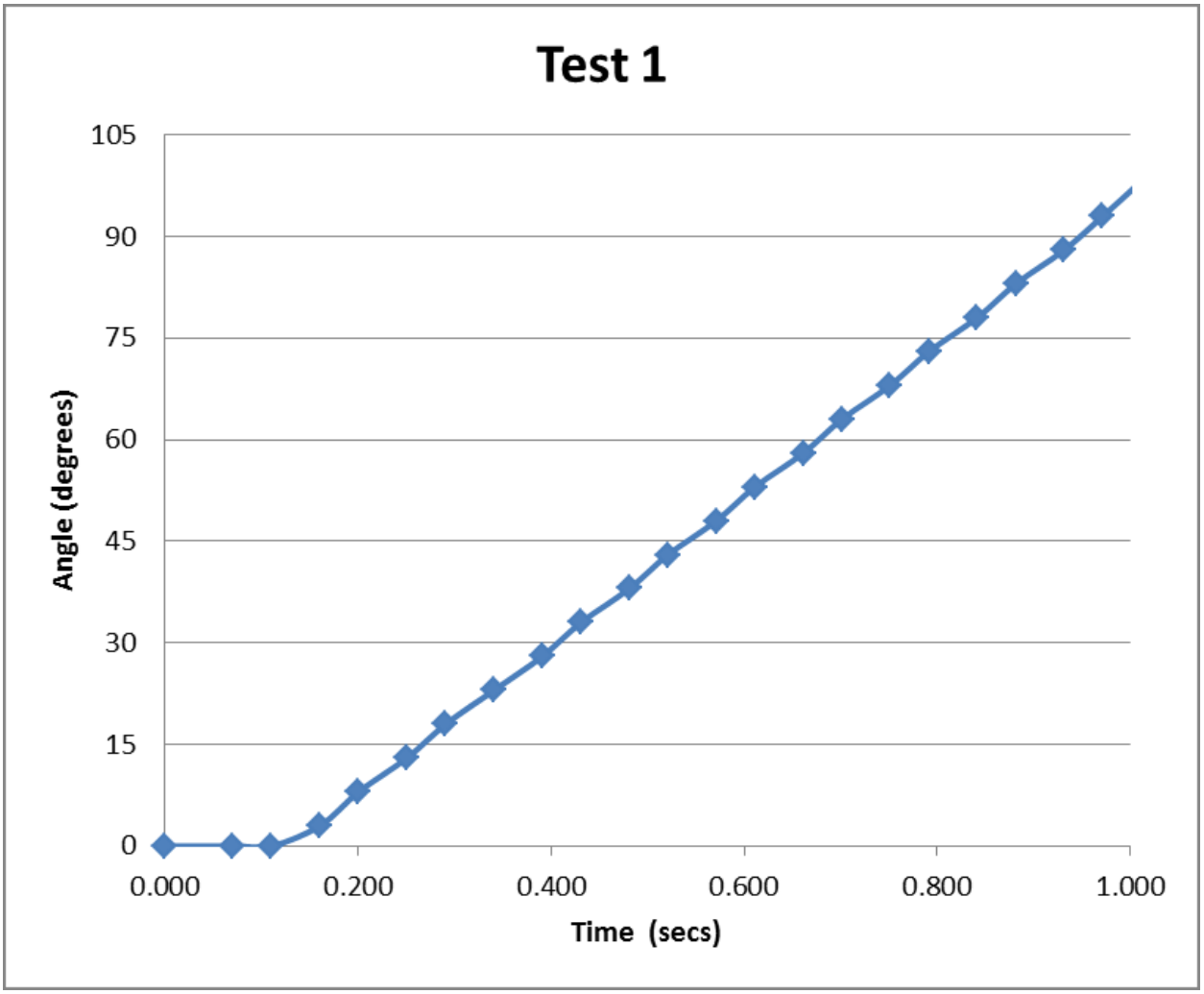
One of the first tasks that I did was fixing the iCreate driving routine. I took all of the previous "create_distance" and "create_sensor_update" functions and replaced them with the current "get_create_distance(float lag)." Afterwards, I put in .1 as the lag value, told the iCreate to go 14 inches, and ran the drive routine. Unfortunately, I did not fully understand the new distance update function; the iCreate ended up going 15.3 inches, an error percentage that was unacceptable to me. After looking into the library files that were included with the KISS-C compiler, I found out that the current sensor update caches values and uses the cached values when called too quickly. I do not understand why this was implemented, but I trust the judgment and foresight of KIPR. I put -1 as the lag value to force a new value to be updated and everything worked. I put -1 down because of my fear that the CBC might delay the clock, but no tests were conducted to validate that precaution. The iCreate now can drive forward accurately with little tweaks here and there to adjust the straightness of the drive train. This was one hurdle that I jumped over, but the turning routine was a hurdle that I may need a ladder to cross.

Turning

I tackled the turning thinking that it was as easy as the driving routine. I was horribly wrong. The iCreate would spin incessantly, mocking me every time it spun. After numerous attempts at trying to tweak the routine, I decided to scrap the routine and make a new one. A thought came to me to use time as a deciding factor to turn. I doubted myself, because time is notorious for failure. Using time accurately is a challenge, a challenge that I would gladly take on. Also, we only had 4 days left to build and drive the iCreate.

After an hour of brainstorming and with some help, I decided to log the values: inputted angle, current angle at specific time intervals, overshoot angle, time since the function is called, and the time when the function ends. (The program will end when the iCreate stops turning.) I used arrays and pointers to store information at each section of the turn, then print the values out at the end of the function. I also found out that printing while the function is running slows down the processor.

The values that were printed onto the display were a huge milestone. From that data that I accrued, I found out that the CBC has a delay period between when the function is called and when the iCreate moves. Also, the time since movement, and the angle turned correlates in a linear fashion. This is the graph that was produced with the information that was gathered in the test.



This graph explicitly shows that there is a period of time where no movement is recorded. (The actual inactivity is 0.1401 seconds). This information facilitates the prediction of when to stop the iCreate. Coupled with the how much overshoot there is, turning based on time has

become a much easier endeavor. Testing different speeds yields similar delay periods, but different overshoot degree values. By adding:

```
#if (100 == g_turnSpeed)
        #define CALIBRATION_CCW_TURN_DEGpS (39.959 * 382 / 360)  // degrees per sec
        #define CALIBRATION_TURN_B         0.1401  // seconds  (delay)
        #define CALIBRATION_OVERSHOOT      5       //degrees
#endif
#if (250 == g_turnSpeed)
        #define CALIBRATION_CCW_TURN_DEGpS (109.9 * ( 360.0 + 11) / 360)  // degrees per sec
        #define CALIBRATION_TURN_B         0.149  // seconds  (delay)
        #define CALIBRATION_OVERSHOOT      11      //degrees
#endif
#if (350 == g_turnSpeed)
        #define CALIBRATION_CCW_TURN_DEGpS (161.0 * (360.0+58)/360) // degrees per sec
        #define CALIBRATION_TURN_B         0.153 // seconds  (delay)
        #define CALIBRATION_OVERSHOOT      16      //degrees
#endif
#if (400 == g_turnSpeed)
        #define CALIBRATION_CCW_TURN_DEGpS (173.16 * 440 / 360 ) // degrees per sec
        #define CALIBRATION_TURN_B         0.159  // seconds  (delay)
        #define CALIBRATION_OVERSHOOT      22      //degrees
#endif
```

from various tests, I can accurately predict when to stop the iCreate when turning at the respective speeds.

Conclusion

Overall, this was a very important learning experience, because I am now proficient with pointers and arrays when previously I was not. As a result of this testing and tweaking the iCreate can now turn and drive very accurately, which is a very important asset to the reliability of the robots. Furthermore, the log functions that I have now also provide a basis to record data and display them when I need to test values.