ECE 110 Section: AB5

Mohammad Jaber and Raj Vinjamuri // 12-02-11

*Head TA: Vishnu Kamalnath*

**Objective:**

Our goal in this class is to electronically manipulate a motorized car to follow a white tape (taped on top of a black table) along smooth curves, zigzags, right turns, and straight lines. This car must also follow the right tape when passing over a grey colored split in the tape, and follow the left tape when passing over a white colored split in the tape. Finally, when the car reaches a large white block at the end of the track, the car must stop.

**Theory of Operation:**

For our car to navigate the track properly, we need six sensors (and their sensor bars), an Arduino Board, two CA modules, a working vehicle, and a Proto-Board (bread board) with wires.

IR Sensors:

The sensors are small rectangular prisms that shoot infrared light directly below them and measure the reflectance; the returned voltage value is used to determine whether they are above the white tape, grey tape, or the table. If they are above white tape, they output a voltage greater than 3.3 Volts. If they are above grey tape, they output a voltage between 1.5 and 3.3 Volts. If they are above the black table, they output a voltage less than 1.5 Volts. Note that these values are not arbitrary but ‘set’—that is, the sensors are calibrated to output these values for said scenarios

Arduino Board:

The Arduino Board is the most reliable equipment we can use in this project. It inputs the voltage readings from the sensors and outputs the voltages to the left and right CA Modules to power the left and right motors. The Arduino board is utilized via

CA Module:

Since our Arduino Board cannot give off large currents, we need to use CA Modules to amplify the current passing through the motors of the vehicle. CA Modules are transistors with their base connected to the Arduino, their emitter connected to the ground of the circuit, and their collector connected to the ground of their respective motor.

Proto-Board:

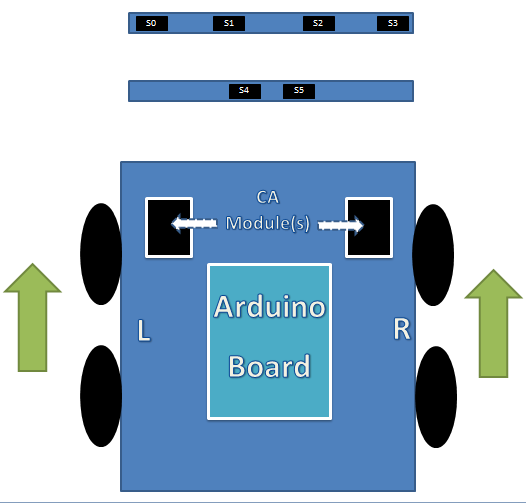
Our Proto-Board is an ancient bread board with banana plug inputs that are falling apart. We connect all the electrical components of our circuit together through the bread board, so it makes our lives much easier. We must be careful to make sure that the banana-plug extension and the board are both well-connected, since ours is partially broken.

Vehicle:

The vehicle is a heavy chunk of circuitry that is made of the following: a right motor (R), a left motor (L), a motor power source, a logic power source, a variable resistor to regulate current through the vehicle, 4 wheels, and a switch to turn the power sources on and off. The motor power source is at about 11 Volts, while the logic power source (powers Arduino and sensors) is at 5 Volts. While the vehicle tends to always run, the left and right motors don’t always power the left and right pair of wheels at the same speed/power. It is imperative that we make sure to adjust the variable resistor to provide just the right amount of current.

Layout:

The diagram below displays the layout of our car, including sensor placement and component placement on the Proto board.



**Our Decision-Making:**

While progressing throughout the lab we encountered chipsets and hardware we have had little to no experience with before. It was an enriching experience; however, this discovery of simple electronics was plagued by unexpected results and, at times, flawed hardware. Although our equipment was not always reliable, it was one do-able method to complete this project.

As ECE 110 is an introductory course into the school of Electronic and Computer Engineering, we thought using Arduino technology would be a fun way to complete the project. This would allow us to practice our basic programming skills and still have a functioning car. Furthermore, using code language instead of only hardware components was easier to adjust and fix on the fly—with the added reliability not provided with electrical parts provided. The thing that made coding a language instead of using logic gates most appealing was the fact that as a team of two, this would provide the most flexibility. That is, we could program on any computer with a basic text editor and collaborate online, instead of solely in the lab.

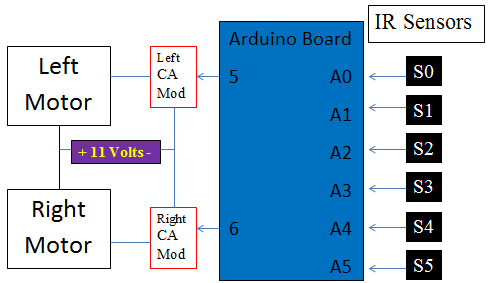
**Our Approach to Navigating the Track:**

We decided to control our car using the tape-avoiding scheme laid out in the lab manual. In this scheme, we have two main sensors which we label S1 and S2 in the diagram previously shown. If we place the car on the track such that the white tape is between the two sensors, then our goal is to make the car move forward while keeping the tape between these two sensors. Therefore, S1 and S2 are always trying to stay on the black of the table and avoid the tape (thus the term “tape-avoiding”). We have four other sensors to deal with stops, splits, and anything else the simple scheme cannot handle.

To build a circuit relating the state of the sensors to the left and right motors, we connected the sensor voltages as inputs to the Arduino board. Then we used the output voltages from the Arduino to regulate the currents passing through the CA modules. Since the CA modules are effectively in series with the motors, the output voltages from the Arduino thus regulate the currents passing through the motors.

Diagram of Circuit Hierarchy:

As displayed in this diagram, we connected each of our sensors to an analog input of the Arduino (each sensor has an arrow pointing to its corresponding Arduino pin). The Arduino outputs a Voltage to the left motor through pin 5 and to the right motor through pin 6.



*Note: The negative terminal of the “11 volts” is ground(ed).*

Programing the Arduino:

The Arduino board used the simple, yet effective, programming language of “C.” The following is an explanation of how our code works. We have included the code in an appendix at the end of the paper.

The Arduino board has multiple pin slots for both inputs and outputs. We utilize the A0 through A5 input pins for our analog sensors and two other pins that allow for PWM (variable) output—used to send a voltage to the CA modules. We also chose to use two LEDs—one installed in the Arduino board (pin 13) and another external LED (pin 12). These pins are all given designations corresponding numerically with their pin names (A0 – S0 and so on). This is done for ease in configuration and programming.

The analog sensor inputs are read using the ‘analogRead’ function. This function outputs a value between 0 and 1024. These values are divided by 1024 and multiplied by five to output a value equal to the voltage being given by each sensor. A variable X# (matched in the same fashion as the sensors: S#/A#/X#) is assigned for each voltage. This variable is what is referenced in our conditional statements.

After setting up that the pins 5/6 and the LEDs are outputs, we write functions that are used for the navigation of the track. A function called ‘navigate’ is the primary function that includes a simple tape-avoid function and right-angle turn function. The tape-avoiding scheme used involves independent motor control. When sensor S1 senses black, the left motor is activated, while when it senses white it receives power. The same goes for S2 and the right motor. The right-angle turns involve special situations where S0/S3 and S1/S2 are activated at the same time. For this, the corresponding motor is solely-activated to turn, while a ‘while’ statement keeps the turn going till S2/S1 sense white again, returning navigation to the regular scheme. The output to the motors is set using the variable ‘SPEED.’ This variable is set in such a way that on the lab day we can adjust the speed of the motors with only a change in one value, versus changing each and every output value. After the ‘navigate’ function are functions for controlling the motors in stop and split (both grey/right and white/left) scenarios. For a stop, both motors receive no voltage. For the split the right or left motor is turned on while the other is off. For white/left splits we activate the LED on the Arduino board, while for the grey/right split we activate an external LED.

The functions define what the car does, but thus far the code does not have logic telling *when* the car does what it is supposed to. This is where we used a loop. In the loop certain situations are used to call upon certain functions: This logic is done with “if, else, and if else” statements. ‘If’ any of our middle sensors sense grey, the right split function is utilized. ‘Else, if’ the two middle sensors sense non-black, and ‘if’ two outer sensors sense white, the stop function is used. If neither the right split nor the stop functions were called upon, then the situation must be a left split, as that is the only other time both the inner sensors sense white in our scheme. When none of the above occurs, which most of the time it does not, we call upon the ‘navigate’ function.

Throughout the code, delays were implemented to pause the code for a designated number of milliseconds in order to continue carrying out the previously made command.

Essentially, that is our code in paragraph form. Below this functioning code we include a change log—a tool used when multiple programmers alter the same code.

**Challenges Faced:**

This lab took almost a month, and with reason. The navigation was simple, yet very punishing.

As previously mentioned, hardware failed to always be reliable. One help to this was the fact that ECE 110 lab was given new CA modules that were more reliable and consistent. Aside from this individual problems, we also encountered the following:

1. Proto board not working in certain (unmarked) places.
2. Dysfunctional sensors.
3. Non-parallel-running motors.
4. Fast battery drainage
5. Reflection from the overhead lighting – messed with IR sensors

By looking at the change log in our code it is apparent that many alterations were made to what ended up being a fairly simple code. This was due to the constant changing of values for PWM outputs, and other values. Our greatest challenged lies with the split. Once our Arduino code was up and running, we were successfully able to navigate most turns, and upon adjustment, navigate all but the splits and right-angle turns. Addition of an emergency or right-angle specific function-logic accounted for the sharp turns, but our split remained a problem. In fact, for the longest time our code ‘ignored’ splits altogether and merely continued a tape-avoiding navigation—ending up with a 50/50 success rate of choosing the correct split. In our most recent version of the code, we feel like we have addressed the issue of the car recognizing a split via altering the conditions of the loop statements.

**Lab Contribution to Overall Knowledge and Final Design:**

With the Arduino, we essentially started from scratch on how to navigate the track. Aside from the concepts on what to do and when (i.e. definition of tape-avoiding circuit) we utilized a quick lesson on C programming for Arduino to make the end product design.

On the other hand, the lab was a very interesting and successful investigation of the components and hardware that constitute the majority of electronics in today’s technology. We learned quite a bit with the labs as they coincided with the lectures being taught on the same topics/parts.

**Conclusion:**

After almost a month of working on this final project, minding a week of Thanksgiving Break, we have—as a team of two—come up with a final design that we confidently will use. We expect our car to navigate the entire track to entitle us to full points. In the event that something wrong happens, we have made certain lines of code to still help us get points (i.e split detection via the LEDs).

Appendix (CODE):

//------------------------Special Considerations-------------------------------

/\*

1. Make sure sensors/sensor bar/battery/ground/etc is all wired correctly with reliable connections.

2. Verify that output voltages from sensors complies with 1.5 "gray" threshold and 3.3 "white" threshold.

\*/

//------------------------Setup------------------------------------------------

//inputs

#define S0 A0

#define S1 A1

#define S2 A2

#define S3 A3

#define S4 A4

#define S5 A5

//outputs

#define L 5//Motor

#define R 6//Motor

#define LED1 13//Split detection - ARDUINO BOARD LED

#define LED2 12//Color Detection - EXTERNAL LED

int SPEED = 45;//the PWM value that is sent to motors -- via CA module -- when they are "on"

//Math Functions that convert analogRead numbers to recognizable voltages

float X0 = (analogRead(S0)/1024.0)\*5.0;

float X1 = (analogRead(S1)/1024.0)\*5.0;

float X2 = (analogRead(S2)/1024.0)\*5.0;

float X3 = (analogRead(S3)/1024.0)\*5.0;

float X4 = (analogRead(S4)/1024.0)\*5.0;

float X5 = (analogRead(S5)/1024.0)\*5.0;

void setup(){

pinMode(L, OUTPUT);

pinMode(R, OUTPUT);

pinMode(LED1, OUTPUT);

pinMode(LED2, OUTPUT);

}

//------------------------Car Movement Functions-------------------------------

void navigate(){

//TAPE AVOIDING NAVIGATION

if(X1 < 3.3) {analogWrite(L, SPEED);}//Left Motor Control

else {analogWrite(L, 0);}

if(X2 < 3.3) {analogWrite(R, SPEED);}//Right Motor Control

else {analogWrite(R, 0);}

//RIGHT ANGLE NAVIGATION

//Left Sharp

if(X0 > 3.3 && X1 > 3.3){

analogWrite(R, SPEED);

digitalWrite(L, LOW);

delay(300);//makes sure that the turn starts

while(digitalRead(S2) == LOW){

analogWrite(R, SPEED);

digitalWrite(L, LOW);

}

}

////Right Sharp

if(X2 > 3.3 && X3 > 3.3){

analogWrite(L, SPEED);

digitalWrite(R, LOW);

delay(300);//makes sure that the turn is complete

while(digitalRead(S1) == LOW){

analogWrite(L, SPEED);

digitalWrite(R, LOW);

}

}

}

void endStop(){

digitalWrite(R, LOW);

digitalWrite(L, LOW);

}

void leftSplit(){//White Split - Left

analogWrite(R, SPEED);

digitalWrite(L, LOW);

digitalWrite(LED1, HIGH); //Turn on white Split LED

delay(300);//makes sure that the turn is complete

digitalWrite(LED1, LOW); //Turn off white Split LED

}

void rightSplit(){//Grey Split - Right

digitalWrite(R, LOW);

analogWrite(L, SPEED);

digitalWrite(LED2, HIGH); //Turn on Grey Split LED

delay(300);//makes sure that the turn is complete

digitalWrite(LED2, LOW); //Turn off Grey Split LED

}

//------------------------Commands---------------------------------------------

void loop(){

//SPLIT/STOP

//Grey = Right Turn (Arduino AND External LED Activated)

if ((X1 > 1.5 && X1 < 3.3) || (X2 > 1.5 && X2 < 3.3) || (X4 > 1.5 && X4 < 3.3) || (X5 > 1.5 && X5 < 3.3)){//Any sensor that would be over the split detects grey

rightSplit();

}

else if (X1 > 1.5 && X2 > 1.5){//Both inside sensors detect white => Only split or stop scenarios

//Stop

if (X0 > 3.3 && X3 > 3.3){//all middle sensors see white

endStop();

}

else{

leftSplit();//If not stop and not right split, and inside sensors are on means left split

}

}

//TAPE AVOIDING SCHEME

else{

navigate();

}

}

//code end

/\*------------------------Change Log:------------------------------------------

1.0 Straight movement, turns, and stop capability

NO DELAYS INCLUDED; NO CODE FOR SPLIT ACTION; NO DEBUGGING CODE

1.1 Initial values for outputs

Math functions (not used in code)

Changed 'int' to 'float' to allow for decimal numbers

Sensor voltages printed twice a second ( for debugging)

1.2 Changed initial L/R values to be set by digitalWrite(L/R, LOW);

Replaced tape avoiding logic with v.Raj from v.Mohammed

Updated voltages with real voltage values (post-calculation)

Initialze print

Fix print to Serial.print

Removed pinMode for analog pins

Removed extra "learning" code from lecture

1.4 Add print for special cases

Simplified printing

Fixed correct turning scheme

1.5 Split fix attempt

Split simplification (robust+)

Added delay in stop function

Math debugging check/print

1.6 Remove initial voltage outputs

Remove print/debugging functions

Remove extra delays

Reduce PWM output to 45

Change most digitalOut to analogOut

Remove some comments for simplicity

1.6.5 Simplify split logic

adjust stop delay

1.7 Emergency Turns

More definate stop

Add another grey turn

1.7.5 Combine grey split code

Add comments - few

1.8 Fix split bug and make split more efficient

remove stop delay and make stop refer to front row only

1.8.5 Add LED output for split and color detection

Added delays to emergency turns and split turns to ensure complete turn

Make emergency turn faster than other functions

Add 'Special Considerations'

Format mods for programming ease

1.8.5fn Combine functions from 1.9 while keeping scenarios

Simplified output PWM with a "SPEED" integer.

Simplify emergency/right-angle turns scenario

1.8.5fnb Changed TA scheme to more simple/robust scheme as given in lab instruction

Initiated this new navigation scheme

Reformatted and updated comments for ease of reading and to shorten length of code

1.8.6 Cleaned and compiled code from v1.8.5bfunction

Replaced any "analogWrite(X, 0)" with "digitalWrite(x, LOW)"

1.8.7 Made Split/Stop loop directions more efficient

Added conditions to stop to avoid any accidental situations

2.0 Verified code and successfully compiled

\*/