Lab 03

Wall Following: Feedback PD Control Worksheet

Robot Name Murphy

Team Member Name: Jackson Seida

Team Member name: Justin DeWitt

Purpose

1. In your own words, state the purpose of Lab 03 in the following space. What behaviors are you implementing on the robot?

   The purpose of lab 03 is to code the robot to follow a wall and follow 2 walls in a hallway, utilizing bang bang and pd control, and to create a go to goal function that has obstacle avoidance.

Part 1 – Follow Wall (Layer 1)

Bang-Bang Control

2. Did you decide to drive your robot forward or backward, how did you decide? What were the pros and cons?

   I decided to drive the robot backwards because it allowed the robot to more effectively follow the wall. Only the lidar sensors were working sufficiently so those were the only sensors utilized. This made it so when the robot was driven forward it would often lose the wall.

3. How far and how long was the robot able to follow the wall between 4 and 6 inches without losing it?

   The robot could follow the wall for a seemingly infinite amount of time. This is because the robot was moving at a slow speed so the control system would not become unstable.

Proportional Control

4. What proportional gain did you use so that the robot followed the wall with regular oscillations?

   The proportional gain used was 20.

5. How far and how long was the robot able to follow the wall without losing it?

The robot could follow the wall for a seemingly infinite amount of time.

Proportional-Derivative Control

6. What derivative gain did you use so that the robot followed the wall with minimal oscillations and limited hitting?

   The derivative gain used was 1.

7. How far and how long was the robot able to follow the wall without losing it?

   The robot could follow the wall for a seemingly infinite amount of time.

8. How did you modify the code so that the robot could detect and outside corner or doorway?

   The code was modified by having a variable that would keep track of the most recent wall that was sensed. That way if there was an outside corner the robot would know which way to turn to return to the wall. A doorway was detected by the robot sensing if there are walls on both sides of it.

Part 2 – Avoid Obstacle (Layer 0)

9. How did you integrate avoid obstacle into the previous part?

   If an object gets within 2 cm of either the left or right sensor, the runaway() method is triggered.

10. How does your robot handle a stuck situation? Did the robot ever get stuck?

    The robot has not gotten stuck, but if it did runaway() would likely be activated.

Part 3 – Random Wander (Layer 3)

11. Describe how the robot's random wander behavior worked and how you integrated it with wall following and avoid obstacle.

    Random wander worked by turning a random angle, then moving a random number of steps forward. It was integrated with the system by triggering random wander when the robot loses all walls for 4 seconds.

Part 4 – Follow Center (Layer 2)

12. Describe how you add the follow center layer to the subsumption architecture that you're already built?

The follow center layer was added identically to the follow left / right layers. If the criteria (sensing walls on both sides) are met, the architecture will switch to using the follow center behavior until one of the walls is no longer detected.

Part 6 – Go To Goal

13. How did you keep track of the robot's progress around an obstacle and ensure that it was still making progress toward the goal from the current position?

    The progress towards the goal was counted via the encoders. I kept track of the robot's progress by only counting the encoder steps that were made towards the goal. This was done by counting all steps before the first turn, not counting the steps after the first turn, counting all the steps after the second turn, not counting the steps after the third turn, and counting the steps after the fourth turn.

Conclusions

14. How does what you implemented on the robot compare to what you planned to based upon your software design plan?

    Our software design plan has 7 states, ours only has 5. The reason is our Random-Wander state contains the logic for 3 of the states used in wandering. We re-used some of our old code as it made it easier and quicker to implement.

15. When tuning the proportional controller and/or derivative controller, did the robot exhibit any oscillating, damping, overshoot or offset error? If so, how much?

    The robot did experience some overshoot when turning the controller. There would be times where the overshoot was so great that the robot completely lost the wall, then turned to face it.

16. What were the results of the different P and D controller gains? How did you decide which one to use?

    The P gain selected was 20 and the D gain selected was 1. They were chosen by running multiple trials, including trials with an obstacle. Running the trials with the obstacle showed the need for larger proportional gains so that the robot can maneuver corners. The derivative gain helped to dampen the response.

17. How accurate was the robot at maintaining a distance between 4 and 6 inches?

    The robot was very accurate at maintaining a distance between 4 and 6 inches, almost never falling out of that range.

18. Did the robot ever lose the wall?

    The robot did lose the wall before switching the direction it was moving to backwards. But once the robot started moving backwards it never lost the wall.

19. Compare and contrast the performance of the *Wander* and *Avoid* behaviors compared to last week's lab.

    The wander and avoid behaviors are the same methods as used in lab 2. The wander method operated the same as the pervious lab. The avoid behavior performed better in this lab as the obstacles were not as difficult to avoid.

20. What was the general plan to implement the feedback control and subsumption architecture on the robot?

The general idea was to focus on the proportional gain constant and create an error term based on the distances from the lidars.

21. How could you improve the control architecture and/or wall following/follow center behaviors?

    To improve the control architecture, we could try to make it more modular. A lot of logic is similar between various implementations of the wallFollow (wallFollowStates, wallFollowPD, etc.). Our wall following code works well, but the robot doesn't usually drive in a straight line, instead doing a slight weave at the specified distance from the wall. If we include integral or derivative constants to our PID controller it might be smoother.

22. How did you implement the finite state machine to integrate the various behaviors? Did you use any inhibition and suppression to create layers in this behavior?

    We created the 4 different wall states: LEFT_WALL, RIGHT_WALL, NO_WALL, CENTER_WALL, and another state for random wander: RANDOM_WANDER. Random wander is its own category and runs until a wall is detected, where it returns to one of the various wall following states.

23. How did you keep track of the robot's state and as it switched between behaviors?

    I kept track of the of the robot's state by assigning it to an integer variable. The state was updated based on what the robot sensed around it.

24. What did you learn? What did you observe? How could you improve your performance?

    Our sonar sensors are far too noisy to utilize for wall following. We used trial and error to determine the proportional and derivative gains, but having a faster way to get feedback rather than waiting for the long compilation would be very useful. (since we're compiling with RPC). Implementing integral gain and experimenting with more gain constants may help reduce the weaving of the robot too, however we don't have a good way to efficiently test a variety of PID constants.

Appendix

Insert your properly commented and modular code in the appendix of the worksheet.

```
/*
  NOTE:
   THIS IS THE STANDARD FOR HOW TO PROPERLY COMMENT CODE
   Header comment has program, name, author name, date created
   Header comment has brief description of what program does
   Header comment has list of key functions and variables created with decription
   There are sufficient in line and block comments in the body of the program
   Variables and functions have logical, intuitive names
   Functions are used to improve modularity, clarity, and readability
***********************************
  RobotIntro.ino
  Carlotta Berry 11.21.16

  This program will introduce using the stepper motor library to create motion
algorithms for the robot.
```

```
  The motions will be go to angle, go to goal, move in a circle, square, figure
eight and teleoperation (stop, forward, spin, reverse, turn)
  It will also include wireless commmunication for remote control of the robot by
using a game controller or serial monitor.
  The primary functions created are
  moveCircle - given the diameter in inches and direction of clockwise or
counterclockwise, move the robot in a circle with that diameter
  moveFigure8 - given the diameter in inches, use the moveCircle() function with
direction input to create a Figure 8
  forward, reverse - both wheels move with same velocity, same direction
  pivot- one wheel stationary, one wheel moves forward or back
  spin - both wheels move with same velocity opposite direction
  turn - both wheels move with same direction different velocity
  stop -both wheels stationary

  Interrupts
  https://www.arduino.cc/reference/en/language/functions/external-
interrupts/attachinterrupt/
  https://www.arduino.cc/en/Tutorial/CurieTimer1Interrupt
  https://playground.arduino.cc/code/timer1
  https://playground.arduino.cc/Main/TimerPWMCheatsheet
  http://arduinoinfo.mywikis.net/wiki/HOME

  Hardware Connections:
  Arduino pin mappings: https://www.arduino.cc/en/Hacking/PinMapping2560
  A4988 Stepper Motor Driver Pinout: https://www.pololu.com/product/1182

  digital pin 48 - enable PIN on A4988 Stepper Motor Driver StepSTICK
  digital pin 50 - right stepper motor step pin
  digital pin 51 - right stepper motor direction pin
  digital pin 52 - left stepper motor step pin
  digital pin 53 - left stepper motor direction pin
  digital pin 13 - enable LED on microcontroller

  digital pin 5 - red LED in series with 220 ohm resistor
  digital pin 6 - green LED in series with 220 ohm resistor
  digital pin 7 - yellow LED in series with 220 ohm resistor

  digital pin 18 - left encoder pin
  digital pin 19 - right encoder pin

  INSTALL THE LIBRARY
  AccelStepper Library: https://www.airspayce.com/mikem/arduino/AccelStepper/
```

```
  Sketch->Include Library->Manage Libraries...->AccelStepper->Include
  OR
  Sketch->Include Library->Add .ZIP Library...->AccelStepper-1.53.zip
  See PlatformIO documentation for proper way to install libraries in Visual
Studio
*/

//include all necessary libraries
#include <Arduino.h>       //include for PlatformIO Ide
#include <AccelStepper.h>  //include the stepper motor library
#include <MultiStepper.h>  //include multiple stepper motor library
#include <RPC.h>
#include <List.hpp>

// Create lists for moving averages
#define SONAR_ARR_SIZE 6
int* frontLidarArr = new int[6];
int* backLidarArr = new int[6];
int* leftLidarArr = new int[6];
int* rightLidarArr = new int[6];
int* leftSonarArr = new int[SONAR_ARR_SIZE];
int* rightSonarArr = new int[SONAR_ARR_SIZE];

// Bool to determine whether to count encoder ticks
bool countTicksL = true;
bool countTicksR = false;

//state LEDs connections
#define redLED 5            //red LED for displaying states
#define grnLED 6            //green LED for displaying states
#define ylwLED 7            //yellow LED for displaying states
#define enableLED 13        //stepper enabled LED
int leds[3] = { 5, 6, 7 };  //array of LED pin numbers

//define motor pin numbers
#define stepperEnable 48  //stepper enable pin on stepStick
#define rtStepPin 50      //right stepper motor step pin
#define rtDirPin 51       // right stepper motor direction pin
#define ltStepPin 52      //left stepper motor step pin
#define ltDirPin 53       //left stepper motor direction pin

//define the Lidar constants
#define LIDAR_FRONT 0
#define LIDAR_BACK 1
```

```
#define LIDAR_LEFT 2
#define LIDAR_RIGHT 3
#define numOfSens 4

//define the behavior constants
#define NO_BEHAVIOR 0
#define COLLIDE 1

//define the Lidar variables
int16_t ft_lidar = 8;
int16_t bk_lidar = 9;
int16_t lt_lidar = 10;
int16_t rt_lidar = 11;
int16_t lidar_pins[numOfSens] = {8,9,10,11};
int16_t lidarDist[numOfSens] = {0,0,0,0};

//define the Sonar constants
#define VELOCITY_TEMP(temp) ((331.5 + 0.6 * (float)(temp)) * 100 / 1000000.0)  //
The ultrasonic velocity (cm/us) compensated by temperature
#define SONAR_RIGHT 0
#define SONAR_LEFT 1

//define the Sonar variables
int16_t rt_trigechoPin = 3;
int16_t lt_trigechoPin = 4;
int16_t trig_EchoPin[2] = { 3,4 };
int16_t sonarDist[2] = {0,0};

AccelStepper stepperRight(AccelStepper::DRIVER, rtStepPin, rtDirPin);  //create
instance of right stepper motor object (2 driver pins, low to high transition
step pin 52, direction input pin 53 (high means forward)
AccelStepper stepperLeft(AccelStepper::DRIVER, ltStepPin, ltDirPin);   //create
instance of left stepper motor object (2 driver pins, step pin 50, direction
input pin 51)
MultiStepper steppers;                                                 //create
instance to control multiple steppers at the same time

#define stepperEnTrue false  //variable for enabling stepper motor
#define stepperEnFalse true  //variable for disabling stepper motor

int pauseTime = 2500;  //time before robot moves
int stepTime = 500;    //delay time between high and low on step pin
int wait_time = 1000;  //delay for printing data
```

```
#define WANDER_TIME 4000 //time between change of wander wheel speeds in millis
int wanderTimer = 0; //timer to determine when to change wander wheel speeds

//define encoder pins
#define LEFT 0                          //left encoder
#define RIGHT 1                         //right encoder
const int ltEncoder = 18;               //left encoder pin (Mega Interrupt pins 2,3 18,19,20,21)
const int rtEncoder = 19;               //right encoder pin (Mega Interrupt pins 2,3 18,19,20,21)
volatile long encoder[2] = { 0, 0 };  //interrupt variable to hold number of encoder counts (left, right)
int lastSpeed[2] = { 0, 0 };            //variable to hold encoder speed (left, right)
int accumTicks[2] = { 0, 0 };           //variable to hold accumulated ticks since last reset

bool run = false;

struct sensor_data {
  // this can easily be extended to contain sonar data as well
  int lidar_front;
  int lidar_back;
  int lidar_left;
  int lidar_right;
  int sonar_left;
  int sonar_right;
  // this defines some helper functions that allow RPC to send our struct (I found this on a random forum)
  MSGPACK_DEFINE_ARRAY(lidar_front, lidar_back, lidar_left, lidar_right, sonar_left, sonar_right)
} sensors;

// read_lidars is the function used to get lidar data to the M7
struct sensor_data read_sensors() {
  return sensors;
}
// reads a lidar given a pin
int read_lidar(int pin) {
  int16_t t = pulseIn(pin, HIGH);
  int d; //distance to  object
  if (t == 0){
    // pulseIn() did not detect the start of a pulse within 1 second.
    //Serial.println("timeout");
```

```
    d = 100000; //no object detected
  }
  else if (t > 1850)  {
    //Serial.println("timeout");
    d = 100000; //no object detected
  }
  else  {
    // Valid pulse width reading. Convert pulse width in microseconds to distance
in millimeters.
    d = (t - 1000) * 3 / 40;

    // Limit minimum distance to 0.
    if (d < 0) { d = 0; }
  }
  //   Serial.print(d);
  // Serial.print(" cm, ");
  return d;
}

int movingAverage(int arr[], int arrSize) {
  int sum = 0;
  for (int i = 0; i < arrSize; i++) {
    sum += arr[i];
  }
  return sum / arrSize;
}

int* shiftArray(int arr[], int arrSize, int newValue) {
  for (int i = arrSize - 1; i > 0; i--) {
    arr[i] = arr[i - 1];
  }
  arr[0] = newValue;

  return arr;
}

void setupM4() {
  // bind a method to return the lidar data all at once
  RPC.bind("read_sensors", read_sensors);
}
void loopM4() {
  // update the struct with current lidar data

  struct sensor_data data;
```

```
  float lidarFrontCurr = read_lidar(8);
  float lidarBackCurr = read_lidar(9);
  float lidarLeftCurr = read_lidar(10);
  float lidarRightCurr = read_lidar(11);

  frontLidarArr = shiftArray(frontLidarArr, 6, lidarFrontCurr);
  backLidarArr = shiftArray(backLidarArr, 6, lidarBackCurr);
  leftLidarArr = shiftArray(leftLidarArr, 6, lidarLeftCurr);
  rightLidarArr = shiftArray(rightLidarArr, 6, lidarRightCurr);

  data.lidar_front = movingAverage(frontLidarArr, 6);
  data.lidar_back = movingAverage(backLidarArr, 6);
  data.lidar_left = movingAverage(leftLidarArr, 6);
  data.lidar_right = movingAverage(rightLidarArr, 6);

  // float sonarLeftCurr = readSonar(SONAR_LEFT);
  // float sonarRightCurr = readSonar(SONAR_RIGHT);

  // leftSonarArr = shiftArray(leftSonarArr, SONAR_ARR_SIZE, sonarLeftCurr);
  // rightSonarArr = shiftArray(rightSonarArr, SONAR_ARR_SIZE, sonarRightCurr);

  // data.sonar_left = movingAverage(leftSonarArr, SONAR_ARR_SIZE);
  // data.sonar_right = movingAverage(rightSonarArr, SONAR_ARR_SIZE);

  sensors = data;
}

// Helper Functions

//interrupt function to count left encoder tickes
void LwheelSpeed() {
  if (countTicksL) {
    encoder[LEFT]++;  //count the right wheel encoder interrupts
  }
}

//interrupt function to count right encoder ticks
void RwheelSpeed() {
  if (countTicksR) {
    encoder[RIGHT]++;  //count the right wheel encoder interrupts
  }
}
```

```
void allOFF() {
  for (int i = 0; i < 3; i++) {
    digitalWrite(leds[i], LOW);
  }
}

//function to set all stepper motor variables, outputs and LEDs
void init_stepper() {
  pinMode(rtStepPin, OUTPUT);                   //sets pin as output
  pinMode(rtDirPin, OUTPUT);                    //sets pin as output
  pinMode(ltStepPin, OUTPUT);                   //sets pin as output
  pinMode(ltDirPin, OUTPUT);                    //sets pin as output
  pinMode(stepperEnable, OUTPUT);               //sets pin as output
  digitalWrite(stepperEnable, stepperEnFalse);  //turns off the stepper motor
driver
  pinMode(enableLED, OUTPUT);                   //set enable LED as output
  digitalWrite(enableLED, LOW);                 //turn off enable LED
  pinMode(redLED, OUTPUT);                      //set red LED as output
  pinMode(grnLED, OUTPUT);                      //set green LED as output
  pinMode(ylwLED, OUTPUT);                      //set yellow LED as output
  digitalWrite(redLED, HIGH);                   //turn on red LED
  digitalWrite(ylwLED, HIGH);                   //turn on yellow LED
  digitalWrite(grnLED, HIGH);                   //turn on green LED
  delay(pauseTime / 5);                         //wait 0.5 seconds
  digitalWrite(redLED, LOW);                    //turn off red LED
  digitalWrite(ylwLED, LOW);                    //turn off yellow LED
  digitalWrite(grnLED, LOW);                    //turn off green LED

  stepperRight.setMaxSpeed(1000);              //set the maximum permitted speed
limited by processor and clock speed, no greater than 4000 steps/sec on Arduino
  stepperRight.setAcceleration(500);          //set desired acceleration in
steps/s^2
  stepperLeft.setMaxSpeed(1000);               //set the maximum permitted speed
limited by processor and clock speed, no greater than 4000 steps/sec on Arduino
  stepperLeft.setAcceleration(500);           //set desired acceleration in
steps/s^2
  steppers.addStepper(stepperRight);           //add right motor to MultiStepper
  steppers.addStepper(stepperLeft);            //add left motor to MultiStepper
  digitalWrite(stepperEnable, stepperEnTrue);  //turns on the stepper motor
driver
  digitalWrite(enableLED, HIGH);               //turn on enable LED
}

//function prints encoder data to serial monitor
```

```
void print_encoder_data() {
  static unsigned long timer = 0;                             //print manager
timer
  if (millis() - timer > 100) {                               //print encoder data
every 100 ms or so
    lastSpeed[LEFT] = encoder[LEFT];                          //record the latest
left speed value
    lastSpeed[RIGHT] = encoder[RIGHT];                        //record the latest
right speed value
    accumTicks[LEFT] = accumTicks[LEFT] + encoder[LEFT];      //record accumulated
left ticks
    accumTicks[RIGHT] = accumTicks[RIGHT] + encoder[RIGHT];  //record accumulated
right ticks
    Serial.println("Encoder value:");
    Serial.print("\tLeft:\t");
    Serial.print(encoder[LEFT]);
    Serial.print("\tRight:\t");
    Serial.println(encoder[RIGHT]);
    Serial.println("Accumulated Ticks: ");
    Serial.print("\tLeft:\t");
    Serial.print(accumTicks[LEFT]);
    Serial.print("\tRight:\t");
    Serial.println(accumTicks[RIGHT]);
    encoder[LEFT] = 0;    //clear the left encoder data buffer
    encoder[RIGHT] = 0;   //clear the right encoder data buffer
    timer = millis();     //record current time since program started
  }
}



/*function to run both wheels to a position at speed*/
void runAtSpeedToPosition() {
  stepperRight.runSpeedToPosition();
  stepperLeft.runSpeedToPosition();
}

/*function to run both wheels continuously at a speed*/
void runAtSpeed() {
  while (stepperRight.runSpeed() || stepperLeft.runSpeed()) {}
}

/*This function, runToStop(), will run the robot until the target is achieved and
   then stop it
```

```
*/
void runToStop() {
  int runNow = 1;
  int rightStopped = 0;
  int leftStopped = 0;

  while (runNow) {
    if (!stepperRight.run()) {
      rightStopped = 1;
      stepperRight.stop();  //stop right motor
    }
    if (!stepperLeft.run()) {
      leftStopped = 1;
      stepperLeft.stop();  //stop ledt motor
    }
    if (rightStopped && leftStopped) {
      runNow = 0;
    }
  }
}


/*
  INSERT DESCRIPTION HERE, what are the inputs, what does it do, functions used
*/
void spin(int angle, int dir) {
  int steps = angle * 5.585;
  if (dir) {
    stepperLeft.move(steps);    //move one full rotation forward relative to
current position
    stepperRight.move(-steps);  //move one full rotation forward relative to
current position
  } else {
    stepperRight.move(steps);  //move one full rotation forward relative to
current position
    stepperLeft.move(-steps);  //move one full rotation forward relative to
current position
  }
  runToStop();  //run until the robot reaches the target
}

/*
  INSERT DESCRIPTION HERE, what are the inputs, what does it do, functions used
*/
```

```
void pivot(int angle, int dir) {
  int steps = angle * 5.585 * 2;
  if (dir) {
    stepperLeft.move(steps);  //move steps
  } else {
    stepperRight.move(steps);
  }

  runToStop();  //run until the robot reaches the target
}

/*
  INSERT DESCRIPTION HERE, what are the inputs, what does it do, functions used
*/
void turn(int time, int dir) {
  int steps = time * 500;

  if (dir) {
    stepperLeft.setMaxSpeed(500);
    stepperRight.setMaxSpeed(250);
    stepperLeft.move(steps);       //move one full rotation forward relative to
current position
    stepperRight.move(steps / 2);  //move one full rotation forward relative to
current position
  } else {
    stepperRight.setMaxSpeed(500);
    stepperLeft.setMaxSpeed(250);
    stepperRight.move(steps);     //move one full rotation forward relative to
current position
    stepperLeft.move(steps / 2);  //move one full rotation forward relative to
current position
  }

  runToStop();  //run until the robot reaches the target
  stepperRight.setMaxSpeed(1000);
  stepperLeft.setMaxSpeed(1000);
  init_stepper();
}

/*
  INSERT DESCRIPTION HERE, what are the inputs, what does it do, functions used
*/
void forward(int steps) {
  // int steps = distance / 0.034375; // for distance in cm
```

```
  stepperRight.move(steps);  //move steps forward relative to current position
  stepperLeft.move(steps);   //move steps forward relative to current position

  runToStop();               //run until the robot reaches the target
}
/*
  INSERT DESCRIPTION HERE, what are the inputs, what does it do, functions used
*/
void reverse(int distance) {
  int steps = distance / 0.034375;
  stepperRight.move(-steps);  //move one full rotation reverse relative to
current position
  stepperLeft.move(-steps);   //move one full rotation reverse relative to
current position
  runToStop();                //run until the robot reaches the target
}
/*
  INSERT DESCRIPTION HERE, what are the inputs, what does it do, functions used
*/
void stop() {
  stepperRight.setSpeed(0);  //set right motor speed
  stepperLeft.setSpeed(0);   //set left motor speed
}

//this function will read the left or right sensor based upon input value
uint16_t readSonar(uint16_t side) {
  uint16_t distance;
  uint32_t pulseWidthUs;
  int16_t dist, temp, dist_in;

  pinMode(trig_EchoPin[side], OUTPUT);
  digitalWrite(trig_EchoPin[side], LOW);
  digitalWrite(trig_EchoPin[side], HIGH);  //Set the trig pin High
  delayMicroseconds(10);                   //Delay of 10 microseconds
  digitalWrite(trig_EchoPin[side], LOW);   //Set the trig pin Low
  pinMode(trig_EchoPin[side], INPUT);                //Set the pin to input mode
  pulseWidthUs = pulseIn(trig_EchoPin[side], HIGH);  //Detect the high level time
on the echo pin, the output high level time represents the ultrasonic flight time
(unit: us)
  distance = pulseWidthUs * VELOCITY_TEMP(20) / 2.0;  //The distance can be
calculated according to the flight time of ultrasonic wave,/
                                                      //and the ultrasonic sound
speed can be compensated according to the actual ambient temperature
  dist_in = 0.394*distance;    //convert cm to inches
```

```
  // Serial.print(dist_in, DEC);   //print inches
  // Serial.print(" inches ");
  // Serial.print(distance, DEC);  //print cm
  // Serial.println(" cm");
  return distance;
}

/*
goToAngle rotates the robot to a specified angle
*/

void goToAngle(int angle) {
  //A wheel travels 27.5cm per revolution
  //A wheel travels 69.1cm per 360 spin
  //There are 800 steps per wheel revolution (quarter stepping)
  //69.1/27.5*800 = 2010.6 steps per 360 spin

  digitalWrite(grnLED, HIGH);   //turn on green LED

  if (angle == 0) {
    return;
  }

  countTicksL = true;
  countTicksR = true;

  int eCounts = abs(angle / 3.45);
  int speed = 100;

  if (angle < 0) {
    stepperLeft.setSpeed(speed);  //set left motor speed
    stepperRight.setSpeed(-speed);  //set right motor speed
    Serial.println("neg");
  } else {
    stepperLeft.setSpeed(-speed);  //set left motor speed
    stepperRight.setSpeed(speed);  //set right motor speed
    Serial.println("pos");
  }

  while (encoder[RIGHT] - eCounts <= 0 || encoder[LEFT] - eCounts <= 0) {
    stepperRight.runSpeed();
    stepperLeft.runSpeed();

    // Serial.print("Right Encoder: ");
```

```
    // Serial.print(encoder[RIGHT]);
    // Serial.print(" ");
    // Serial.print("Left Encoder: ");
    // Serial.println(encoder[LEFT]);
  }

  encoder[RIGHT] = 0;
  encoder[LEFT] = 0;

  digitalWrite(grnLED, LOW);       //turn off green LED
}

/*
randomWander spins the robot to a random angle then moves it a random amount of
steps forward
*/

void randomWander() {
  digitalWrite(grnLED, HIGH);      //turn on green LED

    stepperRight.setSpeed(-300);  //set right motor speed
    stepperLeft.setSpeed(-300);   //set left motor speed

  if (millis() - wanderTimer > WANDER_TIME) {
    spin(random(30, 180), random(0,2));

    wanderTimer = millis();
  }

  runAtSpeed();

  // int angle = random(20, 180);
  // int dir = random(0,2);

  // spin(angle, dir);

  // int distance = random(2000);

  // forward(distance);

}

/*
collide stops the robot when an object is in front of it
```

```
*/

void collide(void) {
  stepperRight.setSpeed(500);  //set right motor speed
  stepperLeft.setSpeed(500);   //set left motor speed

  sensors = RPC.call("read_sensors").as<struct sensor_data>();

  run = true;

  if (sensors.lidar_front <= 15 || sensors.lidar_back <= 15 || sensors.lidar_left
<= 15 || sensors.lidar_right <= 15) {
    run = false;
    digitalWrite(redLED, HIGH);       //turn on red LED
  }

  if (run) {
    runAtSpeed();
    digitalWrite(redLED, LOW);       //turn off red LED
    // Serial.println("run");
  }
}

/*
runaway avoids all obstacles around the robot
*/

void runaway(void) {
  int maxSpeed = 300;
  int rightSpeed;
  int leftSpeed;
  int x;
  int y;

  sensors = RPC.call("read_sensors").as<struct sensor_data>();

  // Serial.print("left = ");
  // Serial.print(sensors.sonar_left);
  // Serial.print(" right = ");
  // Serial.println(sensors.sonar_right);

  if (abs(sensors.lidar_back) < 30 && abs(sensors.lidar_front) < 30) {
    x = sensors.lidar_front - sensors.lidar_back; // x direction of repulsive
vector
```

```
  } else if (abs(sensors.lidar_back) < 30) {
    x = 30 - sensors.lidar_back; // x direction of repulsive vector
  } else if (abs(sensors.lidar_front) < 30) {
    x = -30 + sensors.lidar_front; // x direction of repulsive vector
  } else {
    x = 0;
  }

  if (abs(sensors.lidar_left) < 30 && abs(sensors.lidar_right) < 30) {
    y = sensors.lidar_left - sensors.lidar_right; // x direction of repulsive
vector
  } else if (abs(sensors.lidar_right) < 30) {
    y = 30 - sensors.lidar_right; // x direction of repulsive vector
  } else if (abs(sensors.lidar_left) < 30) {
    y = -30 + sensors.lidar_left; // x direction of repulsive vector
  } else {
    y = 0;
  }

  int angle = atan2(y,x) * 180 / 3.1415;

  Serial.print("x = ");
  Serial.print(x);
  Serial.print(" y = ");
  Serial.print(y);
  Serial.print(" angle = ");
  Serial.println(angle);

  if (abs(x) > 10 || abs (y) > 10) {
    digitalWrite(ylwLED, HIGH);       //turn on yellow LED
    if (angle > -45 && angle <= 45) {
      rightSpeed = maxSpeed;
      leftSpeed = maxSpeed;
    } else if ((angle > 45 && angle <= 90) || (angle > -135 && angle < -90)) {
      rightSpeed = maxSpeed;
      leftSpeed = -maxSpeed/2;
    } else if ((angle >= -90 && angle <= -45) || (angle > 90 && angle <= 135)) {
      rightSpeed = -maxSpeed/2;
      leftSpeed = maxSpeed;
    } else {
      rightSpeed = -maxSpeed;
      leftSpeed = -maxSpeed;
    }
```

```
  } else if (sensors.lidar_left > 0 && sensors.lidar_left < 30 &&
sensors.lidar_right > 0 && sensors.lidar_right < 30 && abs(x) < 4 ) {
    digitalWrite(ylwLED, HIGH);       //turn on yellow LED
    rightSpeed = maxSpeed;
    leftSpeed = maxSpeed;
  } else if (sensors.lidar_front > 0 && sensors.lidar_front < 30 &&
sensors.lidar_back > 0 && sensors.lidar_back < 30 &&  sensors.lidar_left > 30 &&
sensors.lidar_right > 30) {
    digitalWrite(ylwLED, HIGH);       //turn on yellow LED
    spin(90, 0);
  } else {
    digitalWrite(ylwLED, LOW);       //turn off yellow LED
    rightSpeed = 0;
    leftSpeed = 0;
  }

  // if (abs(x) > 10  || abs (y) > 10) {
  //   if (angle <= 90 && angle >= -90) {
  //     rightSpeed = maxSpeed * abs((angle + 90)) / 180;
  //     leftSpeed = maxSpeed * abs((angle - 90)) / 180;
  //   } else {
  //     rightSpeed = -maxSpeed * abs((angle + 90)) / 180;
  //     leftSpeed = -maxSpeed * abs((angle - 90)) / 180;
  //   }
  // }

  // float mag = 200;
  // if(angle < 0) {
  // mag *= -1;
  // angle += 180;
  // }
  // float left_power = mag * max(-1, 1 - angle/45);
  // float right_power = mag * min(1, 3 - angle/45);

  stepperRight.setSpeed(rightSpeed);  //set right motor speed
  stepperLeft.setSpeed(leftSpeed);   //set left motor speed

  runAtSpeed();
}

/*
follow follows an object that is in front of the robot
*/
```

```
void follow(void) {
  digitalWrite(redLED, HIGH);       //turn on red LED
  digitalWrite(grnLED, HIGH);       //turn on green LED

  int maxSpeed = 300;
  int rightSpeed;
  int leftSpeed;
  int x = 0;
  int y = 0;

  sensors = RPC.call("read_sensors").as<struct sensor_data>();

  // Serial.print("left = ");
  // Serial.print(sensors.sonar_left);
  // Serial.print(" right = ");
  // Serial.println(sensors.sonar_right);

  // Determine x direction of attractive vector
  if (sensors.lidar_back < 30){
    x += -30 + sensors.lidar_back;
  }
  if (sensors.lidar_front < 30){
    x += 30 - sensors.lidar_front;
  }
  if (sensors.sonar_left < 15) {
    x += 15 - sensors.sonar_left;
  }
  if (sensors.sonar_right < 15) {
    x += 15 - sensors.sonar_right;
  }

  // Determine y direction of attractive vector
  if (sensors.lidar_right < 30){
    y += -30 + sensors.lidar_right;
  }
  if (sensors.lidar_left < 30){
    y += 30 - sensors.lidar_left;
  }
  if (sensors.sonar_left < 15) {
    y += 15 - sensors.sonar_left;
  }
  if (sensors.sonar_right < 15) {
    y += -15 + sensors.sonar_right;
  }
```

```
  int angle = atan2(y,x) * 180 / 3.1415;

  Serial.print("x = ");
  Serial.print(x);
  Serial.print(" y = ");
  Serial.print(y);
  Serial.print(" angle = ");
  Serial.println(angle);
  if(abs(y) > 5 || abs(x) > 5) {
    if (angle > -30 && angle < 30 && abs(x) < 25 ) {
      rightSpeed = maxSpeed;
      leftSpeed = maxSpeed;
      Serial.println("Forward");
    } else if (angle > -30 && angle < 30 && abs(x) > 35 ) {
      rightSpeed = -maxSpeed;
      leftSpeed = -maxSpeed;
      Serial.println("Backward");
    } else if (angle >= 30 && angle <= 180) {
      rightSpeed = maxSpeed;
      leftSpeed = -maxSpeed;
      Serial.println("Left");
    } else if (angle <= -30 && angle >= -180) {
      rightSpeed = -maxSpeed;
      leftSpeed = maxSpeed;
      Serial.println("Right");
    } else {
      rightSpeed = 0;
      leftSpeed = 0;
    }
  } else {
    rightSpeed = 0;
    leftSpeed = 0;
  }

  stepperRight.setSpeed(rightSpeed);  //set right motor speed
  stepperLeft.setSpeed(leftSpeed);    //set left motor speed

  runAtSpeed();
}


#define STATE_WANDER 0
#define STATE_COLLIDE 1
```

```
#define STATE_RUNAWAY 2
int state = 0;
void smartWander(void) {
  sensors = RPC.call("read_sensors").as<struct sensor_data>();
  switch (state) {
    case STATE_WANDER:
      digitalWrite(ylwLED, LOW);       //turn off yellow LED
      Serial.println("wander");
      randomWander();

      if (sensors.lidar_front < 15 || sensors.lidar_back < 15 ||
sensors.lidar_right < 15 || sensors.lidar_left < 15) {
        state = STATE_COLLIDE;
      }
      break;
    case STATE_COLLIDE:
      digitalWrite(grnLED, LOW);       //turn off green LED
      Serial.println("collide");
      collide();
      delay(1000);
      state = STATE_RUNAWAY;
      break;
    case STATE_RUNAWAY:
      digitalWrite(redLED, LOW);       //turn off red LED
      Serial.println("runaway");
      runaway();
      if (sensors.lidar_front > 20 && sensors.lidar_back > 20 &&
sensors.lidar_right > 20 && sensors.lidar_left > 20) {
        state = STATE_WANDER;
      }
      break;
    default:
      Serial.println("left state machine");
      break;
  }
}

#define STATE_FOLLOW 3
void smartFollow(void) {
  sensors = RPC.call("read_sensors").as<struct sensor_data>();
  switch (state) {
    case STATE_WANDER:
      digitalWrite(redLED, LOW);       //turn off yellow LED
      digitalWrite(grnLED, LOW);       //turn off yellow LED
```

```
      Serial.println("wander");
      randomWander();
      if (sensors.lidar_front < 15 || sensors.lidar_back < 15 ||
sensors.lidar_right < 15 || sensors.lidar_left < 15) {
        state = STATE_COLLIDE;
      }
      break;
    case STATE_COLLIDE:
      digitalWrite(grnLED, LOW);       //turn off green LED
      Serial.println("collide");
      collide();
      delay(1000);
      state = STATE_FOLLOW;
      break;
    case STATE_FOLLOW:
      digitalWrite(redLED, LOW);       //turn off red LED
      Serial.println("follow");
      follow();
      if (sensors.lidar_front > 20 && sensors.lidar_back > 20 &&
sensors.lidar_right > 20 && sensors.lidar_left > 20 && sensors.sonar_left > 20 &&
sensors.sonar_left > 20) {
        state = STATE_WANDER;
      }
      break;
    default:
      Serial.println("left state machine");
      break;
  }
}

/*
wallFollowBB implements bang bang control in order to follow a wall
*/

#define NO_WALL 0
#define LEFT_WALL 1
#define RIGHT_WALL 2
#define CENTER_WALL 3
#define LOST_WALL 4
#define RANDOM_WANDER 5
#define BACK_WALL 6
void wallFollowBB(void) {
  int maxSpeed = 300;
  int rightSpeed;
```

```
int leftSpeed;
int x = 0;
int y = 0;

sensors = RPC.call("read_sensors").as<struct sensor_data>();

Serial.print("left = ");
Serial.print(sensors.lidar_left);
Serial.print(" right = ");
Serial.println(sensors.lidar_right);

if (sensors.lidar_left < 30 && sensors.lidar_right < 30) {
  state = CENTER_WALL;
} else if (sensors.lidar_left < 30) {
  state = LEFT_WALL;
} else if (sensors.lidar_right < 30) {
  state = RIGHT_WALL;
}

switch(state) {
  case NO_WALL:
    rightSpeed = 0;
    leftSpeed = 0;
    break;
  case LEFT_WALL:
    if (sensors.lidar_left >= 10 && sensors.lidar_left <= 15){
      digitalWrite(redLED, LOW);       //turn off red LED
      digitalWrite(ylwLED, LOW);       //turn off yellow LED
      rightSpeed = maxSpeed;
      leftSpeed = maxSpeed;
    } else if (sensors.lidar_left <= 10) {
      digitalWrite(ylwLED, HIGH);       //turn on yellow LED
      rightSpeed = maxSpeed/1.5;
      leftSpeed = maxSpeed;
    } else {
      digitalWrite(redLED, HIGH);       //turn on red LED
      rightSpeed = maxSpeed;
      leftSpeed = maxSpeed/1.5;
    }
    break;
  case RIGHT_WALL:
    if (sensors.lidar_right >= 10 && sensors.lidar_right <= 15){
      digitalWrite(redLED, LOW);       //turn off red LED
      digitalWrite(ylwLED, LOW);       //turn off yellow LED
```

```
        rightSpeed = maxSpeed;
        leftSpeed = maxSpeed;
      } else if (sensors.lidar_right < 10) {
        digitalWrite(ylwLED, HIGH);       //turn on yellow LED
        rightSpeed = maxSpeed;
        leftSpeed = maxSpeed/1.5;
      } else {
        digitalWrite(redLED, HIGH);       //turn on red LED
        rightSpeed = maxSpeed/1.5;
        leftSpeed = maxSpeed;
      }
      break;
    case CENTER_WALL:
      y = sensors.lidar_left - sensors.lidar_right;

      if (y >= -3 && y <= 3) {
        digitalWrite(redLED, LOW);       //turn off red LED
        digitalWrite(ylwLED, LOW);       //turn off yellow LED
        rightSpeed = maxSpeed;
        leftSpeed = maxSpeed;
      } else if (y > 3) {
        rightSpeed = maxSpeed;
        leftSpeed = maxSpeed/2;
      } else {
        rightSpeed = maxSpeed/2;
        leftSpeed = maxSpeed;
      }
      break;
    case RANDOM_WANDER:
      randomWander();
      break;
  }

  stepperRight.setSpeed(rightSpeed);  //set right motor speed
  stepperLeft.setSpeed(leftSpeed);    //set left motor speed
  runAtSpeed();
}

/*
wallFollowP implements proportional control in order to follow a wall
*/

float prop = 0;
void wallFollowP(void) {
```

```
int maxSpeed = 200;
int frontTurnDist = 15;
int rightSpeed;
int leftSpeed;
int x = 0;
int y = 0;
int error = 0;
float kp = 3;

sensors = RPC.call("read_sensors").as<struct sensor_data>();

// Serial.print("Sonar left = ");
// Serial.print(sensors.sonar_left);
// Serial.print("Sonar right = ");
// Serial.println(sensors.sonar_right);

if (sensors.lidar_left < 30 && sensors.lidar_right < 30) {
  state = CENTER_WALL;
} else if (sensors.lidar_left < 30) {
  state = LEFT_WALL;
} else if (sensors.lidar_right < 30) {
  state = RIGHT_WALL;
}

switch(state) {
  case NO_WALL:
    rightSpeed = 0;
    leftSpeed = 0;
    break;
  case LEFT_WALL:
    if (sensors.lidar_left >= 10 && sensors.lidar_left <= 15){
      digitalWrite(redLED, LOW);        //turn off red LED
      digitalWrite(ylwLED, LOW);        //turn off yellow LED
    } else if (sensors.lidar_left <= 10) {
      digitalWrite(ylwLED, HIGH);       //turn on yellow LED
    } else {
      digitalWrite(redLED, HIGH);       //turn on red LED
    }

    error = min(sensors.lidar_left - 12.5, 12);

    prop = kp * error;
    rightSpeed = maxSpeed + prop;
    leftSpeed = maxSpeed - prop;
```

```
      break;
    case RIGHT_WALL:
      if (sensors.lidar_right >= 10 && sensors.lidar_right <= 15){
        digitalWrite(redLED, LOW);       //turn off red LED
        digitalWrite(ylwLED, LOW);       //turn off yellow LED
      } else if (sensors.lidar_right < 10) {
        digitalWrite(ylwLED, HIGH);       //turn on yellow LED
      } else {
        digitalWrite(redLED, HIGH);       //turn on red LED
      }

      error = min(sensors.lidar_right - 12.5, 12);

      prop = kp * error;
      rightSpeed = maxSpeed - prop;
      leftSpeed = maxSpeed + prop;
      break;
    case CENTER_WALL:
      y = sensors.lidar_left - sensors.lidar_right;

      if (y >= -3 && y <= 3) {
        digitalWrite(redLED, HIGH);       //turn on red LED
        digitalWrite(ylwLED, HIGH);       //turn on yellow LED
        digitalWrite(grnLED, HIGH);       //turn on green LED
      } else {
        digitalWrite(redLED, LOW);       //turn off red LED
        digitalWrite(ylwLED, LOW);       //turn off yellow LED
        digitalWrite(grnLED, LOW);       //turn off green LED
      }

      error = min(y, 12);
      prop = kp * error;
      rightSpeed = maxSpeed + prop;
      leftSpeed = maxSpeed - prop;
      break;
  }

  if (sensors.lidar_front < 15) {
    if (state == LEFT_WALL) {
      collide();
      delay(1000);
      spin(90, 1);
    } else {
      collide();
```

```
      delay(1000);
      spin(90, 0);
    }
  }

  Serial.print("left = ");
  Serial.print(leftSpeed);
  Serial.print(" right = ");
  Serial.println(rightSpeed);

  stepperRight.setSpeed(rightSpeed);  //set right motor speed
  stepperLeft.setSpeed(leftSpeed);   //set left motor speed
  stepperRight.runSpeed();
  stepperLeft.runSpeed();
}

/*
wallFollowPD implements proportional/derivative control in order to follow a wall
*/

float pd = 0;
float lastError = 0;
void wallFollowPD(void) {
  int maxSpeed = -200;
  int frontTurnDist = 10;
  int rightSpeed;
  int leftSpeed;
  float x = 0;
  float y = 0;
  float error = 0;
  float kp = 20;
  float kd = 1;
  float kp_back = 200;

  sensors = RPC.call("read_sensors").as<struct sensor_data>();

  if (sensors.lidar_left < 30 && sensors.lidar_right < 30) {
    state = CENTER_WALL;
  } else if (sensors.lidar_left < 30) {
    state = LEFT_WALL;
  } else if (sensors.lidar_right < 30) {
    state = RIGHT_WALL;
  }
```

```
lightState(state, sensors);

switch(state) {
  case NO_WALL:
    rightSpeed = 0;
    leftSpeed = 0;
    break;
  case LEFT_WALL:
    error = min(sensors.lidar_left - 12.5, 12);

    pd = kp * error + kd * (error - lastError);

    if (sensors.lidar_back <= frontTurnDist) {
      pd -= kp_back * (frontTurnDist - sensors.lidar_back);
    }

    rightSpeed = maxSpeed - pd;
    leftSpeed = maxSpeed + pd;
    break;
  case RIGHT_WALL:
    error = min(sensors.lidar_right - 12.5, 12);

    pd = kp * error + kd * (error - lastError);

    if (sensors.lidar_back <= frontTurnDist) {
      pd -= kp_back * (frontTurnDist - sensors.lidar_back);
    }

    rightSpeed = maxSpeed + pd;
    leftSpeed = maxSpeed - pd;
    break;
  case CENTER_WALL:
    error = sensors.lidar_left - sensors.lidar_right;
    Serial.print("error = ");
    Serial.print(error);

    if (abs(error) <= 3) {
      pd = 0;
    } else {
      pd = kp * error + kd * (error - lastError);
    }

    rightSpeed = maxSpeed - pd;
    leftSpeed = maxSpeed + pd;
```

```
      break;
  }

  // Serial.print("Sonar left = ");
  // Serial.print(sensors.sonar_left);
  // Serial.print("back = ");
  // Serial.print(sensors.lidar_back);

  Serial.print("left = ");
  Serial.print(leftSpeed);
  Serial.print(" right = ");
  Serial.println(rightSpeed);

  stepperRight.setSpeed(rightSpeed);  //set right motor speed
  stepperLeft.setSpeed(leftSpeed);   //set left motor speed
  stepperRight.runSpeed();
  stepperLeft.runSpeed();

  lastError = error;
}

/*
wallFollowStates implements PD control in order to follow a wall, along with
random wander when all walls are lost, and avoid when the robot gets too
close to a wall
*/

bool timerStarted = false;
int wallTimer = 0;
void wallFollowStates (void) {
  int maxSpeed = -200;
  int frontTurnDist = 10;
  int rightSpeed;
  int leftSpeed;
  float x = 0;
  float y = 0;
  float error = 0;
  float kp = 20;
  float kd = 1;
  float kp_back = 200;
  int lostTimer = 0;

  sensors = RPC.call("read_sensors").as<struct sensor_data>();
```

```
if (sensors.lidar_left < 30 && sensors.lidar_right < 30) {
  state = CENTER_WALL;
  wallTimer = millis();
} else if (sensors.lidar_left < 40) {
  state = LEFT_WALL;
  wallTimer = millis();
} else if (sensors.lidar_right < 40) {
  state = RIGHT_WALL;
  wallTimer = millis();
} else {
  if (millis() - 4000 > wallTimer) {
    state = RANDOM_WANDER;
  }
}

lightState(state, sensors);

switch(state) {
  case NO_WALL:
    rightSpeed = 0;
    leftSpeed = 0;
    break;
  case LEFT_WALL:
    error = min(sensors.lidar_left - 12.5, 12);

    pd = kp * error + kd * (error - lastError);

    if (sensors.lidar_back <= frontTurnDist) {
      pd -= kp_back * (frontTurnDist - sensors.lidar_back);
    }

    rightSpeed = maxSpeed - pd;
    leftSpeed = maxSpeed + pd;
    break;
  case RIGHT_WALL:
    error = min(sensors.lidar_right - 12.5, 12);

    pd = kp * error + kd * (error - lastError);

    if (sensors.lidar_back <= frontTurnDist) {
      pd -= kp_back * (frontTurnDist - sensors.lidar_back);
    }

    rightSpeed = maxSpeed + pd;
```

```
      leftSpeed = maxSpeed - pd;
      break;
    case CENTER_WALL:
      error = sensors.lidar_left - sensors.lidar_right;
      Serial.print("error = ");
      Serial.print(error);

      if (abs(error) <= 3) {
        pd = 0;
      } else {
        pd = kp * error + kd * (error - lastError);
      }

      rightSpeed = maxSpeed - pd;
      leftSpeed = maxSpeed + pd;
      break;
    case RANDOM_WANDER:
      randomWander();
      if (sensors.lidar_back < 10) {
        spin(90, 0);
      }
      break;
  }

  // Serial.print("Sonar left = ");
  // Serial.print(sensors.sonar_left);
  // Serial.print("back = ");
  // Serial.print(sensors.lidar_back);

  Serial.print("left = ");
  Serial.print(leftSpeed);
  Serial.print(" right = ");
  Serial.println(rightSpeed);

  stepperRight.setSpeed(rightSpeed);  //set right motor speed
  stepperLeft.setSpeed(leftSpeed);   //set left motor speed
  stepperRight.runSpeed();
  stepperLeft.runSpeed();

  lastError = error;
}

/*
```

```
goToGoalAvoidObs goes to a specific goal location while being able to avoid
objects in its path
*/

#define NO_OBSTACLE 0
#define SIDE_1 1
#define SIDE_2 2
#define SIDE_3 3
#define POST_OBSTACLE 4

int gtgWall = 0;
int gtgState = NO_OBSTACLE;
bool hasTurned = false;
void goToGoalAvoidObs(int x, int y) {

  int angle;

  angle = atan2(y, x)*180/3.1415;

  // Serial.println("Angle: ");
  // Serial.println(angle);

  goToAngle(angle);
  delay(1000);
  digitalWrite(grnLED, LOW);       //turn off green LED

  double distance = sqrt(pow(x,2) + pow(y,2));

  // Serial.println("Dist: ");
  // Serial.println(distance);

  int eCounts = distance / 10.8 * 40;

  Serial.print("eCount: ");
  Serial.println(eCounts);

  int speed = -300;

  int turnDelay = 4000;
  int changeStateDelay = 10000;
  int turnTimer = 0;

  int obsCount = 0;
```

```
  countTicksR = false;
  encoder[LEFT] = 0;
  encoder[RIGHT] = 0;

  while (eCounts - encoder[LEFT] >= 0) {
    sensors = RPC.call("read_sensors").as<struct sensor_data>();

    Serial.print("Ecounts Left: ");
    Serial.println(eCounts - encoder[LEFT]);

    if (sensors.lidar_back < 10 && gtgState == NO_OBSTACLE){
      gtgState = SIDE_1;
      hasTurned = false;
    } else if (sensors.lidar_right > 40 && sensors.lidar_left > 40 && gtgState ==
SIDE_1) {
      gtgState = SIDE_2;
      turnTimer = millis();
      hasTurned = false;
    } else if (sensors.lidar_right > 40 && sensors.lidar_left > 40 && millis() -
turnTimer > changeStateDelay && gtgState == SIDE_2) {
      gtgState = SIDE_3;
      turnTimer = millis();
      hasTurned = false;
    } else if (obsCount*2 <= encoder[RIGHT] && gtgState == SIDE_3) {
      gtgState = POST_OBSTACLE;
      hasTurned = false;
    }

    if(sensors.lidar_right < 40) {
      gtgWall = RIGHT_WALL;
    } else if (sensors.lidar_left < 40) {
      gtgWall = LEFT_WALL;
    }

    if (gtgState == NO_OBSTACLE) {
      Serial.println("State: NO_OBSTACLE");
      countTicksL == true;
    }

    if (gtgState == SIDE_1) {
      Serial.println("State: SIDE_1");
      countTicksL = false;
      if (!hasTurned) {
        if (gtgWall == LEFT_WALL) {
```

```
        spin(90, 0);
      } else {
        spin(90, 1);
      }
      hasTurned = true;

      countTicksR = true;
    }
  }

  if (gtgState == SIDE_2) {
    Serial.println("State: SIDE_2");
    obsCount = encoder[RIGHT];

    Serial.print("ObsCount: ");
    Serial.println(obsCount);
    if (!hasTurned && millis() - turnTimer > turnDelay) {
      countTicksR = false;
      if (gtgWall == LEFT_WALL) {
        spin(90, 1);
      } else {
        spin(90, 0);
      }
      countTicksL = true;
      hasTurned = true;
    }
  }

  if (gtgState == SIDE_3) {
    Serial.println("State: SIDE_3");
    if (!hasTurned && millis() - turnTimer > turnDelay) {
      if (gtgWall == LEFT_WALL) {
        spin(90, 1);
      } else {
        spin(90, 0);
      }
      hasTurned = true;
      countTicksR = true;
      countTicksL = false;
    }
  }

  if (gtgState == POST_OBSTACLE) {
    Serial.println("State: POST_OBSTACLE");
```

```
      if (!hasTurned) {
        if (gtgWall == LEFT_WALL) {
          spin(90, 0);
        } else {
          spin(90, 1);
        }
        hasTurned = true;
        countTicksL = true;
      }
    }

    if (sensors.lidar_left < 2 && sensors.lidar_right < 2) {
    runaway();
    }

    stepperLeft.setSpeed(speed);   //set left motor speed
    stepperRight.setSpeed(speed);  //set right motor speed

    stepperRight.runSpeed();
    stepperLeft.runSpeed();
  }

  encoder[RIGHT] = 0;
  encoder[LEFT] = 0;
}

/*
lightState updates the leds on the robot
*/

void lightState(int lightState, struct sensor_data sensors) {

  switch (lightState) {
    case NO_WALL:
      digitalWrite(redLED, LOW);       //turn off red LED
      digitalWrite(ylwLED, LOW);       //turn off yellow LED
      digitalWrite(grnLED, LOW);       //turn off green LED
      break;
    case LEFT_WALL:
      if (sensors.lidar_left >= 10 && sensors.lidar_left <= 15){
        digitalWrite(grnLED, HIGH);       //turn on green LED
        digitalWrite(redLED, LOW);       //turn off red LED
        digitalWrite(ylwLED, HIGH);       //turn on yellow LED
      } else if (sensors.lidar_left <= 10) {
```

```
        digitalWrite(ylwLED, HIGH);      //turn on yellow LED
        digitalWrite(grnLED, LOW);       //turn off green LED
        digitalWrite(redLED, LOW);       //turn off red LED
      } else {
        digitalWrite(redLED, HIGH);      //turn on red LED
        digitalWrite(ylwLED, LOW);       //turn off yellow LED
        digitalWrite(grnLED, LOW);       //turn off green LED
      }
      break;
    case RIGHT_WALL:
      if (sensors.lidar_right >= 10 && sensors.lidar_right <= 15){
        digitalWrite(redLED, HIGH);       //turn on red LED
        digitalWrite(ylwLED, HIGH);       //turn on yellow LED
        digitalWrite(grnLED, LOW);       //turn off green LED
      } else if (sensors.lidar_right < 10) {
        digitalWrite(ylwLED, HIGH);       //turn on yellow LED
        digitalWrite(grnLED, LOW);       //turn off green LED
        digitalWrite(redLED, LOW);       //turn off red LED
      } else {
        digitalWrite(redLED, HIGH);       //turn on red LED
        digitalWrite(ylwLED, LOW);       //turn off yellow LED
        digitalWrite(grnLED, LOW);       //turn off green LED
      }
      break;
    case CENTER_WALL:
      if (sensors.lidar_left - sensors.lidar_right >= -3 && sensors.lidar_left -
sensors.lidar_right <= 3) {
        digitalWrite(redLED, HIGH);       //turn on red LED
        digitalWrite(ylwLED, HIGH);       //turn on yellow LED
        digitalWrite(grnLED, HIGH);       //turn on green LED
      } else {
        digitalWrite(redLED, LOW);       //turn off red LED
        digitalWrite(ylwLED, LOW);       //turn off yellow LED
        digitalWrite(grnLED, LOW);       //turn off green LED
      }
      break;
      case RANDOM_WANDER:
        digitalWrite(redLED, LOW);       //turn off red LED
        digitalWrite(ylwLED, LOW);       //turn off yellow LED
        digitalWrite(grnLED, HIGH);       //turn off green LED
        break;
  }

  if (sensors.lidar_back <= 18) {
```

```
    digitalWrite(redLED, HIGH);       //turn on red LED
    digitalWrite(ylwLED, LOW);       //turn off yellow LED
    digitalWrite(grnLED, HIGH);       //turn on green LED
  }

}

void setup() {
  RPC.begin();
  if(HAL_GetCurrentCPUID() == CM7_CPUID) {
    // if on M7 CPU, run M7 setup & loop
    setupM7();
    while(1) loopM7();
  } else {
    // if on M4 CPU, run M7 setup & loop
    setupM4();
    while(1) loopM4();
  }
}

// loop() is never called as setup() never returns
void loop() {}

//// MAIN
void setupM7() {
  int baudrate = 9600;  //serial monitor baud rate'
  init_stepper();       //set up stepper motor

  attachInterrupt(digitalPinToInterrupt(ltEncoder), LwheelSpeed, CHANGE);  //init
the interrupt mode for the left encoder
  attachInterrupt(digitalPinToInterrupt(rtEncoder), RwheelSpeed, CHANGE);  //init
the interrupt mode for the right encoder

  for (int i = 0; i<numOfSens;i++){
    pinMode(lidar_pins[i],INPUT);
  }

  Serial.begin(baudrate);  //start serial monitor communication

  delay(1000);
  Serial.println("Robot starting...Put ON TEST STAND");
  delay(pauseTime);  //always  wait 2.5 seconds before the robot moves
}
```

```
void loopM7() {
  //Uncomment to read Encoder Data (uncomment to read on serial monitor)
  // print_encoder_data();   //prints encoder data

  goToGoalAvoidObs(77, 0);

  delay(5000);

  //delay(wait_time);               //wait to move robot or read data
}
```