190118KA

**A Quick Rundown of ‘Roomba’**

 I’ll be typing explanations in this font.

//start

 So Arduino uses a basic ‘void setup’ and ‘void loop’ structure. To make everything clearer, however, you also have the ability to write classes, functions for classes, and independent functions. I’m sure there are other things you can do as well, but this is what I’ll be using in the code below.

class Motor{

 private:

 int pin\_enable; //this is the thing that you set the speed with

 int pin\_forward; //basically each motor has two of these pins,

 int pin\_backward; //and you use them to reverse polarity and switch direction

 so above I begam creating the Motor class. Bear with me here, because I don’t even totally understand the syntax, but apparently this is how it’s done lol. The above variables are written under ‘private’ because only the Motor class will need to access them. The ‘pins’ are the physical pins on the motors that are used to control them… explanations are commented in above.

 public:

 Motor(int enable, int one, int two){

 pin\_enable = enable;

 pin\_forward = one;

 pin\_backward = two;

 //in the old code they essentially combined these last two steps with something fancy

 //oh well, this is easier to understand lol

 }

 So now I’ve begun writing the ‘public’ parts of the Class. I created this class so that I could create the two motors (I probably wouldn’t have done all of this if I had only one motor lol). Each motor has its own enable, ‘one’ and ‘two’ pins.

 void motorSpeed(int Speed){

 if(Speed > 0 || Speed == 0){ //here, I'm asking which way to set the polarity, with the idea being that I can input pos. or neg. speeds into the function

 digitalWrite(pin\_forward, HIGH);

 digitalWrite(pin\_backward, LOW);

 analogWrite(pin\_enable, Speed);

 }

 else{

 digitalWrite(pin\_forward, LOW);

 digitalWrite(pin\_backward, HIGH);

 analogWrite(pin\_enable, -Speed);

 }

 }

};

 Okay so what I did above is create a function that will take the speed I want to set (+ or -), and then appropriately adjust the polarity of each motor (this is what the ‘one’ and ‘two’ pins are used for). In this case, a (+) speed denotes ‘forward’ and a (-) speed denotes ‘backwards’. This is also useful for turning.

//Reverse Polarity protector is plugged into Pin 12 on the MEGA shield

 Ignore that. (It has to do with the physical wiring of the robot and I just noted it here so I didn’t use that pin for anything else).

Motor right\_motor(5, 42, 43);

Motor left\_motor(6, 44, 45);

 So I’ve just instantiated the two motors that I have on my robot. Phew!

char SIDE;

 I can’t remember what this does right off the bat, but I’m sure it’ll become clearer once we get to the part where it’s actually used.

int right\_speed;

int left\_speed;

 hooray! Two more integers that I’ve instantiated ☺

void setSpeeds(int bothSpeed){

 right\_motor.motorSpeed(bothSpeed);

 left\_motor.motorSpeed(bothSpeed);

 }

 Okay so a thing you can do with Arduino is create functions above or below the actual setup and loop (because they have to just not be included in the setup or loop). Because the rest of the Motor code is above, I’ve just created this function—which is *not* a part of the Motor Class—here. It basically sets both motors at the same speed (ie. Less work for me, and easier to tell when I’ve set the robot going straight forward).

void turn(int bothSpeed)

{

 right\_motor.motorSpeed(bothSpeed);

 left\_motor.motorSpeed(-bothSpeed);

}

 Simple enough: one wheel goes forward, and the other goes backwards. (Something I probably forgot to discuss earlier but may confuse you further now is that the motors polarities are reversed from each other to begin with—think about the way the motors are placed on the robot. If this still doesn’t make sense, feel free to ask.)

//now let's start working with the sensors

//set up the sensor class "Sensor"

class Sensor{

 private:

 int trigPin;

 int echoPin;

 long duration;

 int distance;

 …so now I’m creating another Class, this time for the sensors

void trigger(){

 //clears the trigpin

 digitalWrite(trigPin, LOW);

 delayMicroseconds(2);

 digitalWrite(trigPin, HIGH);

 delayMicroseconds(2);

 digitalWrite(trigPin, LOW);

 }

 So the sensors have a ‘trigpin’ and an ‘echopin’; I think the first sends out a wave, and the second ‘hears’ it (could be vice versa). Here, I was mostly going off previous documentation/code I found online, and these are the steps to get feedback from the sensors… and I guess this is a private function lol

public:

 Sensor(int trig, int echo){

 trigPin = trig;

 echoPin = echo;

 So now each sensor that I instantiate will get these two variables assigned—kind of important because the wiring/pin placement changes for each sensor… you’ll see what I mean when I actually instantiate them

pinMode(trigPin, OUTPUT);

 pinMode(echoPin, INPUT);

 }

 ^okay, so I was right about what they do

 int dist() {

 trigger(); //i literally floundered for the longest time BC I FORGOT THIS AAAAH

 ah, so that’s why the trigger class was private—because it’s called here, and *this* function is public

 //reads the echopin and returns the sound wave travel time in ms

 duration = pulseIn(echoPin, HIGH);

 //calculate the distance

 distance = duration\*.034/2;

return distance;

 }

};

 I put comments in the code, so this part should be self-explanatory…

//finished setting up the class

//now let's instantiate the actual sensors (yay!)

//it's (trig, echo)

Sensor bl(20, 21); //ie back right

Sensor bm(18, 19);

Sensor br(16, 17);

Sensor fl(36, 37);

Sensor fm(35, 34);

// Sensor fr(32, 33); //out of comission

 You’ll notice there’s no legit order to the way the pins are allocated… all depends on the physical spacing on the MEGA shield

void adjust(){

 brief interlude: you’ll notice this function is after I’ve finished setting up all the classes. In other words, it’s a way for me to take a huge section of the code out of the actual ‘void loop’, which makes it easier to understand what’s going on when you’re actually reading ‘void loop’. This code is *not* a part of any class.

int tooclose = 20;

int justright = 27;

int toofar = 50;

int goaway = 10;

 ^Those are pretty arbitrary values… also, they’re supposed to be in centimeters but it’s never quite right. Also to note: the sensors don’t do well below 10 centimeters—that’s where they start giving numbers in the thousands instead of single digits.

//since one of the front sensors is broken, I'm going to code this with the back

 And then I also end up ignoring the entire back because the lab is so crowded that the ‘back’ and ‘front’ commands interfere with each other.

if(bm.dist() > tooclose && bl.dist() > goaway && br.dist() > goaway)

{

 setSpeeds(-70);

}

else if (bm.dist() < tooclose)

{

 if(bl.dist() > br.dist() && bl.dist() > tooclose)

 {

 turn(-90);

 delay(250);

 }

 else if(br.dist() > tooclose)

 {

 turn(90);

 delay(250);

 }

 else if(bl.dist() < tooclose && br.dist() < tooclose)

 {

 setSpeeds(70);

 delay(400);

 }

}

}

 ... so that’s that. ☺

void readSpeed()

 {

 So this is a random function that lets you set the robot wheel speed based on Serial Monitor Input. It’s really only relevant for easily stopping the robot’s wheels (considering that the ‘adjust’ function has it moving forwards unless it hits something, ie the wheels are always running).

 char SIDE = Serial.read(); //read a character that will determine which motor speed to set in next if-loops

 int in = Serial.parseInt(); //now read the motor speed

 if (SIDE == 'R'){ //R designates the right motor

 right\_motor.motorSpeed(in);

 right\_speed = in;

 }

 else if (SIDE == 'L') { //L designates the left motor

 left\_motor.motorSpeed(in);

 left\_speed = in;

 }

 else if (SIDE == 'B'){

 setSpeeds(in);

 right\_speed = in;

 left\_speed = in;

 }

 Serial.print("Right: ");

 Serial.println(right\_speed, DEC); //print out the new speeds (ie. check that the correct one was changed)

 Serial.print("Left: ");

 Serial.println(left\_speed, DEC);

 Serial.println("--------------");

 delay(10);

 }\

 I think it’s explained well enough in the comments, lmk if you would like anything explained further.

void setup() {

 Serial.begin(9600);

 Serial.println("Starting up...");

 digitalWrite(40, LOW); //interruptPin

 So this is where I tried once more to figure out how attachInterrupt works but never really figured it out. The interrupt pin is something I did not end up using because of this, but for some reason I left it in the code…

}

void loop() {

 //this is the loop that checks Serial availability and then for motor speed inputs

 if (Serial.available() > 0) { //check that the Serial Monitor works

 }

 adjust();

 Serial.println("adjust!");

 //readSpeed();

 See how there are only like three functions in the loop! It’s so much nicer than having a whole bunch of code stacked in here will endless {}, etc. readSpeed is commented out because I couldn’t figure out how to run it in the background (common theme lol) and it is possible to kill the motors by just flipping the battery-power switch on the robot itself.

}

 If you have any more specific questions, please feel free to ask ☺.