Lab 04 Behavior-Based and Hybrid Control Worksheet

Member Names: Jackson Seida, Justin Dewitt

Robot Name: Murphy

****

Part I – Photoresistor test

Complete the following photoresistor data tables.

Table 1: Environment Data

Conditions	Left Photoresistor (V)	Right Photoresistor (V)
Ambient light on the table	960	960
Ambient light under the table	730	716
Sensor covered	159	95
In front of a flashlight or cell phone light	1020	1021

Table 2: Distance and Angle of Incidence Data

		Left Photoresistor (V)			Right Photoresistor (V)		
Environment	Distance (in)	Angle of Incidence -45°	Angle of Incidence 0°	Angle of Incidence 45°	Angle of Incidence -45°	Angle of Incidence 0°	Angle of Incidence 45°
On the table	6	950	962	980	968	968	960
On the table	12	954	960	964	964	964	960
On the table	18	952	966	954	960	962	960
On the table	24	952	964	953	954	962	952
On the table	30	952	960	952	953	960	953
Under the table	6	856	948	960	954	888	838
Under the table	12	848	964	854	928	940	840
Under the table	18	832	932	838	845	910	838
Under the table	24	836	912	825	832	910	832
Under the table	30	816	884	820	816	888	822

1. How reliable was the photoresistor at detecting the light in different environments, various distances, and angles of incidence (head on, slightly left, slight right)?

The light was not very effective at detecting the light in bright environments, and when the light source was far away. The angle of incidence also affected the sensor readings, with the angle predictably changing the sensor readings to be greater on the side which the light source was on.

2. How significant was the difference in photoresistor voltages for the left and right sides? How did you use this difference to extract directional information to move the robot toward the beacon?

   The difference between the photoresistor voltages depends on the proximity to the light source, and the angle of the light source. This difference was used to extract directional information by having the voltage difference determine the turn direction. Whatever side the voltage from the sensor is lower, should turn faster to move towards the light.

3. How reliable was the photoresistor at detecting the light at various angles and distances? Compare and contrast sensor data.

   The photoresistor was very reliable at detecting light at various angles and distances, especially when in a dark environment. This can be seen by the large variance of voltage values when data was taken underneath the table.

4. How significant was the difference in sensor data based upon distance from the source? How did you use this difference to extract distance information to move the robot toward the beacon?

   There was about a 100 difference in the analog read values between 6in to 30in away. This was used to move the robot towards the beacon by having the robot move faster the farther away the light source was.

Part II - Reactive Control

Excitatory Behavior & Cross Excitatory Behavior

How does the robot behave when (a) the light source is directly in front of the robot, (b) the light source is to one side of the robot? Is there anything about the robot's behavior that surprises you? ***Answer this question in the lab worksheet.***

a) If it's Fear, the robot goes faster, but turns away from the light source. If it's love, the robot goes faster towards the light.

b) If it's Fear the motor closest to the light increases in speed to quickly turn away from the light. If it's love, the motor opposite to the light increases in speed to quickly turn towards the light.

Inhibitory Behavior & Cross Inhibitory Behavior

How does the robot behave when (a) the light source is directly in front of the robot, (b) the light source is to one side of the robot? Is there anything about the robot's behavior that surprises you? ***Answer this question in the lab worksheet.***

a) ***If it's Explorer, the robot slows the motor furthest from the light source and the robot turns away from the light. If it's Aggressive, the robot slows down the motor closest to the light source so the robot turns and slowly approaches the light.***
b) ***If it's Explorer, the robot will turn directly away from the light source by slowing the opposite motor. If it's Aggressive, it will turn towards the source by slowing the closest motor.***

Light Sensing Behaviors

Match the four light sensing behaviors with inhibitory, excitatory, cross inhibitory, cross excitatory.

Fear	Excitatory
Aggression	Inhibitory

Love	Cross Excitatory
Explorer	Cross Inhibitory

Photoresistor Mounting Position

How did you decide on the position of the photoresistors? Were there certain lighting conditions that were more difficult or easier for the robot to sense?

Since we drive the robot backwards, we put them on the back of the robot. The robot performed much better in a dark environment than a light one.

Part III - Obstacle Avoidance

Describe how you implemented obstacle avoidance with light tracking.

Obstacle avoidance with light tracking was implemented by utilizing the love behavior to track the light, and having the robot entire avoid behavior when the lidar sensors detect and object too close to the robot.

How did you integrate the light sensors into the obstacle avoidance behavior?

We used subsumption architecture and used the light sensing excitatory/inhibitory behaviors as a final layer on top of the wall following / state machine layer. This means both behaviors are always active simultaneously.

Part IV – Homing or Docking

Nothing to report here.

Part V – Docking the Robot and Return to the Wall

1. What does the hybrid control architecture for your design look like? What was on the planning layer? Middle layer? Reactive layer?

The planning layer contained flags keeping track of what step it was in the docking processes. The Reactive Layer contained the "Love" behavior, where it drove towards and stopped at the

light, and the middle layer was a state machine that moved between different wall following and movement states.

2. What was your general strategy for planning the path back to the wall from the beacon?

Turn around and go until we reach the wall, then turn to be parallel with the wall and continue wall following.

3. How did the architecture respond to differences in robot start position or beacon location?

Assuming the beacon is close enough to the robot to detect it, it accounts for regular positional changes. It also keeps track of which wall it was initially following, so it can be run starting from either direction.

4. How did the robot's hybrid controller respond to dynamic changes in the environment (i.e., other robots and people) and compare this to purely deliberative control?

Deliberative control cannot react to changes in the environment. Our hybrid controller is constantly polling the sensors, so that despite not having a world model, it can still complete tasks with

5. Were there any challenges in implementing the homing routine?

Finding the right sensor threshold for the ambient light in the room was the most difficult part. Each time we changed the value we'd have to wait multiple minutes for compilation.

6. What could you do to improve the robot homing?

Turn off the lights and increase sensor sensitivity. This would allow the robot to home at further destinations and identify them earlier.

7. How did docking the robot modify the control architecture or algorithm?

We had to create a planning state which controlled returning to the wall after it homed as well as preventing it from getting distracted by the homing block when it drove past it later. Aside from the inhibitory layer, the middle layer also had extra states for docking and homing.

Conclusions

Respond to the following questions.

1. How reliable was the photoresistor at detecting the light in different environments, various distances, and angles of incidence (head on, slightly left, slight right)?

   The photoresistor was not reliable at detecting light in bright environments but was reliable in dark environments. The photoresistor could effective detect different distance that the light was from the sensor, as well as detecting angle of incidences.

2. How significant was the difference in photoresistor voltages for the left and right sides? How did you use this difference to extract directional information to move the robot toward the beacon?

   The difference was significant depending on how far it was turned away from the light. Using the basic love behavior, the robot reactively turned in proportion to the difference between the left and right sensors.

3. How did you integrate the light sensors into the obstacle avoidance behavior?

   The light sensors were utilized as beacon tracking for obstacle avoidance behavior. These sensors are what drove the robot to the light source. A state machine was utilized to determine if the robot would light follow or avoid.

4. How reliable was the photoresistor at detecting the light at various angles and distances? Compare and contrast sensor data.

   The photoresistor was very reliable at detecting light at various angles and distances, especially when in a dark environment. This can be seen by the large variance of voltage values when data was taken underneath the table.

5. How significant was the difference in sensor data based upon distance from the source? How did you use this difference to extract distance information to move the robot toward the beacon?

The light values quickly trailed off with distance (which makes sense due to the inverse square law), which meant our beacon had to be close to the robot for it to be seen, especially with the ambient light in the room.

6. What does the hybrid control architecture for your design look like? What was on the planning layer? Middle layer? Reactive layer?

Planning layer -> flags for making sure that once the robot homes, it docks, and stays on the wall after.

Middle layer -> contains the main state machine of what “task” it is currently on. (Wall following, homing, docking, etc.)

Reactive layer -> contains the basic light seeking behavior “love”. It’s turned on/off by the middle and planning layer.

7. What was your general strategy for planning the path back to the wall from the beacon?

Once we home, we turn 180 degrees and drive forward until the wall was sensed.

8. How did the architecture respond to differences in robot start position or beacon location?

The architecture was able to respond to many different start positions and beacon positions because the lidar and photoresistor sensors were utilized to locate the wall and the beacon.

9. How did the robot’s hybrid controller respond to dynamic changes in the environment (i.e., other robots and people) and compare this to purely deliberative control?

The robot was able to detect and avoid other robots and people. If the robot were purely deliberative, then the robot would not be able to react to a changing environment.

10. Were there any challenges in implementing the homing routine?

The biggest challenge of implementing the homing routine was detecting when to change states between, wall following, moving to the beacon, and moving back to the wall.

11. What could you do to improve the robot homing?

The homing could be improved by implementing a more profound method of tracking the path the robot takes to the beacon.

12. How did docking the robot modify the control architecture or algorithm?

Docking the robot changed the control architecture by making a state machine necessary to track what behavior the robot was doing.

13. What did you learn? What did you observe? What could you improve?

I learned that light sensors require a dark environment to operate effectively and each trial there are different light levels that could effect operation. I observed that I needed to set a proper light level that would trigger the love behavior when doing the homing and docking behaviors. This could be improved by, instead of setting a light level, running a light calibration before each trial.

Appendix

Attach properly commented, modular, cleaned up code here.

```
/*
  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;
  int photoresistor_left;
  int photoresistor_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, photoresistor_left, photoresistor_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);
  data.photoresistor_left = analogRead(A0);
  data.photoresistor_right = analogRead(A1);

  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();
}


/*
smartWander randomly wanders the robot while avoiding obstacles
*/
#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;
  }
}

/*
smartFollow follows an object that is in front of the robot
*/
#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;
bool loved = false;
void wallFollowPD(void) {
  int maxSpeed = -300;
  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 (!loved) {
    if (sensors.lidar_left < 15 && sensors.lidar_right < 15) {
      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;
      }
    }

    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
  }

}

/*
fear uses the photoresistors to turn the robot away from light when sensed
*/
void fear(void) {
  digitalWrite(redLED, HIGH);       //turn on red LED
  digitalWrite(ylwLED, HIGH);       //turn on yellow LED
  digitalWrite(grnLED, HIGH);       //turn on green LED
  sensors = RPC.call("read_sensors").as<struct sensor_data>();

  int rightSpeed = 0;
  int leftSpeed = 0;

  if (sensors.photoresistor_right > 950) {
    rightSpeed = -4* (sensors.photoresistor_right - 950);
  }
  if (sensors.photoresistor_left > 950) {
    leftSpeed = -4 * (sensors.photoresistor_left - 950);
  }


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

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

/*
aggression uses the photoresistors to turn the robot towards the light
*/
void aggression(void) {
  digitalWrite(redLED, HIGH);       //turn on red LED
  digitalWrite(ylwLED, HIGH);       //turn off yellow LED
  digitalWrite(grnLED, LOW);       //turn on green LED

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

  int rightSpeed = -200;
  int leftSpeed = -200;

  if (sensors.photoresistor_right > 950) {
    rightSpeed += 4 * (sensors.photoresistor_right - 950);
  }
  if (sensors.photoresistor_left > 950) {
    leftSpeed += 4 * (sensors.photoresistor_left - 950);
  }


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

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

/*
love uses the photoresistors to turn the robot towards the light when sensed
*/
void love(void) {
  digitalWrite(redLED, HIGH);       //turn on red LED
  digitalWrite(ylwLED, LOW);       //turn off yellow LED
  digitalWrite(grnLED, HIGH);       //turn on green LED

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

  int rightSpeed = 0;
  int leftSpeed = 0;

  // Serial.print("Left Sensor: ");
  // Serial.print(sensors.photoresistor_left);
  // Serial.print("   ");
  // Serial.print("Right Sensor: ");
  // Serial.println(sensors.photoresistor_right);

  if (sensors.photoresistor_left > 800) {
    rightSpeed = -2* (sensors.photoresistor_left - 800);
  }
  if (sensors.photoresistor_right > 800) {
    leftSpeed = -2 * (sensors.photoresistor_right - 800);
  }


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

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

/*
explorer uses the photoresistors to turn the robot away from light
```

```
*/
void explorer(void) {
  digitalWrite(redLED, HIGH);       //turn on red LED
  digitalWrite(ylwLED, LOW);       //turn off yellow LED
  digitalWrite(grnLED, HIGH);       //turn on green LED

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

  int rightSpeed = -200;
  int leftSpeed = -200;

  if (sensors.photoresistor_left > 950) {
    rightSpeed += 4 * (sensors.photoresistor_left - 950);
  }
  if (sensors.photoresistor_right > 950) {
    leftSpeed += 4 * (sensors.photoresistor_right - 950);
  }


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

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

/*
loveAvoid uses the love behavior, but has obstacle avoidance when the robot gets
too close
to obstacles
*/
void loveAvoid(void) {
  sensors = RPC.call("read_sensors").as<struct sensor_data>();

  if (sensors.lidar_back < 10 || sensors.lidar_right < 15 || sensors.lidar_left <
15) {
    runaway();
  } else {
    love();
  }
}
```

```
/*
homing utlizes a state machine to have the robot wall follow until a light is
sensed,
then move towards the light, turn 180 degrees, go back to the wall, and wall
follow
*/
#define STATE_WALLFOLLOW 0
#define STATE_LOVE 1
#define STATE_AFTERLOVE 2
#define LOVE_LIGHT 900
int homingState = STATE_WALLFOLLOW;
void homing(void) {
  sensors = RPC.call("read_sensors").as<struct sensor_data>();

  Serial.print("Left Sensor: ");
  Serial.print(sensors.photoresistor_left);
  Serial.print("   ");
  Serial.print("Right Sensor: ");
  Serial.print(sensors.photoresistor_right);

  if (homingState == STATE_WALLFOLLOW && !loved && (sensors.photoresistor_right >
LOVE_LIGHT || sensors.photoresistor_left > LOVE_LIGHT)) {
    homingState = STATE_LOVE;
    loved = true;
  } else if (homingState == STATE_LOVE && sensors.lidar_back < 10) {
    homingState = STATE_AFTERLOVE;
    delay(3000);
    spin(180, 0);
  } else if (homingState == STATE_AFTERLOVE && sensors.lidar_back < 10) {
    if (state == RIGHT_WALL) {
      spin(135, 1);
    } else if (state == LEFT_WALL)  {
      spin(135, 0);
    }
    homingState = STATE_WALLFOLLOW;
  }

  switch (homingState) {
    case STATE_WALLFOLLOW:
      wallFollowPD();
      Serial.println("  State: Wall Follow");
      break;
    case STATE_LOVE:
      love();
      Serial.println("  State: Love");
      break;
    case STATE_AFTERLOVE:
      stepperLeft.setSpeed(-300);   //set left motor speed
      stepperRight.setSpeed(-300);  //set right motor speed

      stepperRight.runSpeed();
      stepperLeft.runSpeed();
      Serial.println("  State: After Love");
      break;
  }
}

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

  // homing();
  homing();

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