# 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.

1. 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.

1. 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.

1. 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.***

1. ***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.***
2. ***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.

1. 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.

1. 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.

1. 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

1. 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.

1. 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.

1. 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.

1. 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.

1. 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.

1. 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.

1. 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.

1. 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.

1. 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.

1. 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.

1. 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.

1. 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.

1. 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.

1. 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.

1. 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

}