# Lab 02

# Avoid-Obstacle, Follow-Object and Random Wander Worksheet

Robot Name \_\_\_\_\_\_\_\_\_Murphy\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Team Member Name \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Salik Ahmad\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Team Member name \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Jackson Seida\_\_\_\_\_\_\_\_\_\_

Team Member Name \_\_\_\_\_\_\_\_\_\_\_\_\_Justin DeWitt\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Purpose

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

## Part I – Random Wander (Layer 1)

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

We made the random wander behavior turn a random angle, and then set a random number of steps to travel. For the random wander behavior, we wanted it to be non-blocking; thus, a non-blocking call to movement is used.

## Part II – Test the Sonar Sensors

Based upon your testing, what is the maximum and minimum range for the sonar sensors on your robot?  
Max: 30cm

Min: 2cm

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

Sonar sensors will not be disrupted by ambient light or dust.

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

Fairly accurate within the range (within 1-2 cm); of course, the sensors is extremely inaccurate once outside of the range specified.

## 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?

Max: 39

Min: 1

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

Lidar sensors are more accurate than sonar sensors and have higher resolution.

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

More accurate than the sonar in the range (within 1 cm), of course, outside of range it is not very accurate.

## Part IV – Collide Behavior

What distance did you use for the collide behavior?

15 cm

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

Lidar because for the collide behavior, we only needed an idea of the general direction in the cardinal directions. The lidars were implemented in the cardinal directions, while the sonar in at 45 degree angles between the cardinal directions. We saw no need to use sonar yet.

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

In a similar way to the front sensors; if any sensor suddenly detects an object/obstacle within the specified distance, the collide behavior will activate.

## Part V – Runaway Behavior (Layer 0)

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

We represented it as a vector made of an x and y variable and a corresponding angle variable. These were computed using the readings from the sensors.

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.

For our potential field navigation, we had the robot turn in most scenarios and then move away from the repulsive vector, but in one case where the repulsive vector directly in front, the object would move directly backward in reverse.

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

When the vectors sum for 0, we directly check the sensor readings to see where the open direction is, and have our robot drive there. Thus, we handle these as edge cases with if statements.

## Part VI – Follow Behavior (Alternate Layer 0)

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

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

In order to implement proportional control, we measure the distance from the object, and we calculate the error based on the intended distance from the target. Then, we use this error times some constant to calculate our speed to move forward/backward. The robot may get stuck oscillating, in which case the proportionality constant may have to be changed to be lower. Another example is a suddenly changing enviroment may cause the robot to get stuck, such as sudden obstacles. The robot would need a way to handle edge cases using sensor input and recognize the obstacle and move around it, and continue to follow.

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

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

On-off implies that the output of the controller is either on or completely off. For example, in a motor this would mean that the motor is either rotating, or not moving. In proportional control the magnitude of the error affects the magnitude of the output.

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

Feedback control improved the state machine because it allowed for the robot to recognize when it needed to change states.

## 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?

If the input reads that an object is very close to the robot the avoid behavior could be activated. When there is an object far away from the robot the follow behavior could be activated.

## Conclusions

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

Fairly well, our state machines and flowcharts matched the implementation of our code. Our code implemented the same states and same state transitions.

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

The robot performed well according to our software design plans; we had to some minor adjustments to certain distances for when to transition, but our robot still worked according to our general theory.

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

The program was modular because the wander behavior, and the follow and avoid behaviors were created separately. The robot then changed between the behaviors via a state machine.

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

The IR sensors would detect the object and its distance while the sonar was at an angle and helped determine what route was the best when avoiding the obstacle while turning, and this redundant sensing also helped us avoid errors and situations where the robot may get stuck.

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

You would implement the smart wander routine, but you could have the robot also keep track of a map of where it has been, and thus, the robot will slowly finish its map as it wanders; obstacles are treated as walls.

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

Our robot did not know how to handle walls on the front and back; Our robot would sometimes turn into walls when trying to avoid them. Our robot did not know how to handle being boxed in. Those are some of the errors we encountered.

1. How could you improve obstacle avoidance behavior?

Have the robot account for its own turn radius, and adjust its position from the wall before it makes the turn to avoid hitting walls and collisions is one possible way to improve the behavior.

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

No the robot did not run into that problem.

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

Yes we had that situation with our sonar sensor, because we were reading them too frequently, and were not using dual core probably.

1. How did you keep track of the robot’s states in the program?

We had the robot have boolean variables that helped us transition from state to state and we used cases in the code for each state.

1. Did the robot encounter any “stuck” situations? How did you account for those?

Yes the robot was stuck when walls were on its front and back, and when it was boxed in. We accounted for those by handling them as edge cases. We added code specifically for those situations and corresponding sensor readings where the robot would turn and drive out, or where the robot would stop.

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

The go-to-goal behavior would be the highest priority behavior and on the top layer, and the avoid obstacle behavior would factor in one layer below. These behaviors would have the highest priority in the subsumption architecture, where the robot would attempt to go to goal, but the avoid-obstacle behavior could interrupt, triggered by behaviors such as collide/runaway in lower layers.

1. What did you learn?

We learned what a subsumption architecture was, and how to actually implement it in a real mobile robot and in our code. We also learned how to have our robot avoid walls and escape situations that our general state diagram could not account for (edge cases). We learned about potential fields theory and how to also implement that within our robot’s functioning.

1. What did you observe?

We observed that sensors will occasionally have erratic readings, so it is important and useful to have redundant sensing and take frequent sensing measurements. We also observed that there are many situations where a robot can be put in a situation that the general program cannot account for. Therefore, the coders must account for those situations.

1. What questions do you still have?

We wonder what different architectures are available outside of the subsumption architecture, and what they look like in real-time when implemented?

## Appendix

Insert your properly commented, modular code in the following space.

/\*

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];

//state LEDs connections

#define redLED 5 //red LED for displaying states

#define grnLED 6 //green LED for displaying states

#define ylwLED 7 //yellow LED for displaying states

#define enableLED 13 //stepper enabled LED

int leds[3] = { 5, 6, 7 }; //array of LED pin numbers

//define motor pin numbers

#define stepperEnable 48 //stepper enable pin on stepStick

#define rtStepPin 50 //right stepper motor step pin

#define rtDirPin 51 // right stepper motor direction pin

#define ltStepPin 52 //left stepper motor step pin

#define ltDirPin 53 //left stepper motor direction pin

//define the Lidar constants

#define LIDAR\_FRONT 0

#define LIDAR\_BACK 1

#define LIDAR\_LEFT 2

#define LIDAR\_RIGHT 3

#define numOfSens 4

//define the behavior constants

#define NO\_BEHAVIOR 0

#define COLLIDE 1

//define the Lidar variables

int16\_t ft\_lidar = 8;

int16\_t bk\_lidar = 9;

int16\_t lt\_lidar = 10;

int16\_t rt\_lidar = 11;

int16\_t lidar\_pins[numOfSens] = {8,9,10,11};

int16\_t lidarDist[numOfSens] = {0,0,0,0};

//define the Sonar constants

#define VELOCITY\_TEMP(temp) ((331.5 + 0.6 \* (float)(temp)) \* 100 / 1000000.0) // The ultrasonic velocity (cm/us) compensated by temperature

#define SONAR\_RIGHT 0

#define SONAR\_LEFT 1

//define the Sonar variables

int16\_t rt\_trigechoPin = 3;

int16\_t lt\_trigechoPin = 4;

int16\_t trig\_EchoPin[2] = { 3,4 };

int16\_t sonarDist[2] = {0,0};

AccelStepper stepperRight(AccelStepper::DRIVER, rtStepPin, rtDirPin); //create instance of right stepper motor object (2 driver pins, low to high transition step pin 52, direction input pin 53 (high means forward)

AccelStepper stepperLeft(AccelStepper::DRIVER, ltStepPin, ltDirPin); //create instance of left stepper motor object (2 driver pins, step pin 50, direction input pin 51)

MultiStepper steppers; //create instance to control multiple steppers at the same time

#define stepperEnTrue false //variable for enabling stepper motor

#define stepperEnFalse true //variable for disabling stepper motor

int pauseTime = 2500; //time before robot moves

int stepTime = 500; //delay time between high and low on step pin

int wait\_time = 1000; //delay for printing data

#define WANDER\_TIME 4000 //time between change of wander wheel speeds in millis

int wanderTimer = 0; //timer to determine when to change wander wheel speeds

//define encoder pins

#define LEFT 0 //left encoder

#define RIGHT 1 //right encoder

const int ltEncoder = 18; //left encoder pin (Mega Interrupt pins 2,3 18,19,20,21)

const int rtEncoder = 19; //right encoder pin (Mega Interrupt pins 2,3 18,19,20,21)

volatile long encoder[2] = { 0, 0 }; //interrupt variable to hold number of encoder counts (left, right)

int lastSpeed[2] = { 0, 0 }; //variable to hold encoder speed (left, right)

int accumTicks[2] = { 0, 0 }; //variable to hold accumulated ticks since last reset

bool run = false;

struct sensor\_data {

// this can easily be extended to contain sonar data as well

int lidar\_front;

int lidar\_back;

int lidar\_left;

int lidar\_right;

int sonar\_left;

int sonar\_right;

// this defines some helper functions that allow RPC to send our struct (I found this on a random forum)

MSGPACK\_DEFINE\_ARRAY(lidar\_front, lidar\_back, lidar\_left, lidar\_right, sonar\_left, sonar\_right)

} sensors;

// read\_lidars is the function used to get lidar data to the M7

struct sensor\_data read\_sensors() {

return sensors;

}

// reads a lidar given a pin

int read\_lidar(int pin) {

int16\_t t = pulseIn(pin, HIGH);

int d; //distance to object

if (t == 0){

// pulseIn() did not detect the start of a pulse within 1 second.

//Serial.println("timeout");

d = 100000; //no object detected

}

else if (t > 1850) {

//Serial.println("timeout");

d = 100000; //no object detected

}

else {

// Valid pulse width reading. Convert pulse width in microseconds to distance in millimeters.

d = (t - 1000) \* 3 / 40;

// Limit minimum distance to 0.

if (d < 0) { d = 0; }

}

// Serial.print(d);

// Serial.print(" cm, ");

return d;

}

int movingAverage(int arr[], int arrSize) {

int sum = 0;

for (int i = 0; i < arrSize; i++) {

sum += arr[i];

}

return sum / arrSize;

}

int\* shiftArray(int arr[], int arrSize, int newValue) {

for (int i = arrSize - 1; i > 0; i--) {

arr[i] = arr[i - 1];

}

arr[0] = newValue;

return arr;

}

void setupM4() {

// bind a method to return the lidar data all at once

RPC.bind("read\_sensors", read\_sensors);

}

void loopM4() {

// update the struct with current lidar data

struct sensor\_data data;

int lidarFrontCurr = read\_lidar(8);

int lidarBackCurr = read\_lidar(9);

int lidarLeftCurr = read\_lidar(10);

int 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);

int sonarLeftCurr = readSonar(SONAR\_LEFT);

int sonarRightCurr = readSonar(SONAR\_RIGHT);

leftSonarArr = shiftArray(leftSonarArr, SONAR\_ARR\_SIZE, sonarLeftCurr);

rightSonarArr = shiftArray(rightSonarArr, SONAR\_ARR\_SIZE, sonarRightCurr);

data.sonar\_left = movingAverage(leftSonarArr, SONAR\_ARR\_SIZE);

data.sonar\_right = movingAverage(rightSonarArr, SONAR\_ARR\_SIZE);

sensors = data;

}

// Helper Functions

//interrupt function to count left encoder tickes

void LwheelSpeed() {

encoder[LEFT]++; //count the right 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(500); //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(500); //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

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;

}

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);

}

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");

}

}

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();

}

void follow(void) {

digitalWrite(redLED, HIGH); //turn on red LED

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

int maxSpeed = 300;

int rightSpeed;

int leftSpeed;

int x = 0;

int y = 0;

sensors = RPC.call("read\_sensors").as<struct sensor\_data>();

// Serial.print("left = ");

// Serial.print(sensors.sonar\_left);

// Serial.print(" right = ");

// Serial.println(sensors.sonar\_right);

// Determine x direction of attractive vector

if (sensors.lidar\_back < 30){

x += -30 + sensors.lidar\_back;

}

if (sensors.lidar\_front < 30){

x += 30 - sensors.lidar\_front;

}

if (sensors.sonar\_left < 15) {

x += 15 - sensors.sonar\_left;

}

if (sensors.sonar\_right < 15) {

x += 15 - sensors.sonar\_right;

}

// Determine y direction of attractive vector

if (sensors.lidar\_right < 30){

y += -30 + sensors.lidar\_right;

}

if (sensors.lidar\_left < 30){

y += 30 - sensors.lidar\_left;

}

if (sensors.sonar\_left < 15) {

y += 15 - sensors.sonar\_left;

}

if (sensors.sonar\_right < 15) {

y += -15 + sensors.sonar\_right;

}

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

Serial.print("x = ");

Serial.print(x);

Serial.print(" y = ");

Serial.print(y);

Serial.print(" angle = ");

Serial.println(angle);

if(abs(y) > 5 || abs(x) > 5) {

if (angle > -30 && angle < 30 && abs(x) < 25 ) {

rightSpeed = maxSpeed;

leftSpeed = maxSpeed;

Serial.println("Forward");

} else if (angle > -30 && angle < 30 && abs(x) > 35 ) {

rightSpeed = -maxSpeed;

leftSpeed = -maxSpeed;

Serial.println("Backward");

} else if (angle >= 30 && angle <= 180) {

rightSpeed = maxSpeed;

leftSpeed = -maxSpeed;

Serial.println("Left");

} else if (angle <= -30 && angle >= -180) {

rightSpeed = -maxSpeed;

leftSpeed = maxSpeed;

Serial.println("Right");

} else {

rightSpeed = 0;

leftSpeed = 0;

}

} else {

rightSpeed = 0;

leftSpeed = 0;

}

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

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

runAtSpeed();

}

#define STATE\_WANDER 0

#define STATE\_COLLIDE 1

#define STATE\_RUNAWAY 2

int state = 0;

void smartWander(void) {

sensors = RPC.call("read\_sensors").as<struct sensor\_data>();

switch (state) {

case STATE\_WANDER:

digitalWrite(ylwLED, LOW); //turn off yellow LED

Serial.println("wander");

randomWander();

if (sensors.lidar\_front < 15 || sensors.lidar\_back < 15 || sensors.lidar\_right < 15 || sensors.lidar\_left < 15) {

state = STATE\_COLLIDE;

}

break;

case STATE\_COLLIDE:

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

Serial.println("collide");

collide();

delay(1000);

state = STATE\_RUNAWAY;

break;

case STATE\_RUNAWAY:

digitalWrite(redLED, LOW); //turn off red LED

Serial.println("runaway");

runaway();

if (sensors.lidar\_front > 20 && sensors.lidar\_back > 20 && sensors.lidar\_right > 20 && sensors.lidar\_left > 20) {

state = STATE\_WANDER;

}

break;

default:

Serial.println("left state machine");

break;

}

}

#define STATE\_FOLLOW 3

void smartFollow(void) {

sensors = RPC.call("read\_sensors").as<struct sensor\_data>();

switch (state) {

case STATE\_WANDER:

digitalWrite(redLED, LOW); //turn off yellow LED

digitalWrite(grnLED, LOW); //turn off yellow LED

Serial.println("wander");

randomWander();

if (sensors.lidar\_front < 15 || sensors.lidar\_back < 15 || sensors.lidar\_right < 15 || sensors.lidar\_left < 15) {

state = STATE\_COLLIDE;

}

break;

case STATE\_COLLIDE:

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

Serial.println("collide");

collide();

delay(1000);

state = STATE\_FOLLOW;

break;

case STATE\_FOLLOW:

digitalWrite(redLED, LOW); //turn off red LED

Serial.println("follow");

follow();

if (sensors.lidar\_front > 20 && sensors.lidar\_back > 20 && sensors.lidar\_right > 20 && sensors.lidar\_left > 20 && sensors.sonar\_left > 20 && sensors.sonar\_left > 20) {

state = STATE\_WANDER;

}

break;

default:

Serial.println("left state machine");

break;

}

}

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);

}

pinMode(3, INPUT);

pinMode(4, 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

//collide();

// runaway();

// follow();

// smartWander();

smartFollow();

//delay(wait\_time); //wait to move robot or read data

}