//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Define Ultrasonics Pins \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//

#define TRIG\_FRONT\_LEFT 5

#define ECHO\_FRONT\_LEFT 18

#define TRIG\_FRONT\_MID 19

#define ECHO\_FRONT\_MID 21

#define TRIG\_FRONT\_RIGHT 22

#define ECHO\_FRONT\_RIGHT 23

#define TRIG\_BACK 32

#define ECHO\_BACK 33

#define TRIG\_LEFT 25

#define ECHO\_LEFT 26

#define TRIG\_RIGHT 27

#define ECHO\_RIGHT 14

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Define Motors Pins \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//

#define LEFT\_MOTOR\_P1 2

#define LEFT\_MOTOR\_P2 4

#define RIGHT\_MOTOR\_P1 15

#define RIGHT\_MOTOR\_P2 13

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Define IR Sensor Pins \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//

#define IR\_FRONT\_PIN 34

#define IR\_BACK\_PIN 35

#define IR\_LEFT\_PIN 36

#define IR\_RIGHT\_PIN 39

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Define constants \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//

#define SPEED 255

#define TURN\_SPEED 155

#define minDistance 15

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ultrasonic Class \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//

class Ultrasonic

{

private:

int \_trigPin;

int \_echoPin;

public:

Ultrasonic(int trigPin, int echoPin)

{

\_trigPin = trigPin;

\_echoPin = echoPin;

pinMode(\_trigPin, OUTPUT);

pinMode(\_echoPin, INPUT);

}

float read()

{

digitalWrite(\_trigPin, LOW);

delayMicroseconds(2);

digitalWrite(\_trigPin, HIGH);

delayMicroseconds(10);

digitalWrite(\_trigPin, LOW);

long duration = pulseIn(\_echoPin, HIGH);

float distance = duration \* 0.034 / 2;

return distance;

}

};

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Motor Class \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//

class Motor

{

private:

int \_leftMotorPin1;

int \_leftMotorPin2;

int \_rightMotorPin1;

int \_rightMotorPin2;

public:

Motor(int leftMotorPin1, int leftMotorPin2, int rightMotorPin1, int rightMotorPin2)

{

\_leftMotorPin1 = leftMotorPin1;

\_leftMotorPin2 = leftMotorPin2;

\_rightMotorPin1 = rightMotorPin1;

\_rightMotorPin2 = rightMotorPin2;

pinMode(\_leftMotorPin1, OUTPUT);

pinMode(\_leftMotorPin2, OUTPUT);

pinMode(\_rightMotorPin1, OUTPUT);

pinMode(\_rightMotorPin2, OUTPUT);

}

void moveForward(int speed)

{

setMotorSpeed(\_leftMotorPin1, \_leftMotorPin2, speed);

setMotorSpeed(\_rightMotorPin1, \_rightMotorPin2, speed);

}

void moveBackward(int speed)

{

setMotorSpeed(\_leftMotorPin1, \_leftMotorPin2, -speed);

setMotorSpeed(\_rightMotorPin1, \_rightMotorPin2, -speed);

}

void turnLeft(int speed)

{

setMotorSpeed(\_leftMotorPin1, \_leftMotorPin2, -speed);

setMotorSpeed(\_rightMotorPin1, \_rightMotorPin2, speed);

}

void turnRight(int speed)

{

setMotorSpeed(\_leftMotorPin1, \_leftMotorPin2, speed);

setMotorSpeed(\_rightMotorPin1, \_rightMotorPin2, -speed);

}

void stop()

{

setMotorSpeed(\_leftMotorPin1, \_leftMotorPin2, 0);

setMotorSpeed(\_rightMotorPin1, \_rightMotorPin2, 0);

}

void setMotorSpeed(int motorPin1, int motorPin2, int speed)

{

if(speed > 0)

{

analogWrite(motorPin1, speed);

analogWrite(motorPin2, 0);

}

else if(speed < 0)

{

analogWrite(motorPin1, 0);

analogWrite(motorPin2, -speed);

}

else

{

analogWrite(motorPin1, 0);

analogWrite(motorPin2, 0);

}

}

};

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ IR Sensor Class \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//

class IRSensor

{

private:

int \_irPin;

public:

IRSensor(int irPin)

{

\_irPin = irPin;

pinMode(\_irPin, INPUT);

}

bool detectWhiteLine()

{

return digitalRead(\_irPin) == LOW; // Assuming LOW means white line detected

}

};

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ultrasonic sensor positions \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//

enum SensorPosition\_t

{

FRONT\_LEFT,

FRONT\_MID,

FRONT\_RIGHT,

BACK,

LEFT,

RIGHT

};

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ultrasonic sensor objects \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//

Ultrasonic frontLeft(TRIG\_FRONT\_LEFT, ECHO\_FRONT\_LEFT);

Ultrasonic frontMid(TRIG\_FRONT\_MID, ECHO\_FRONT\_MID);

Ultrasonic frontRight(TRIG\_FRONT\_RIGHT, ECHO\_FRONT\_RIGHT);

Ultrasonic back(TRIG\_BACK, ECHO\_BACK);

Ultrasonic left(TRIG\_LEFT, ECHO\_LEFT);

Ultrasonic right(TRIG\_RIGHT, ECHO\_RIGHT);

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ IR sensor objects \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//

IRSensor irFront(IR\_FRONT\_PIN);

IRSensor irBack(IR\_BACK\_PIN);

IRSensor irLeft(IR\_LEFT\_PIN);

IRSensor irRight(IR\_RIGHT\_PIN);

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Motor object \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//

Motor motor(LEFT\_MOTOR\_P1, LEFT\_MOTOR\_P2, RIGHT\_MOTOR\_P1, RIGHT\_MOTOR\_P2);

float measureDistance(SensorPosition\_t SensorPosition)

{

switch (SensorPosition)

{

case FRONT\_LEFT:

return frontLeft.read();

case FRONT\_MID:

return frontMid.read();

case FRONT\_RIGHT:

return frontRight.read();

case BACK:

return back.read();

case LEFT:

return left.read();

case RIGHT:

return right.read();

default:

return -1;

}

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Interrupt Handlers Declaration \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//

void IRFrontISR();

void IRBackISR();

void IRLeftISR();

void IRRightISR();

void setup()

{

attachInterrupt(digitalPinToInterrupt(IR\_FRONT\_PIN), IRFrontISR, FALLING);

attachInterrupt(digitalPinToInterrupt(IR\_BACK\_PIN), IRBackISR, FALLING);

attachInterrupt(digitalPinToInterrupt(IR\_LEFT\_PIN), IRLeftISR, FALLING);

attachInterrupt(digitalPinToInterrupt(IR\_RIGHT\_PIN), IRRightISR, FALLING);

}

void loop()

{

float frontLeftDistance = measureDistance(FRONT\_LEFT);

delay(10);

float frontMidDistance = measureDistance(FRONT\_MID);

delay(10);

float frontRightDistance = measureDistance(FRONT\_RIGHT);

delay(10);

float backDistance = measureDistance(BACK);

delay(10);

float leftDistance = measureDistance(LEFT);

delay(10);

float rightDistance = measureDistance(RIGHT);

delay(10);

if(frontMidDistance < minDistance)

{

motor.moveForward(SPEED);

delay(10);

}

else if(frontLeftDistance < minDistance || leftDistance < minDistance)

{

while(frontMidDistance > minDistance)

{

motor.turnLeft(TURN\_SPEED);

delay(10);

frontMidDistance = measureDistance(FRONT\_MID);

}

}

else if(frontRightDistance < minDistance || backDistance < minDistance || rightDistance < minDistance)

{

while(frontMidDistance > minDistance)

{

motor.turnRight(TURN\_SPEED);

delay(10);

frontMidDistance = measureDistance(FRONT\_MID);

}

}

delay(10);

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Interrupt Handlers Definition \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//

void IRFrontISR()

{

if (irFront.detectWhiteLine())

{

motor.moveBackward(SPEED); // Move backward if white line is detected in front

delay(500);

motor.turnRight(TURN\_SPEED);

delay(250);

}

}

void IRBackISR()

{

if (irBack.detectWhiteLine())

{

motor.moveForward(SPEED); // Move forward if white line is detected in back

delay(500);

motor.turnLeft(TURN\_SPEED);

delay(250);

}

}

void IRLeftISR()

{

if (irLeft.detectWhiteLine())

{

motor.turnRight(TURN\_SPEED); // Turn right if white line is detected on the left

delay(500);

motor.moveForward(SPEED);

delay(500);

}

}

void IRRightISR()

{

if (irRight.detectWhiteLine())

{

motor.turnLeft(TURN\_SPEED); // Turn left if white line is detected on the right

delay(500);

motor.moveForward(SPEED);

delay(500);

}

}