Lab 03

Wall Following: Feedback PD Control Worksheet

Robot Name ___Arkin_____

Team Member Name ___Ben Williams_____

Team Member name ___Donald Woodruff_____

Purpose

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

To use PD control and sensors to create a wall following behavior, detecting walls and maintaining a specified distance from the walls. Also, to implement a go-to-goal ability that will avoid obstacles in the robot's path.

Part 1 – Follow Wall (Layer 1)

Bang-Bang Control

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

We drove our robot forwards, which would in theory allow for sonars to be used in parallel with the lidar sensors. A con of this decision was that when turning, the back side of the robot can sometimes collide with objects.

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

The robot was able to follow the wall for however long the wall was, however it didn't maintain a distance between 4 and 6 inches.

Proportional Control

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

A factor of 3 was multiplied by the difference between the desired distance from the wall and the current distance of the wall (in cm) and used to alter the robot's angle by that much. However, the robot did not stay in the desired band of 4-6 inches.

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

The robot was able to follow the wall for the full length of the room without losing it.

Proportional-Derivative Control

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

We used a derivative gain of 0.2 to change the angle of the robot proportional with the change in distance with respect to the wall.

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

The robot was able to follow the wall for however long the wall was, and it did a better job of maintaining a distance between 4 and 6 inches than previously.

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

We used a variable to track whether the wall was on the left or right side of the robot, and then when it lost track of the wall, it would turn according to its last known location of the wall.

Part 2 – Avoid Obstacle (Layer 0)

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

The robot used a repelling vector to push it away from obstacles, while trying to maintain a specific distance from the wall.

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

We give the robot a small perturbation to hopefully remove it from the stuck position.

Part 3 – Random Wander (Layer 3)

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

If the robot did not see any obstacles or a wall to follow, the robot immediately went into random wander mode.

Part 4 – Follow Center (Layer 2)

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

The robot will follow a wall, but if it gets to a point that it can no longer go forward, the robt will backup to leave the enclosed space.

Part 6 – Go To Goal

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

The robot, all else being equal, will attempt to follow a straight line to it's target. If it detects an obstacle, the robot will track its X and Y position, and go around the obstacle. Once it reaches a point where the obstacle is no longer in the way, the robot will follow the path back to the obstacle (simple and following algorithm)

Conclusions

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

Significant portions of the code had to be hand-tuned and edge cases had to be considered. For instance, for the Follow Center, it was originally assumed the robot would turn around and go back, but it was found to be significantly more efficient to just have the robot travel backwards. Another example is the follow wall. We expected to be able to just adjust the robot's heading haphazardly, but it was found we needed to directly track the heading in order to adjust the angle appropriately.

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

Oscillations and damping were constant, since the lidars were especially finicky, and don't constantly update their readings. This means that they had to be hand tuned and accommodate potentially large amounts of error (~5 cm was not found to be unlikely)

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

The P controller was very good at getting the robot in the target deadband, but the D component was required to stop the robot from oscillating about the set point.

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

The robot did an okay job at maintaining the distance, however due to the variations in lidar sensor readings, it did not maintain those distances perfectly.

18. Did the robot ever lose the wall?

Yes, likely due to the sensors giving poor readings.

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

Wander was largely the same, since the only difference was instead of detecting a wall and then moving a prescribed distance away, the robot would instead track the wall.
The avoid behavior was effectively discarded, since now the robot is following the wall instead of trying to move away from it.

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

The plan was to have the robot estimate the X & Y position to better facilitate the Go To Goal command, while the other commands could integrate just the follow function, and the avoid function could be added on to set a cumulative vector for the robot.

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

Right now the architecture does not keep track of past information very well, only current positioning. This means the robot will never 'learn,' and if it gets stuck in a loop, the robot will not attempt to find another viable solution. By incorporating some further planning, the robot might be able to find a better solution.

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

For the 'Go To Goal' behavior, a state machine was used to keep track of whether the robot was actively going around the object, or heading towards the goal point.

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

The states were largely self contained, but a global variable was used to store the state so different behaviors could have theoretically interacted with the same state.

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

As mentioned previously, the current 'Go To Goal' behavior does not keep track of past information. By storing more information, the robot can better implement some kind of planning algorithm.

Appendix

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

```
/*
**********************************
 RobotIntro.ino
 Donald & Ben 2/7/25

 This program has basic motion algorithms for the robot.
 This includes goToAngle, goForward, goToGoal, circle, square, figureEight, pivot, spin, turn, and stop.

 The primary functions created are:
 moveCircle - given the diameter in cm, direction of rotation, and arc length, the robot will move along
in an arc with that given diameter, in the specified direction
 moveFigure8 - given the diameter in cm, use the moveCircle() function with an arc length of 360 to
create two back to back circles.
 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://docs.arduino.cc/tutorials/giga-r1-wifi/cheat-sheet#pins
  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

  Wheel Diameter = 8.5 cm, 26.7035375555 cm circumference
  Wheel Distance = 21.5 cm
  Stepper motor = 0.45 degree steps
*/

//includew 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 "MedianFilterLib2.h"

//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
int frontFilter;
```

```
int backFilter;
int leftFilter;
int rightFilter;

//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

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
#define max_speed 1500 //maximum stepper motor speed
#define max_accel 10000 //maximum motor acceleration

int pauseTime = 2500;   //time before robot moves
int stepTime = 500;     //delay time between high and low on step pin
int wait_time = 100;   //delay for printing data
double angleConversion = 6;
double distanceConversion = 30;
int closeIRDistance = 15;
int minIRDistance = 2;
int changeEveryOther = 1; // boolean to changeEveryOther every other instance

//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

// define Sensor pins
#define frontLdr 8
#define backLdr 9
#define leftLdr 10
#define rightLdr 11
#define leftSnr 4
```

```
#define rightSnr 3
int windowSize = 4;
MedianFilter2<int> rightSnrFilt(windowSize);
MedianFilter2<int> leftSnrFilt(windowSize);
MedianFilter2<int> frontLdrFilt(windowSize);
MedianFilter2<int> backLdrFilt(windowSize);
MedianFilter2<int> rightLdrFilt(windowSize);
MedianFilter2<int> leftLdrFilt(windowSize);

// Global Variables
String state = "forward";
int dangerDistance = 10;
int offset = 3;
int avoidDistance;
bool frontWall;
int wallDetected = 0;
int leftSpd = 0;
int rightSpd = 0;
int frontDistance = 0;
int nonFrontDistance = 1;
String direction = "north";

// Structures
// a struct to hold lidar data
struct lidar {
  // this can easily be extended to contain sonar data as well
  int front;
  int back;
  int left;
  int right;
  // this defines some helper functions that allow RPC to send our struct (I found this on a random
forum)
  MSGPACK_DEFINE_ARRAY(front, back, left,
right);  //https://stackoverflow.com/questions/37322145/msgpack-to-pack-structures
https://www.appsloveworld.com/cplus/100/391/msgpack-to-pack-structures
} dist;

// a struct to hold sonar data
struct sonar {
  // this can easily be extended to contain sonar data as well
  int left;
  int right;
  // this defines some helper functions that allow RPC to send our struct (I found this on a random
forum)
  MSGPACK_DEFINE_ARRAY(left, right);  //https://stackoverflow.com/questions/37322145/msgpack-to-
pack-structures https://www.appsloveworld.com/cplus/100/391/msgpack-to-pack-structures
} dist2;
```

```
struct lidar read_lidars() {
  return dist;
}

// read_lidars is the function used to get lidar data to the M7
struct sonar read_sonars() {
  return dist2;
}

int frontFilterFunction() {
  struct lidar data = RPC.call("read_lidars").as<struct lidar>();

  if(frontFilter != data.front) {
    frontLdrFilt.AddValue(data.front);
    frontFilter = data.front;
    Serial.println(data.front);
  }

  return frontLdrFilt.GetFiltered();
}

int leftFilterFunction() {
  struct lidar data = RPC.call("read_lidars").as<struct lidar>();

  if(frontFilter != data.left) {
    leftLdrFilt.AddValue(data.left);
    leftFilter = data.left;
    Serial.println(data.left);
  }

  return leftLdrFilt.GetFiltered();
}

int rightFilterFunction() {
  struct lidar data = RPC.call("read_lidars").as<struct lidar>();

  if(rightFilter != data.right) {
    rightLdrFilt.AddValue(data.right);
    rightFilter = data.right;
    Serial.println(data.right);
  }

  return rightLdrFilt.GetFiltered();
}

int backFilterFunction() {
```

```
  struct lidar data = RPC.call("read_lidars").as<struct lidar>();

  if(backFilter != data.back) {
    backLdrFilt.AddValue(data.back);
    backFilter = data.back;
    Serial.println(data.back);
  }

  return backLdrFilt.GetFiltered();
}

void frontFilterClear() {

}

// helper functions
// reads a lidar given a pin
int read_lidar(int pin) {
  int d;
  int16_t t = pulseIn(pin, HIGH);
  d = (t - 1000) * 3 / 40;
  if (t == 0 || t > 1850 || d < 0) { d = 0; }
  return d;
}

// reads a sonar given a pin
int read_sonar(int pin) {
  float velocity ((331.5 + 0.6 * (float)(20)) * 100 / 1000000.0);
  uint16_t distance, pulseWidthUs;

  pinMode(pin, OUTPUT);
  digitalWrite(pin, LOW);
  digitalWrite(pin, HIGH);          //Set the trig pin High
  delayMicroseconds(10);            //Delay of 10 microseconds
  digitalWrite(pin, LOW);           //Set the trig pin Low
  pinMode(pin, INPUT);              //Set the pin to input mode
  pulseWidthUs = pulseIn(pin, 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 / 2.0;
  if (distance < 0 || distance > 50) { distance = 0; }
  return distance;
}

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

```
}

//interrupt function to count right encoder ticks
void RwheelSpeed()
{
  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(max_speed);//set the maximum permitted speed limited by processor and
clock speed, no greater than 4000 steps/sec on Arduino
  stepperRight.setAcceleration(max_accel);//set desired acceleration in steps/s^2
  stepperLeft.setMaxSpeed(max_speed);//set the maximum permitted speed limited by processor and
clock speed, no greater than 4000 steps/sec on Arduino
  stepperLeft.setAcceleration(max_accel);//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 Lidar Data
void print_lidar() {
  // read lidar data from struct
  struct lidar data = RPC.call("read_lidars").as<struct lidar>();

  // print lidar data
  Serial.println("lidar (front, back, left, right): ");
  Serial.print(data.front);
  Serial.print(", ");
  Serial.print(data.back);
  Serial.print(", ");
  Serial.print(data.left);
  Serial.print(", ");
  Serial.print(data.right);
  Serial.println();

}

//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 ( void ) {
  while (stepperRight.runSpeed() || stepperLeft.runSpeed()) {
  }
}

/*
  goToAngle takes in an angle in degrees, and moves the robot that many degrees, moving both motors at equal and opposite speeds so as to stay in position
  It uses an arbitrary speed of 500, and the function's stepperLeft/stepperRight to control each wheel independently
  If the angle is negative, the robot will move in the opposite direction.
*/
void goToAngle(int angle) {
  int leftPos = -angle*angleConversion;//right motor absolute position
  int rightPos = (angle*angleConversion);//left motor absolute position
  int leftSpd = -500;//right motor speed
  int rightSpd = 500; //left motor speed

  if(angle < 0) {
    rightSpd = -rightSpd;
    leftSpd = -leftSpd;
  }

  stepperLeft.move(leftPos);//move left wheel to relative position
  stepperRight.move(rightPos);//move right wheel to relative position

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

  steppers.runSpeedToPosition(); // Blocks until all are in position
}

/*
  This function moves the robot a certain distance in cm, moving both motors at equal speed in one direction.
  It uses an arbitrary speed of 500, and the function's stepperLeft/stepperRight to control each wheel independently
  If the distance is negative, the robot will move backwards.
*/
void goForward(double distance) {
  int leftPos = distance*distanceConversion;//right motor absolute position
  int rightPos = distance*distanceConversion;//left motor absolute position
  int leftSpd = 500;//right motor speed
  int rightSpd = 500; //left motor speed
```

```
  if(distance < 0) {
    rightSpd = -rightSpd;
    leftSpd = -leftSpd;
  }

  stepperLeft.move(leftPos);//move left wheel to relative position
  stepperRight.move(rightPos);//move right wheel to relative position

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

  steppers.runSpeedToPosition(); // Blocks until all are in position
}

/*
  stop sets both motor's speeds to 0, causing the robot to stop in place.
*/
void stop() {
  stepperLeft.setSpeed(0);
  stepperRight.setSpeed(0);

  runAtSpeed();
}

/*
  randomWander picks a random distance between 10 and 30, and a random angle between -180 and
180, and then turns that angle, and moves that distance.
  It uses the arbitrary speeds of the goForward and goToAngle functions.
  This is combined with the other states to create higher level behaviours.
*/
void randomWander() {

  digitalWrite(redLED, LOW);//turn off red LED
  digitalWrite(ylwLED, LOW);//turn off yellow LED
  digitalWrite(grnLED, HIGH);//turn on green LED
  double randAngle = random(-180,180);
  double randDist = random(10,30);

  goToAngle(randAngle);
  delay(1000);

  goForward(randDist);
  delay(1000);
}

/*
```

```
 aggressiveKid goes forward at an arbitrary speed of 150, before stopping the wheels and holding
position when it detects something within 7 cm of the front lidar.
 There is no filtering, so it will likely stop at some distance less then 7.
*/
void aggressiveKid() {

 // read lidar data from struct
 struct lidar data = RPC.call("read_lidars").as<struct lidar>();
 struct sonar data2 = RPC.call("read_sonars").as<struct sonar>();

 int rightSpd = 150;
 int leftSpd = 150;

 if (data.front > 7) {
  stepperLeft.setSpeed(leftSpd);//set left motor speed
  stepperRight.setSpeed(rightSpd);//set right motor speed
  stepperLeft.runSpeed();
  stepperRight.runSpeed();
 } else {
  stepperLeft.stop();
  stepperRight.stop();
 }
}

/*
 curiousKid will go forward at an arbitrary speed of 350, until an object is detected by the front lidar.
 Then, it will enter a while loop, in which the robot tries to stay between a distance of 11-17 cm in front
of the object.
 We found filtering was necessary to get a precise value, currently the filter looks at about the most
recent 7 values, which does create some delay.
 If the object is taken away, the robot exits the while loop, and continues to go on its way at the
arbitrary speed.
*/
void curiousKid() {

 int gain = 7;
 int followMotorSpd = 10;
 int motorSpd = 350;
 int bandDistance = 3;
 bool done = 0;

 int trackDistance = frontFilterFunction();

 if (trackDistance < 3) {
  stepperLeft.setSpeed(motorSpd);//set left motor speed
  stepperRight.setSpeed(motorSpd);//set right motor speed
  stepperLeft.runSpeed();
```

```
    stepperRight.runSpeed();
  } else {
    stepperLeft.stop();
    stepperRight.stop();
    int distance = 14;
    delay(1000);

    while(done == 0) {
      trackDistance = frontFilterFunction();
      if(trackDistance > distance + bandDistance) {
        motorSpd = (trackDistance - distance)*gain + followMotorSpd;
        stepperLeft.setSpeed(motorSpd);//set left motor speed
        stepperRight.setSpeed(motorSpd);//set right motor speed
        stepperLeft.runSpeed();
        stepperRight.runSpeed();
      } else if (trackDistance < distance - bandDistance) {
        if(trackDistance < 1) {
          done = 1;
        }
        motorSpd = (trackDistance - distance)*gain - followMotorSpd;
        stepperLeft.setSpeed(motorSpd);//set left motor speed
        stepperRight.setSpeed(motorSpd);//set right motor speed
        stepperLeft.runSpeed();
        stepperRight.runSpeed();
      } else {
        motorSpd = 0;
        stepperLeft.setSpeed(motorSpd);//set left motor speed
        stepperRight.setSpeed(motorSpd);//set right motor speed
        stepperLeft.runSpeed();
        stepperRight.runSpeed();
      }
    }
    delay(2000);
  }
}

/*
  shyKid takes in inputs from all the lidar sensors, and outputs speeds to the motors to have the robot
avoid obstacles.
  Currently if it detects all walls, the robot sits in place, if the robot senses something in front, it will turn
away.
  If the robot senses nothing in front, it moves away proportional from the wall/something in front.
  If nothing is detected, the robot just moves forward.
  If front & back is detected, but not the sides, the robot will randomly turn 90 degrees to the left or
right.

  This function can be combined with other to amke higher level behaviour.
```

```
*/
void shyKid() {

  // read lidar data from struct
  struct lidar data = RPC.call("read_lidars").as<struct lidar>();
  int rightSpd = 250;
  int leftSpd = 250;
  int gain = 40;
  int delay_int = 1500;

  if(data.front > minIRDistance && data.left > minIRDistance && data.right > minIRDistance &&
data.back > minIRDistance) {
    //detecting all Walls
    leftSpd = 0;
    rightSpd = 0;
    stepperLeft.setSpeed(leftSpd); //set left motor speed
    stepperRight.setSpeed(rightSpd); //set right motor speed
    stepperLeft.runSpeed();
    stepperRight.runSpeed();
    delay(delay_int);

  } else if (data.front > minIRDistance && data.left > minIRDistance && data.right > minIRDistance &&
data.back < minIRDistance) {
    // detecting Front, Left, & Right
    goToAngle(180);
    delay(delay_int);
    struct lidar data = RPC.call("read_lidars").as<struct lidar>();

  } else if (data.front > minIRDistance && data.left > minIRDistance && data.right < minIRDistance) {
    // detecting Front & Left
    goToAngle(-90);
    delay(delay_int);

  } else if (data.front > minIRDistance && data.left < minIRDistance && data.right > minIRDistance) {
    //Detecting Front & Right
    goToAngle(90);
    delay(delay_int);

  } else if (data.front > minIRDistance && data.left < minIRDistance && data.right < minIRDistance &&
data.back < minIRDistance) {
    // Just Detecting Front
    int backSpeed = (15 - data.front) * 15 + 50;

    rightSpd = -backSpeed;
    leftSpd = -backSpeed;

    stepperLeft.setSpeed(leftSpd);//set left motor speed
```

```
  stepperRight.setSpeed(rightSpd);//set right motor speed
  stepperLeft.runSpeed();
  stepperRight.runSpeed();

 } else if (data.front > minIRDistance && data.left < minIRDistance && data.right < minIRDistance &&
data.back > minIRDistance) {
  // Detecting Front & Back
  if (changeEveryOther ==1) {
   goToAngle(90);
   delay(delay_int);
   changeEveryOther = 0;
  } else {
   goToAngle(90);
   delay(delay_int);
   changeEveryOther = 1;
  }

 } else if (data.front < minIRDistance && data.left < minIRDistance && data.right < minIRDistance &&
data.back < minIRDistance) {
  // Detects Nothing
  leftSpd = 0;
  rightSpd = 0;
  stepperLeft.setSpeed(leftSpd);//set left motor speed
  stepperRight.setSpeed(rightSpd);//set right motor speed
  stepperLeft.runSpeed();
  stepperRight.runSpeed();

 } else {
  // Detecting Nothing in Front
  if (data.left > minIRDistance && data.left < closeIRDistance) {
   leftSpd = leftSpd + (closeIRDistance-data.left) * gain;
  }

  if (data.right > minIRDistance && data.right < closeIRDistance) {
   rightSpd = rightSpd + (closeIRDistance-data.left) * gain;
  }

  stepperLeft.setSpeed(leftSpd);//set left motor speed
  stepperRight.setSpeed(rightSpd);//set right motor speed
  stepperLeft.runSpeed();
  stepperRight.runSpeed();
 }
}

/*
 smartFollow combines the shyKid(), aggressiveKid(), and curiousKid() behaviour to create emergent
behaviour of following a specific distance of an object when it detects something
```

```
 on the front lidar.
 The robot starts in random wander ('wander' state), and then moves until it detects something on the
Lidars, at which point it enters into aggressiveKid ('collide' state),
 and then once a threshold is reached, the robot will enter into shyKid ('avoid' state). Once nothing is
detected, the robot will re-enter the 'wander' state.
 If the robot senses nothing on the lidars during 'collide', it will also re-enter wander.
 This behaviour is made up of the lower level functions, so changing those functions will change the
robot's behaviour.
 a delay of 2000ms or 2 seconds is given between the states to let the lidars detect new distances.
*/
void smartFollow() {
 if(state.equals("wander")) {
  randomWander();
  struct lidar data = RPC.call("read_lidars").as<struct lidar>();

  digitalWrite(redLED,LOW);
  digitalWrite(ylwLED,LOW);
  digitalWrite(grnLED,HIGH);

  if(data.front < dangerDistance || data.back < dangerDistance || data.left < dangerDistance ||
data.right < dangerDistance) {
    state = "collide";
    delay(2000);
  }

 } else if (state.equals("collide")) {
  digitalWrite(redLED,HIGH);
  digitalWrite(ylwLED,LOW);
  digitalWrite(grnLED,LOW);
  aggressiveKid();
  struct lidar data = RPC.call("read_lidars").as<struct lidar>();
  print_lidar();
  if(data.front == 0 && data.back == 0 && data.left == 0 && data.right == 0) {
    state = "wander";
    delay(2000);
  } else if ((int)data.front < 10) {
    state = "avoid";
    delay(2000);
  }
 } else if(state.equals("avoid")) {
  digitalWrite(redLED,LOW);
  digitalWrite(ylwLED,HIGH);
  digitalWrite(grnLED,LOW);
  shyKid();
  struct lidar data = RPC.call("read_lidars").as<struct lidar>();
  if(data.front == 0 && data.back == 0 && data.left == 0 && data.right == 0) {
    state = "wander";
```

```
    delay(2000);
   }
  } else {
   Serial.println("Bad State");
  }
}

/*
  smartWander is very similiar to smartFollow, but instead of going to a specific distance from an object,
the shyKid() function was changed to have the robot go around
  the detected object.
*/
void smartWander() {
  if(state.equals("wander")) {
   randomWander();
   struct lidar data = RPC.call("read_lidars").as<struct lidar>();

   digitalWrite(redLED,LOW);
   digitalWrite(ylwLED,LOW);
   digitalWrite(grnLED,HIGH);

   if(data.front < dangerDistance || data.back < dangerDistance || data.left < dangerDistance ||
data.right < dangerDistance) {
    state = "collide";
    delay(2000);
   }

  } else if (state.equals("collide")) {
   digitalWrite(redLED,HIGH);
   digitalWrite(ylwLED,LOW);
   digitalWrite(grnLED,LOW);
   aggressiveKid();
   struct lidar data = RPC.call("read_lidars").as<struct lidar>();
   print_lidar();
   if(data.front == 0 && data.back == 0 && data.left == 0 && data.right == 0) {
    state = "wander";
    delay(2000);
   } else if ((int)data.front < 10) {
    state = "avoid";
    delay(2000);
   }
  } else if(state.equals("avoid")) {
   digitalWrite(redLED,LOW);
   digitalWrite(ylwLED,HIGH);
   digitalWrite(grnLED,LOW);
   shyKid();
   struct lidar data = RPC.call("read_lidars").as<struct lidar>();
```

```
   if(data.front == 0 && data.back == 0 && data.left == 0 && data.right == 0) {
     state = "wander";
     delay(2000);
    }
  } else {
    Serial.println("Bad State");
  }
}

/*
  The followWall function uses a variety of variables to try and keep the robot within a dead band of 4-6 inches from the wall. Currently, these values ar hardcoded.
  This function also keeps track of the last detected wall in the global variable 'wallDetected' in which a 1 means a wall was detected on the right, 2 on the left, and 3 is both.
  When the robot loses a wall (meeting a corner), the robot will turn the direction of the last detected wall.
  If no wall is detected, the followWall command will just have the robot travel forward at a slow speed until a wall is detected.
*/

void followWall() {

  struct lidar data = RPC.call("read_lidars").as<struct lidar>();

  int leftDistance = data.left;
  int rightDistance = data.right;
  int frontDistance = data.front;

  int rightSpd = 0;
  int leftSpd = 0;
  int regSpd = 100;
  int slowSpd = 75;
  int fastSpd = 125;

  int innerDistance = 17;
  int outerDistance = 21;
  int frontWallMin = 2;
  int frontWallMax = 5;
  int multFact = 3;
  int difference;

  if(frontDistance > 1 && frontDistance < frontWallMax) {
    if (rightDistance > 1 && leftDistance < 1) {
      // Wall is detected in front & right
      goToAngle(90);
    } else if (leftDistance > 1 && rightDistance < 1) {
      // Wall is detected in front & left
```

```
    goToAngle(-90);
   } else {
    // Wall is detected front, left, & right
    goForward(-5);
    goToAngle(180);
   }
   delay(500);
  } else if(leftDistance > 1 && rightDistance > 1) {
   wallDetected = 3;
   // detects Both left and right Wall
   digitalWrite(redLED,LOW);
   digitalWrite(ylwLED,LOW);
   digitalWrite(grnLED,HIGH);
   if(leftDistance > rightDistance + 4) {
    digitalWrite(redLED,LOW);
    digitalWrite(ylwLED,HIGH);
    digitalWrite(grnLED,LOW);
    // Move left
    leftSpd = slowSpd - (leftDistance - rightDistance) * multFact;
    rightSpd = slowSpd + (leftDistance - rightDistance) * multFact;
    Serial.println("Moving Left");
   } else if (rightDistance > leftDistance + 4) {
    digitalWrite(redLED,LOW);
    digitalWrite(ylwLED,HIGH);
    digitalWrite(grnLED,LOW);
    // Move right
    rightSpd = slowSpd - (rightDistance - leftDistance) * multFact;
    leftSpd = slowSpd + (rightDistance - leftDistance) * multFact;
    Serial.println("Moving Right");
   } else {
    // within Deadband
    leftSpd = regSpd;
    rightSpd = regSpd;
    Serial.println("Moving Forward");
   }
  } else if(leftDistance > 0) {
   wallDetected = 2;
   leftDistance = leftFilterFunction();
   if(leftDistance < innerDistance) {
    // Too close to left wall, move Right
    digitalWrite(redLED,LOW);
    digitalWrite(ylwLED,HIGH);
    digitalWrite(grnLED,LOW);
    difference = innerDistance - leftDistance;
    goToAngle(-difference*multFact);
    goForward(difference);
    goToAngle(difference*multFact);
```

```
  } if (leftDistance > outerDistance) {
    // Too far from left wall, move Left
    digitalWrite(redLED,HIGH);
    digitalWrite(ylwLED,LOW);
    digitalWrite(grnLED,LOW);
    difference = leftDistance - outerDistance;
    goToAngle(difference*multFact);
    goForward(difference);
    goToAngle(-difference*multFact);
  } else {
    // Correct Distance, Stay The Course
    digitalWrite(redLED,LOW);
    digitalWrite(ylwLED,LOW);
    digitalWrite(grnLED,LOW);
    leftSpd = regSpd;
    rightSpd = regSpd;
  }
 } else if(rightDistance > 0) {
  wallDetected = 1;
  rightDistance = rightFilterFunction();
  if(rightDistance < innerDistance) {
    // Too close to right wall, move Left
    digitalWrite(redLED,LOW);
    digitalWrite(ylwLED,HIGH);
    digitalWrite(grnLED,LOW);
    difference = innerDistance - rightDistance;
    goToAngle(difference*multFact);
    goForward(difference);
    goToAngle(-difference*multFact);
  } if (rightDistance > outerDistance) {
    // Too far from right wall, move right
    digitalWrite(redLED,HIGH);
    digitalWrite(ylwLED,LOW);
    digitalWrite(grnLED,LOW);
    difference = rightDistance - outerDistance;
    goToAngle(-difference*multFact);
    goForward(difference);
    goToAngle(difference*multFact);
  } else {
    // Correct Distance, Stay The Course
    digitalWrite(redLED,LOW);
    digitalWrite(ylwLED,LOW);
    digitalWrite(grnLED,LOW);
    leftSpd = regSpd;
    rightSpd = regSpd;
  }
 } else {
```

```
    // No Sensors
    digitalWrite(redLED,HIGH);
    digitalWrite(ylwLED,LOW);
    digitalWrite(grnLED,HIGH);
    if (wallDetected == 1) {
      // Was previously sensing right wall
      goForward(15);
      goToAngle(-90);
      goForward(45);
      wallDetected = 0;
    } else if (wallDetected == 2) {
      // Was previously sensing left wall
      goForward(15);
      goToAngle(90);
      goForward(45);
      wallDetected = 0;
    } else if (wallDetected == 0) {
      randomWander();
    }
    leftSpd = regSpd;
    rightSpd = regSpd;
  }

  stepperLeft.setSpeed(leftSpd);//set left motor speed
  stepperRight.setSpeed(rightSpd);//set right motor speed
  stepperLeft.runSpeed();
  stepperRight.runSpeed();
}
/*
  Fcn hallway goes down the hallway, returns, and then turns based on the detected wall dissapearance
*/
void hallway() {

  while(state.equals("forward")) {
    struct lidar data = RPC.call("read_lidars").as<struct lidar>();
    Serial.println(data.front);
    digitalWrite(grnLED, HIGH);
    digitalWrite(ylwLED, LOW);
    digitalWrite(redLED, LOW);

    if(data.front > 1 && data.front < 10) {
      state = "backward";
      digitalWrite(grnLED, LOW);
      digitalWrite(ylwLED, HIGH);
      digitalWrite(redLED, LOW);
      delay(500);
    }
```

```
  leftSpd = 150;
  rightSpd = 150;

  stepperLeft.setSpeed(leftSpd);//set left motor speed
  stepperRight.setSpeed(rightSpd);//set right motor speed
  stepperLeft.runSpeed();
  stepperRight.runSpeed();
 }
 while(state.equals("backward")) {
  struct lidar data = RPC.call("read_lidars").as<struct lidar>();

  if(data.left < 1) {
   state = "follow";
   digitalWrite(grnLED, LOW);
   digitalWrite(ylwLED, LOW);
   digitalWrite(redLED, HIGH);
   delay(500);
   goForward(-25);
   delay(500);
   goToAngle(90);
   delay(500);
  }
  if(data.right < 1) {
   state = "follow";
   digitalWrite(grnLED, LOW);
   digitalWrite(ylwLED, HIGH);
   digitalWrite(redLED, HIGH);
   delay(500);
   goForward(-25);
   delay(500);
   goToAngle(-90);
   delay(500);
  }

  leftSpd = -150;
  rightSpd = -150;

  stepperLeft.setSpeed(leftSpd);//set left motor speed
  stepperRight.setSpeed(rightSpd);//set right motor speed
  stepperLeft.runSpeed();
  stepperRight.runSpeed();
 }
 while(state.equals("follow")) {
  followWall();
 }
```

```
}

/*
 goToGoal command keeps track of the robots distance in X & Y space to allow the robot to move around any obstacle. This assumes the object is oriented parallel or perpendicular
 to the robot's starting configuration.
 If no object is in place, the robt will move forward until it reaches the goal.
 If an object is in place, the robt will move forward until the object is detected, then go around the object, and return back in a direction perpendicular to the path (Y direction)
 until the distance to theobject is minimized, at which point the robot will turn and move towards the goal.
*/
void goToGoal() {

 if(state.equals("forward")) {
  digitalWrite(grnLED, HIGH);
  digitalWrite(ylwLED, LOW);
  digitalWrite(redLED, LOW);
  if(data.front > 2 && data.front < 10) {
   state = "avoid";
   delay(500);
  } else if (frontDistance < 200) {
   goForward(1);
   frontDistance = frontDistance + 1;
  } else {
   stepperLeft.stop();
   stepperRight.stop();
  }
 }

 if(state.equals("avoid")) {
  digitalWrite(grnLED, LOW);
  digitalWrite(ylwLED, LOW);
  digitalWrite(redLED, HIGH);
  if(nonFrontDistance == 0) {
   goToAngle(90);
   state = "forward";
   direction = "north";
  } else if(data.left > 2 || data.right > 2) {
   goForward(1);
   if(direction.equals("north")) {
    frontDistance = frontDistance + 1;
   } else if (direction.equals("west")) {
    nonFrontDistance = nonFrontDistance + 1;
   } else if (direction.equals("east")) {
    nonFrontDistance = nonFrontDistance - 1;
   }
```

```
   } else if (data.front < 10 && data.front > 1) {
     digitalWrite(grnLED, LOW);
     digitalWrite(ylwLED, HIGH);
     digitalWrite(redLED, LOW);
     goToAngle(90);
     direction = "west";
     delay(500);
   } else if (data.left < 2 && data.right < 2) {
     digitalWrite(grnLED, LOW);
     digitalWrite(ylwLED, HIGH);
     digitalWrite(redLED, LOW);
     int littleBit = 10;
     int largeBit = 40;
     goForward(littleBit);
     delay(500);
     goToAngle(-90);
     delay(500);
     goForward(largeBit);
     if(direction.equals("west")) {
       nonFrontDistance = nonFrontDistance + littleBit;
       frontDistance = frontDistance + largeBit;
       direction = "north";
     } else if (direction.equals("north")) {
       frontDistance = frontDistance + littleBit;
       nonFrontDistance = nonFrontDistance - largeBit;
       direction = "east";
     } else if (direction.equals("east")) {
       nonFrontDistance = nonFrontDistance - littleBit;
       frontDistance = frontDistance - largeBit;
       direction = "south";
     }
   }
  }

}

// Setup for Multi-Core
//set up the M4 to be the server for the sensors data
void setupM4() {
  RPC.bind("read_lidars", read_lidars);  // bind a method to return the lidar data all at once
  RPC.bind("read_sonars", read_sonars);  // bind a method to return the lidar data all at once
}

//poll the M4 to read the data
void loopM4() {
  // update the struct with current lidar data
  dist.front = read_lidar(8);
```

```
  dist.back = read_lidar(9);
  dist.left = read_lidar(10);
  dist.right = read_lidar(11);
  dist2.right = read_sonar(3);
  dist2.left = read_sonar(4);
}

//set up the M7 to be the client and run the state machine
void setupM7() {

  // begin serial interface
  state = "forward";
  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

  Serial.begin(baudrate);     //start serial monitor communication
  Serial.println("Robot starting...Put ON TEST STAND");
  delay(pauseTime); //always wait 2.5 seconds before the robot moves
}

//read sensor data from M4 and write to M7
void loopM7() {
  struct lidar data = RPC.call("read_lidars").as<struct lidar>();

}

//setup function with infinite loops to send and receive sensor data between M4 and M7
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
// this may need to be modified to run th estate machine.
```

```
// consider usingnamespace rtos Threads as seen in previous example
void loop() {}
```