Experiment #2 – Functions and Calibration

Objective:

In experiment #1 you learned how to turn the motors and control directions. The objective of experiment #2 is begin the build of your basic motion library. You will write and test functions to perform each of the 6 primary motions. The 6 primary motions can be programmed into 3 functions if you can pass a direction “CW/CCW” variable to the function. Once the functions are written and tested, you will calibrate them so they may be called with meaningful numbers.

Preliminary setup

In experiment #1 you learned that your motors are controlled by 8 – output pins on the Arduino. You also learned that these 8 bits are addressable as PORTL in your software. To enable the 8-pins you must declare them as outputs in the port data direction register “DDRL”. This is done in the setup() function which acts as the main function in the Arduino.

4 of the output pins/bits control the directions of the 4 motors. 4 of the bits actually drive the motors by repetitive toggling. Each of the 6 basic motions is programmed by a “byte” type variable. The motion is also controlled by a “byte” type variable. These 7 variables should be declared and initialized as global variables so they are available to the functions without the overhead of passing them. To declare them as global variables simply place them above the setup function. For example

```
byte motion = B01010101;
setup() {
DDRL = B11111111;
```

The other 6 byte variables should be available in your notes from experiment #1. Enter them as global variables and initialize them.

Functions

In experiment #1 you found that you could turn the motors by using a loop to repetitively toggle the 4 motion pins of PORTL. This was done with the command

```
PORTL ^= motion;
```

Each motion function at a minimum must control direction, speed, and distance. The direction is controlled setting PORTL to the desired direction byte. This is done once at the beginning of the function before the motion loop. The speed is controlled by the delay inside the loop and the distance is controlled by the number of steps of the loop. See sample code below.

Calibration

The total amount the wheels rotate is proportional the number of “steps” in the loop. To control the distance you must calculate and/or calibrate the numbers of steps per unit distance. To be able to address this value in fractions of an inch you must use float type variables to calculate the actual number of steps for a desired distance. The scale we are operating on is best served using a base unit of inches. The calculation for number of turns per inch is straightforward.

> The wheels are 60 mm in diameter
> There are 25.4 mm per inch
> The motors are in ¼ step mode so a full rotation requires 800 pulses
> Each step requires one cycle of the loop
> So for 1 rotation
> Distance = 60*pi/25.4
> Steps_per_inch = 800/Distance

Pass the desired distance to the function in inches and multiply by Steps_per_inch to calculate the required number of steps. Perform the calculations in floating point using a float type variable. Then convert it to the int variable “steps” by direct assignment.

The above calibration works for forward, reverse, and strafe left and right. For rotation you should pass the function a distance argument in degrees and empirically determine the steps_per_degree. The actual direct calculation can be done given the radius of the wheel centerlines to calculate the turning circumference. This is left as an exercise for the assertive student. To empirically determine start with an estimate of 20 steps_per_degree and test the robot on the calibrated field. Adjust the variable based upon actual motion.

For initial tests pass the function a 90 degree value and test until the results looks good. Then advance to precision calibration. Pass the function 1080 degrees and measure the amplified error. This process is required to obtain degree level accuracy without using additional sensors.

Procedure

1. Program all preliminary global and setup variables
2. Write functions one at a time, start with forward motion, then rotate and strafe.
3. Debug and test one at a time
4. Calibrate one at a time
5. Repeat steps 2-4 for each function

Testing

Program, edit, compile, and download you program onto the robot. Place the robot on the calibration board under the blackboard. Align the robot to the painted gridlines and turn the robot on. The painted lines are on 12 inch centers squared at 90 degrees within 1/8th inch overall. You can align the chassis or wheels to the lines. For straight line motion distance measurement the chassis is the most reliable datum. For rotation the wheels are the most usable indicators of alignment because of their visibility.

Sample code

```
* mov the robot without tracking or interrupt
 */
void mov( byte dir, float dist, long del){
  PORTL = dir;
  float stepf=dist*steps_per_inch;
  long steps=stepf;
  for(long i=0; i< steps;i++){
        delayMicroseconds(del);
        PORTL ^= motion; //toggle the step signal to continue moving
                                          }
                                                  }
```

The mov function takes three arguments

1. Byte dir – motor direction byte
2. Float dist – distance to move in inches
3. Long del – the delay period between steps in microseconds