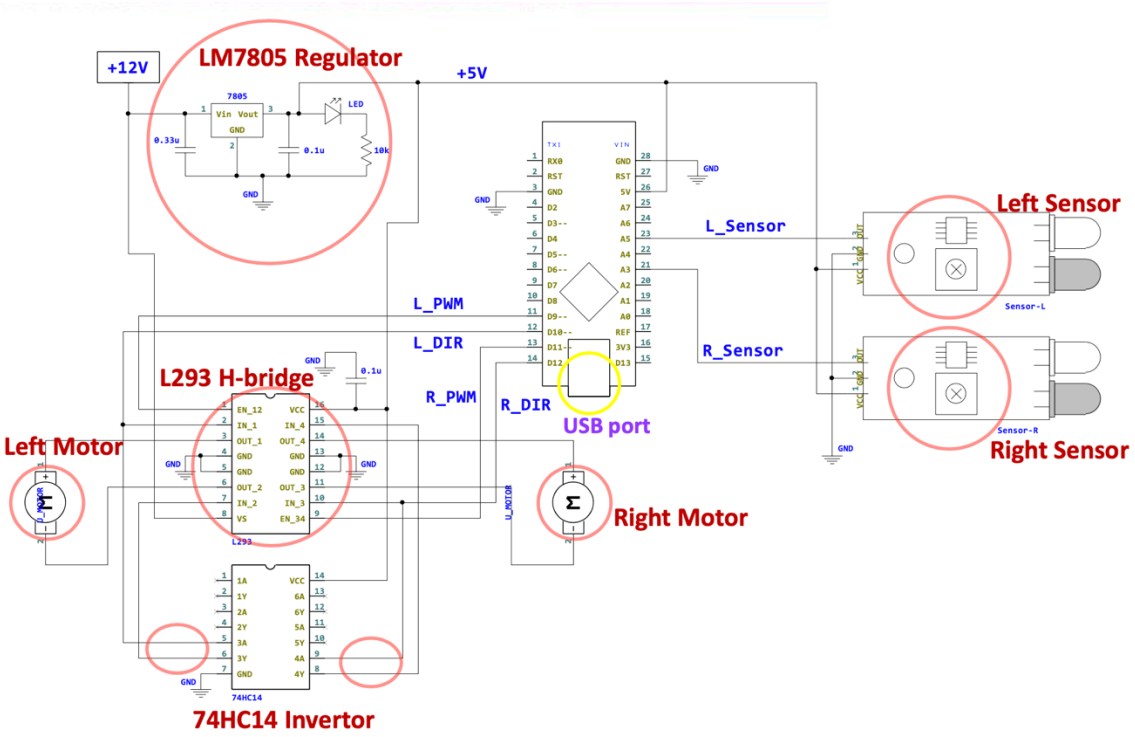
ELEC1100 Project Report – Francisco Heshiki de las Casas

Introduction:

In the course ELEC1100, our team was given a project which consisted of designing and constructing a robot car capable of driving autonomously and navigating a complex track. The car would have to guide itself with sensors along a white track with black background and had to stay on the path while performing certain actions along the way. During the initial stages of this course, we delved into the intricacies of circuits and logic design in order to build the foundational knowledge required for implementing our very own robot car. We also had to learn about integrating an Arduino board and the code behind it to manage all the various inputs and outputs required to make the car move. Finally, our team had to begin building and designing our own circuit board.

The robot car can be separated into three main sections. The structure of the car housed all the necessary components and allowed for the car to move efficiently. The circuit board made sure all the components were interconnected and the proper instructions and power was going to the right places. Finally, the "brain" of the robot car, the Arduino board, had to be coded to handle all the logic and controls and was directly connected to the circuit board.

As the first step in the project, the circuit board had to be completed first as it was the main focus of the course and crucial to the other sections. The circuit board was wired by hand complete with an Arduino and various electrical components like transistors, sensors, motors, microchips, and some other lower-level components. This was all integrated onto a breadboard that would eventually attach to the top of our robot car.



COMPLETE CIRCUIT DESIGN

The second step in the building of the robot car, our team had to build the physical structure of the robot car. The structure consists of two hexagonal plates attached to each other by some spacers. In this internal space was the two motors that controlled the wheels and the battery. On the top of the robot car was attached the breadboard which connected the other components together. On the bottom and outsides of the robot car we attached three sensors for detecting white or black, two wheels, and one swivel in the back for easier maneuverability.

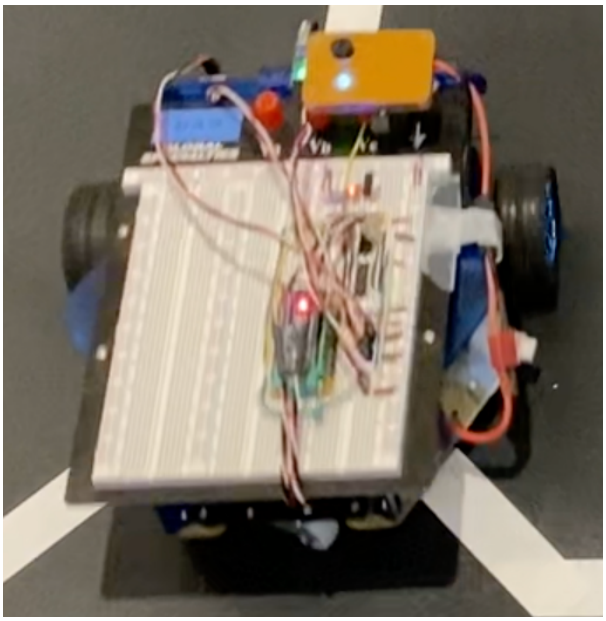
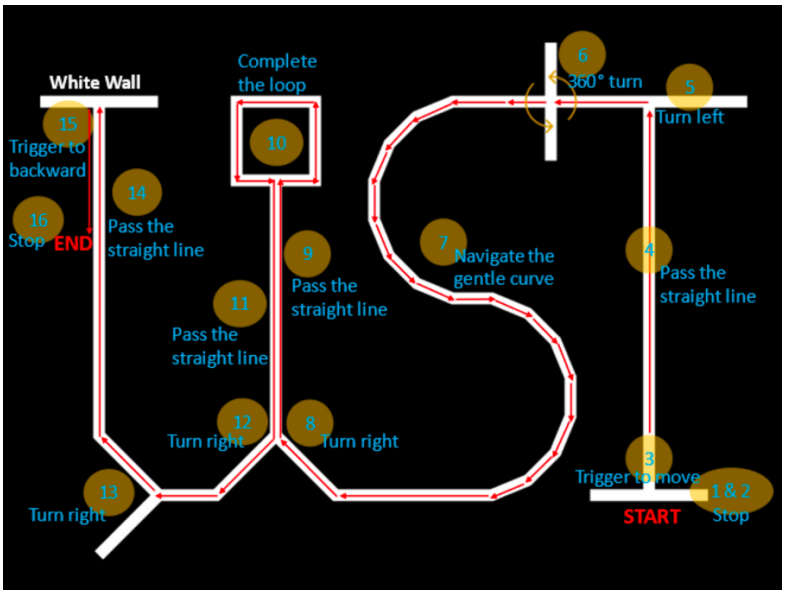


IMAGE OF OUR ROBOT CAR

Finally, the last and most important step was to write code for the Arduino board. This was crucial for the robot car to work since the car couldn't make any decisions without the logic implemented. The code had to manage various inputs and outputs and also have some decision making behind it to make sure the car could complete the track. Logic had to be implemented how much power the motor needed, the direction of the motors, the decisions for each sensor variation, and more. A major hurdle in this step was designing the logic for the different "event" points in the track since specific actions needed to happen at certain points.



ROBOT CAR TRACK

As you can see from the image above, it wasn't as easy as just telling the car to follow the white line. There are many points along the track that would disturb this logic and cause the robot car to fail at its task. Therefore, the code had to be written in an efficient and smart manner as to avoid any wrong decisions taken by the car. A detailed and well-planned implementation had to be created.

Logic Design:

Before discussing the actual code, we need to know how the Arduino integrates with the circuit board and then the physical components. The Arduino has various physical pins that are required for the I/O and power supply. Each input and output pin from the physical components like the sensors and motors needs to be connected to the Arduino for proper handling. The careful linkage of each input and output pin ensures the proper functioning and coordinated operation of the entire system. Additionally, a clear comprehension of these interconnections lays the foundation for effective code implementation, enabling precise control and communication between the Arduino and the various elements of the robot car.

First, each input and output pins from the Arduino board had to be defined in the code. These are constant variable that won't change throughout the code but is very important for the functionality of the robot car. Three input constants had to be set for each pin associated with each individual sensor (two at the front and one at the bumper. An additional four output constants are needed for the left and right motor's speed and direction.

Secondly, each pin needs to be set by the Arduino function "pinMode" as either an input or an output and wrapped into the "setup" function. The "setup" function executes only once at power up of the Arduino board and shouldn't contain any logic besides the initialization phase.

Finally, the main section of the code is the "loop" function. The "loop" function repeatedly executes and quickly performs instructions. Instructions will be sent to the pins of the individual components by various functions handled by the Arduino. There are three main functions being used to handle read/write operations:

- digitalRead(): used to read the value returned by each sensor
- digitalWrite(): used for controlling the direction of the motors. "1" for forward and "0" for backwards.
- analogWrite(): used for adjusting the speed of the motors (how much power should be given to each motor). This value was a constant of 200 was used for our robot car as any less would be too slow to compete the track (more details covered later).

The initial code began with a basic structure which took the three sensors in the front as input and output instructions to the wheels. The two sensors on the front were positioned so that if the car was straying to the left/right then a sensor would detect a white line. Also, to begin moving and tracking the white line, the bumper sensor needed to be activated by waving a white surface in front of it. For the code to know if the bumper had been activated, the variable "countBumper" was added and would increment each time this happened. Additionally, for ease of reading, in the logic table a "1" for the sensors means it detecting white while in the code it's the other way around.

INITIAL LOGIC TABLE

INPUTS				OUTPUTS			
B_Sensor	L_Sensor	R_Sensor	countBumper	L_PWM	R_PWM	L_DIR	R_DIR
0	0	0	0	0	0	0	0
1	0	0	0	200	200	1	1
0	0	0	1	200	200	1	1
0	0	1	1	200	200	1	0
0	1	1	1	200	200	?	?
0	1	0	1	200	200	0	1
1	1	1	1	200	200	?	?

In the logic table above, we left “B_Sensor” as “0” for most entries since the bumper is only triggered during the initializing phase. Also, when both the left sensor and right sensor are “1” the direction of the motors is unknown. This is because no logic for handling this scenario has been added yet so the robot car doesn’t know if it should go left or right. By inspecting the track, our team determined that each time the car encounters a scenario where both left and right sensors are “1” means that a certain action or “event” needs to take place. Therefore, we added the variable “countEvent”. This variable would add 1 each time this scenario occurred and would perform the action corresponding to the given “countEvent” value.

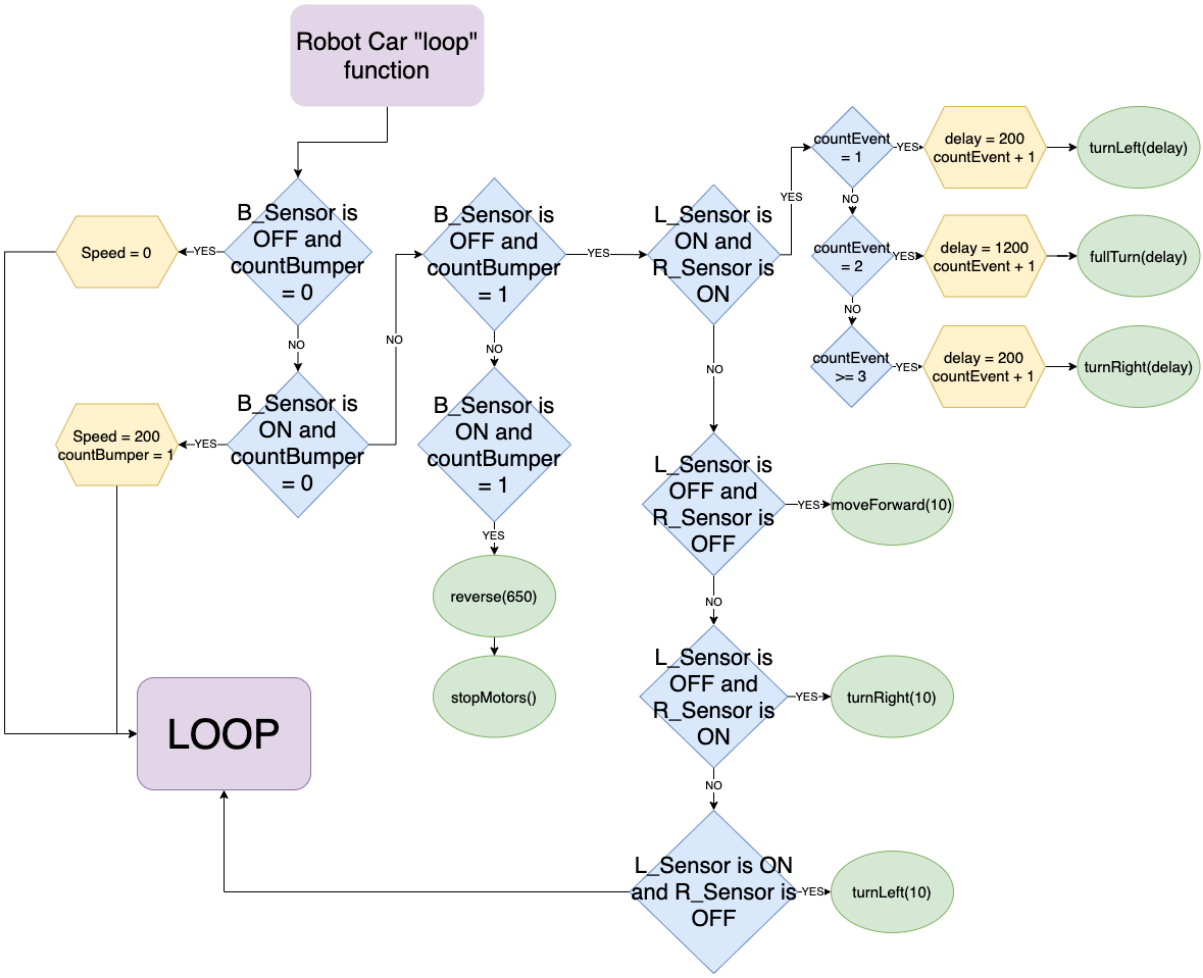
The first event the robot car would encounter is a left turn. The second event is a 360 degree turn in-place. For the next three events, we found out a simple right turn would work. This even applied for the loop event as the main logic could handle the rest after the right turn. The final event was to make the car go in reverse when it encounters the wall at the end of the track. However, the “countEvent” variable wasn’t needed In this case because “countBumper” would increment to two when it meets the wall and we can do this with a simple “if” statement.

Finally, there was a lot of testing required to find the optimal speed or PWM of the car. The values can range from 0 to 255 with 0 being the motors not spinning at all. However, any value below 100 wasn’t big enough to move the car at all due to the weight of it. Also, from the testing phase, any value below 200 was cutting it too close to the 20 second time limit. Therefore, we found that 200 was the right speed for our robot car as it would be fast enough to meet the time limit without accidently running of the track. There also needed to be a certain “delay” for certain actions so that the robot car was able complete it before the main logic started to repeat in the loop function. After various rounds of testing, we determined the right amount of delay for the right speed. Below is the full logic table and logic-flow chart.

FINAL LOGIC TABLE

B_Sensor	L_Sensor	R_Sensor	countBumper	countEvent	L_PWM	R_PWM	L_DIR	R_DIR	Delay
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	200	200	1	1	0
0	0	0	1	0	200	200	1	1	0
0	0	1	1	0	200	200	1	0	10
0	1	0	1	0	200	200	0	1	10
0	1	1	1	1	200	200	0	1	200
0	1	1	1	2	200	200	0	1	1200
0	1	1	1	>=3	200	200	1	0	200
1	1	1	1	>3	200	200	0	0	650

LOGIC-FLOW CHART



Debugging Report:

Although we didn't encounter any issues with the wiring of the circuit or components there were still many problems and bugs with the code. The first bug occurred when trying to implement our "countEvent" variable. Due to the rapid repetitions of the loop function of the code, the car would think it encountered another event and continuously increment countEvent without there being any changes to the left and right sensors. Essentially, the car would think it was at a different event and perform the wrong action. We attempted many things to fix this bug like adding delays, adding more logic statements, and changing the conditionals. However, we realized that this only happened after the second event and the other events could be handled with a simple right turn and a specific delay time. Basically, the bug was fixed by removing redundant functionality and logic that we thought the robot car needed.

The second bug we encountered occurred when the robot car had to complete the square loop. Once it performed the initial right turn into the loop the car would often get confused and lose track of the white line. At first, we thought the car was going too fast and wasn't able to make the sharp turns in time. Our first attempt at fixing this was to manually time the distances and "fake" the loop by hard coding the robot car to complete the square movement. These results weren't satisfactory, so we decided to try slowing the car down. This led us to discover

that the cars sensors were detecting an event in the middle of the loop and triggering a right turn. Ironically, we scrapped all the changes to the code and simply separated the sensors further apart from each other on the car which worked out in the end.

Results & Conclusion:

The strong point of our robot car was the good design of our code and well-structured logic. However, our biggest flaw and the source of our hardest bug to fix was the physical car itself. The build could have been better, and the placement of the sensors and components resulted to be more important than initially thought. The code was well structed and understandable and this made it increasingly easier to test and modify as time went on. Although, it took a lot of work to get to that stage and many iterations occurred before the final product.

During the whole process there were many points that could be improved. For example, the logic behind the robot car was significantly easier to implement than originally thought and a large amount of time and effort could have been saved. We should have started with a simple implementation and testing phase instead of trying to do everything at once. If we could improve the design, we would have added more components like sensors or structural pieces to make the car run smoother.

In conclusion, the results from this project were more than satisfactory. Our robot car performed very well in the final demo needing only one attempt to cover all the points on the track. Although, there were many aspects of the project that could have been improved the final outcome was a great learning experience.