Lab 02

Avoid-Obstacle, Follow-Object and Random Wander Worksheet

Robot Name _____Arkin_____

Team Member Name _____Ben Williams_____

Team Member name _____Donald Woodruff_____

Purpose

In your own words, state the purpose of lab 02 in the following space.

To use sonar and lidar sensor data in combination with feedback control schemes to control a mobile robot to avoid or follow objects.

This is done through the creation of 'layers' of processing, which build off the previous 'layers' to create more complex behaviors.

Part I – Random Wander (Layer 1)

What was the general plan you used to implement the random wander and obstacle avoidance behaviors?

The robot heads in a direction randomly until it detects an object using the Lidar sensors (either in front, behind, to the left or right of the robot). If there is an object in front of the robot, the robot will quickly turn to avoid driving into the obstacle. If there is an object beside the robot, the robot will gradually turn away from that object, and if an object is detected behind, the robot moves forward away from the object.

Part II – Test the Sonar Sensors

Based upon your testing, what is the maximum and minimum range for the sonar sensors on your robot?

The sonar sensors seemed to function from 5 to 30 cm, however even within that range the distances reported would fluctuate dramatically even when standing still. This led to us mostly not incorporating sonar sensing.

What is one advantage of using a sonar sensor instead of an infrared sensor?

The sonar sensors had a larger max range, and their position was adjustable, while the lidars were fixed.

In the following space, discuss the accuracy of the sensor.

Outside of the 5 to 30 cm range, the sensor readings were effectively useless, and even within that range the reported values would fluctuate dramatically, even when the robot and object were staying at a fixed distance.

Part III – Test the Infrared Lidar Sensors

Based upon your testing, what is the maximum and minimum range for the infrared lidar sensors on your robot?

The infrared lidar sensors had a range of about 2cm to 20cm.

What is one advance of using an infrared lidar instead of a sonar sensor?

The infrared lidar sensors seemed to be much more accurate than the sonar sensors

In the following space, discuss the accuracy of the sensor.

The sensor was much more accurate than the sonar sensors from our testing, with the sensor's readings usually staying within 5 cm of the actual distance.

One problem encountered with the lidar was the random occurrence of the lidar reporting a distance of 0. The lidar would also report a distance of 0 if nothing was within range of the sensor, or if the sensor was fully blocked, making it difficult to tell if something was within 1 cm of the lidar, or far away.

Part IV – Collide Behavior

What distance did you use for the collide behavior?

For collide, we used 14cm (or any value less then 14 cm) to trigger the collision behavior.

Did you use the lidar, sonar or a combination of the two sensors? How did you decide?

We used exclusively lidar sensors for our collide behavior, because the sonar sensor readings were too inconsistent to be useful.

How could you implement collide behavior for sensors other than the front sensors?

For the left and right sensors, if the robot detects a close object, the robot will speed up the wheel on the side closest to that object, causing the robot to gradually turn away from the object. For objects detected behind the robot, the robot will stop if it is moving backwards and towards the object.

Part V – Runaway Behavior (Layer 0)

How did you represent the feel force function on your robot?

The sensor readings from all of the robots lidar sensors were summed to create a repulsive vector, which was used to turn the robot away. The robot then turns to this angle and continues it forward motion.

How did the robot respond with respect to the force felt, note that it could move in reverse, or it could turn a certain angle, think about what makes most sense for potential field navigation.

A potential field would likely have the robot move in reverse in order to most effectively move away from the object. Unfortunately, with the current sensing setup, the robot instead just turned to an angle and move away from the objects by moving forward.

How does your robot handle getting the robot unstuck when the vectors sum to zero?

The robot turns to a 45 degree angle, to allow the sensors to see new areas and it may then be able to escape. The robot is also given a slight perturbation, to have it randomly move slightly in any direction.

Part VI – Follow Behavior (Alternate Layer 0)

In your own words, describe how to implement proportional control to follow an object,

The distance between the mobile robot and the object is measured, then the motor speeds are controlled proportionally to the distance measured. If the robot is too close to the object, it will move away. If it is too far, it will move closer. If it is within the desired distance, it will stop moving.

Are there situations where the robot would get stuck when following? If so, how would the robot correct for that?

If the robot finds itself stuck in the same position for a long set amount of time, it could turn or move a prescribed distance to get itself unstuck. That being said, the follow behavior would also presumably result in the robot staying at a fixed distance from the object.

Part VII – Subsumption Architecture – Smart Wander Behavior (Layers 0 and 1)

What is the difference between on-off and proportional control?

On and off (or bang-bang control) simply turns the motors on and off with no speed adjustment, whereas proportional control will change the speed of the motors depending on the error in the desired value, such as the error of a desired distance from an object.

How did feedback control improve the random wander and avoid obstacle state machine?

Using feedback control allows for more finite adjustments of the motor control and thus robot position while moving with respect to some target, such as moving away from an object.

Part VIII – Subsumption Architecture – Smart Follow Behavior (Layers 0 and 1)

You robot currently runs either a smart wander or smart follow behavior. How could you create a state machine that integrates smart follow and smart wander? What type of input would trigger the avoid versus the follow behavior?

The state machine would switch between smart follow and smart wander depending on whether the robot senses an object near it. If there is an object that is detected too close to the robot, the robot can use smart follow to move away from it. Otherwise, it will continue to smart wander. The robot could also decide to follow specific objects (smart follow) or just move around the environment (smart wander) based on other sensor or feedback data.

Conclusions

1. **How well did your software design plans match the reality of what you implemented on the robot?**

There were lots of aspects to the implementation that we did not account for in our design plan, however we roughly followed the design plan well.

2. **How well did your software design plans match the reality of how the robot performed? Compare it to the theory you learned in class.**

Our software design plan did not encapsulate a large majority of the functionality that was needed for our actual robot. Certain additional features had to be implemented, such as sensor data filtering, and taking into account the speed of the robot in regard to getting timely sensor data.

3. **How did you create a modular program and integrate the two layers into the overall program?**

We implemented a two layered state machine to control the robot. The bottom layer was just the base functions (follow, wander, avoid), and the state machine controlled the higher level

behavior (smart follow, smart wander).

4. Did you use the sonar and IR sensors to create redundant sensing on the robot's front half?

Through testing, we discovered that the sonar sensors are highly inaccurate, so we used only the lidar sensors for sensing.

5. How could you create a smart wander routine to entirely cover a room? Similar to what a Roomba does.

The robot could move in a certain direction until it hits an object, then turn 90 degrees, move a short distance, then turn 90 degrees again. This would allow for the robot to move back and forth across the floor.

6. What kind of errors did you encounter with obstacle avoidance behavior?

We had difficulties with sensors reading consistently, which caused it to be difficult to avoid obstacles as consistently as we would like. To remedy this, we incorporated filtering. Additionally, the lidar's coverage is fairly small, so even small objects at a slight angle from the robot wouldn't be picked up, and might be ran into.

7. How could you improve obstacle avoidance behavior?

Adding sensor redundancy by using both sonar and lidar sensors would help greatly with ensuring accurate sensor readings.

8. Were there any obstacles that the robot could not detect?

Obstacles that were small were often hard for the robot to detect, especially if they were at a slight angle with regard to the lidars.

9. Were there any situations when the range sensors did not give you reliable data?

The sonar sensors often gave inaccurate and inconsistent data. The lidar sensors worked better, but would occasionally gives readings of 0 even when something was placed in front of it, which had to be dealt with through the robot's programming/filtering.

10. How did you keep track of the robot's states in the program?

A state machine was used to keep track of the robot's states. Certain triggers were used to transition the robot between states.

11. Did the robot encounter any "stuck" situations? How did you account for those?

If the robot got stuck, the robot would turn 45 degrees and take new sensor readings. If that failed, a slight perturbation was given to the robot to try and remove it from the local minimum.

12. What should the subsumption architecture look like for the addition of the go-to-goal and avoid- obstacle behaviors?

Go to goal should run until an object is detected, at which point the object should move alongside the object while still avoiding it. Once the object is no longer blocking the path towards the goal, go to goal should resume.

13. What did you learn?

We learned a lot about handling sensor inaccuracies, how to implement on-off and proportional control, how to recover from stuck positions, and how to use dual cores to read sensor data in the background.

14. What did you observe?

We observed that the polling rate of the sensors seemed to be surprising low, which sometimes made it difficult for the robot to react fast enough to objects in it's path.

15. What questions do you still have?

How could poor sensing data be dealt with better?

Appendix

```
/*
**********************************
  RobotIntro.ino
  Donald & Ben 1/24/25

  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;
```

```
//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);

String state = "";
int dangerDistance = 10;
int offset = 3;
int avoidDistance;

// 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 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");
  }
}

// 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 = "wander";
```

```
  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() {

}

//// MAIN

//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() {}
```