To: Dr. Berry
From: Peter Heath, Matthew Schack, and Data
Date: 12/13/15
RE: Lab 02 Location and Odometry

The purpose of this lab was to develop a program to make Data remotely programmable for various movement tasks. The specific tasks we had Data do were move to a specified angle, move to a specific coordinate, drive in a square, drive in a circle, and drive in a figure eight. To tell Data which function to do we used our IR remote.

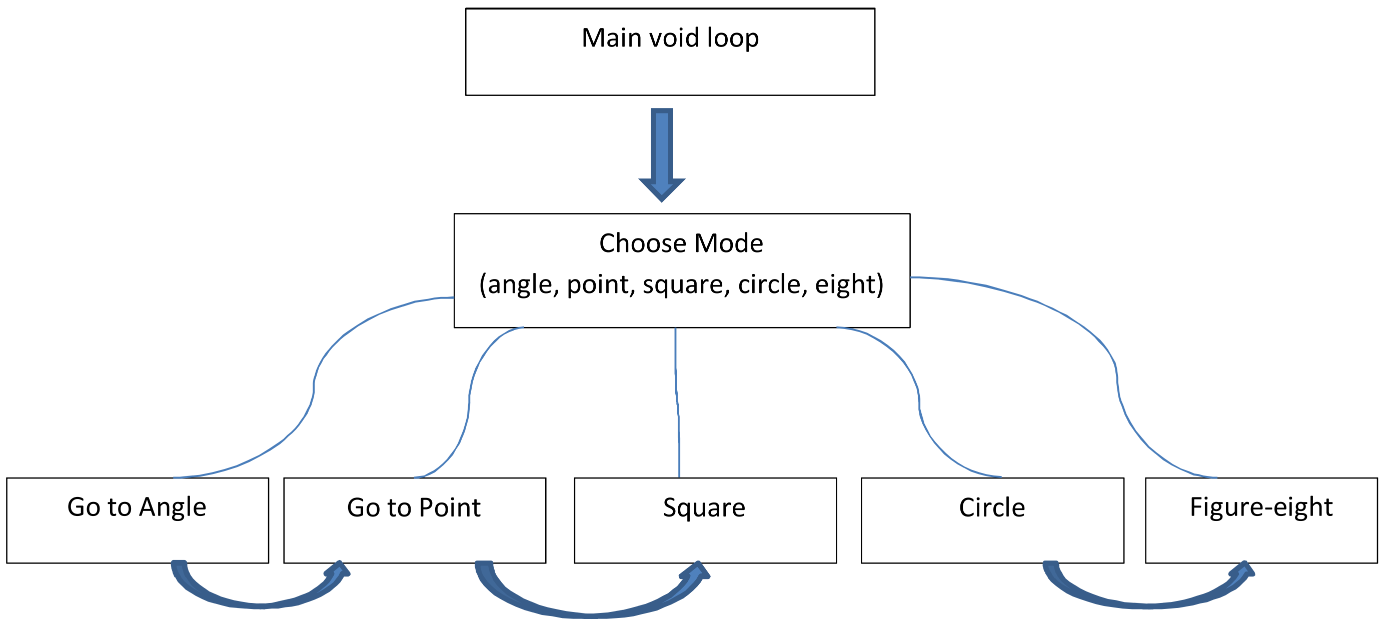
The first thing we implemented on Data was the go to angle behavior. With the go to angle function we had to take an angle which Data would move to and then based on Data's current position calculate the angle needed to move and then move that angle. To calibrate this behavior we guessed at how long the motors would have to run to get to a desired angle then using that tested and corrected the number until we were satisfied with Data's performance. This method is not overly robust due to the open loop nature of our algorithm but without sensor feedback there is not much more we could do to improve it. Our algorithm is best described by this equation:

$$TimeToRun = Angle * AngleCalibrationTime$$

With the go to angle we got very good results on our calibration angle which was 90 degrees but as the angle became significantly larger or smaller than 90 degrees the change in momentum of Data caused our accuracy to deteriorate.

The next function we implemented was the go to point function. In this function we had Data calculate what angle it needed to move to and then the length of the straight line path it would need to move along. Once Data has calculated what it needed to so it could move it executed a go to angle and then drove the necessary amount of time to reach its desired location. The equation for this was very similar to the go to angle equation using only a different calibration factor. The go to point function seemed to work pretty well for all of the cases we encountered if the momentum errors for go to angle were ignored. The only major errors we saw were that the wheels would move at just slightly differing speeds which over long distances caused course deviations. The best way to eliminate these issues would be to add some kind of feedback controller using encoder data or some other kind of odometric feedback. In our go to point method we had data turn and then drive as opposed to trying to turn and drive at the same time, our reasoning for this was because it seemed much easier to implement as well as more reliable since we had it use a function we already developed. Our odometry errors came from imperfections in the motors and surface that they were driving on, with some kind of feedback control these can be all but muted.

The next function we implemented was the circle which we just had Data drive in an arc for a calibrated amount of time so it would return to its original position by completing a circle. Using the circle command we were then able to do our figure eight command as well by tying two circles together with opposite directions of rotation. The last function we implemented was the drive square function which used four instances of the go to point function to drive in a square. All of these motions could be improved with odometric feedback from the wheels as well as by using the compass potentially to hold true angles. With those two forms of feedback we could have achieved much more precise location control over Data but even without these feedbacks we were able to achieve reasonable results through calibration timing.


Main void loop
Choose Mode
(angle, point, square, circle, eight)
Go to Angle
Go to Point
Square
Circle
Figure-eight