Advanced Mechatronics

ROB-GY-6103

Project 2 - Propeller Automated Guided Vehicle

Team Members:

Rishabh Sivarajan(rs8426)

Kelvin Maliyakal(kpm8503)

Manan Malpani (mm11787)

Professor:

Dr. Vikram Kapila

Abstract

The Propeller Project involves the design and implementation of an automated guided vehicle (AGV) that can follow a path in a factory to pick up widgets made by machines in lane A and drop them off for further processing at machine locations in lane B. The robot must follow all lane rules, avoid obstacles, and reach specified locations. The AGV is expected to detect intersections, obstacles, and widget locations and indicate their detection through some display medium. It must also slow down in lanes A and B, check for dynamic obstacles, and calculate and display the distance traveled from the pickup and drop-off locations. Infrared and Ultrasonic sensors were used to implement the requirements and all these tasks were run on the propeller microcontroller. The AGV was able to perform all the required tasks and used an RGB LED to indicate various criteria with different colors and display the distance covered between the pickup and drop-off locations towards the end.

Introduction & Project Background

The manufacturing sector has undergone a transformation thanks to the rise of automation and robotics, which has improved productivity, precision, and efficiency. Designing and creating an Automated Guided Vehicle (AGV) has become an essential part of many industrial processes in this setting. The goal of this project is to build an AGV that can move through a manufacturing floor, pick up widgets produced by machines in **lane A**, and deliver them to certain machine locations in **lane B** for additional processing. The AGV must follow all lane regulations and have the ability to recognize and steer clear of any obstructions. With the help of this project, we hope to develop a durable and dependable AGV that will increase production and manufacturing efficiency.

The Propeller board, a potent microcontroller that gives us the characteristics and capabilities we need to construct an advanced AGV, will be used to carry out this work. Compared to other microcontrollers, such as the well-known Arduino, the Propeller board has a number of benefits.

First off, the **eight cores** on the Propeller board enable true **parallel computing**. In our AGV project, where we must process data from several sensors while also controlling the motors and communicating with other components, this means that we can run multiple jobs concurrently. The Arduino, in contrast, only has a single core, which restricts its ability to multitask.

Second, there are numerous **built-in peripherals** on the Propeller board, including quadrature decoders, PWM generators, and high-speed serial connections. We can simply interface with different sensors and actuators using these peripherals, and we can quickly process data. In comparison, the Arduino needs extra shields or modules to perform comparable functions.

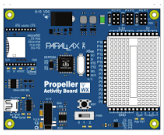
Finally, the Propeller board's extensive configuration and customization capabilities let us modify it to meet our unique requirements. Its large function and object library makes it simple to create sophisticated applications rapidly. It can also be fully programmed in the popular and potent computer language C/C++.

Apparatus

The project required us to build an AGV, and some of the parts were sourced from the previous Mechatronics project. Additionally, a Propeller activity board was purchased and the remaining components were taken from an Arduino Kit.

Bill of Materials(BOM)

Serial No.	Part Name	Quantity	Part Image	Unit Cost($)	Total Cost($)
1	Parallax Screwdriver	1		1.5	1.5
2	Resistor(220Ω)	3		0.33	0.99
3	RGB LED	1		2	2
4	IR Sensor	2		0.6	1.2
5	Ultrasonic Sensor	2		29.71	59.42
6	Machine screws, Phillips	2		0.25	0.5
7	Hex Nuts	4		0.14	0.56
8	Tail wheel ball	1		3.95	3.95
9	Boe-Bot aluminum chassis	1		29.99	29.99

10	Battery holder with cable and barrel plug	1		4.99	4.99
11	Cotter Pin	1		0.3	0.3
12	Plastic wheels	2		2	4
13	Continuous rotation servo	2		17.95	35.9
14	AA Batteries	4		1.33	5.32
15	Propeller Activity Board	1		82.22	82.22
16	Breadboard mini	1		3.95	3.95
17	Data Cable	1		7.99	7.99
18	Jumper Cables	32		0.06	1.92
19	LCD Display	1		34.95	35.95

BOM: Table 1

Total cost: $281.65

Design Layout

In this design layout, the robot is equipped with two infrared (IR) sensors and an Ultrasonic sensor in the front, there is another ultrasonic sensor mounted on the left of the robot.

The IR sensors are used to recognize and follow a black line painted on a white surface. IR sensors produce infrared light and observe how it is reflected off the ground. The IR sensor transmits a signal to the robot's microprocessor, which modifies the movement of the robot to stay on the line when it notices a change in the reflection pattern, such as when it crosses the line.

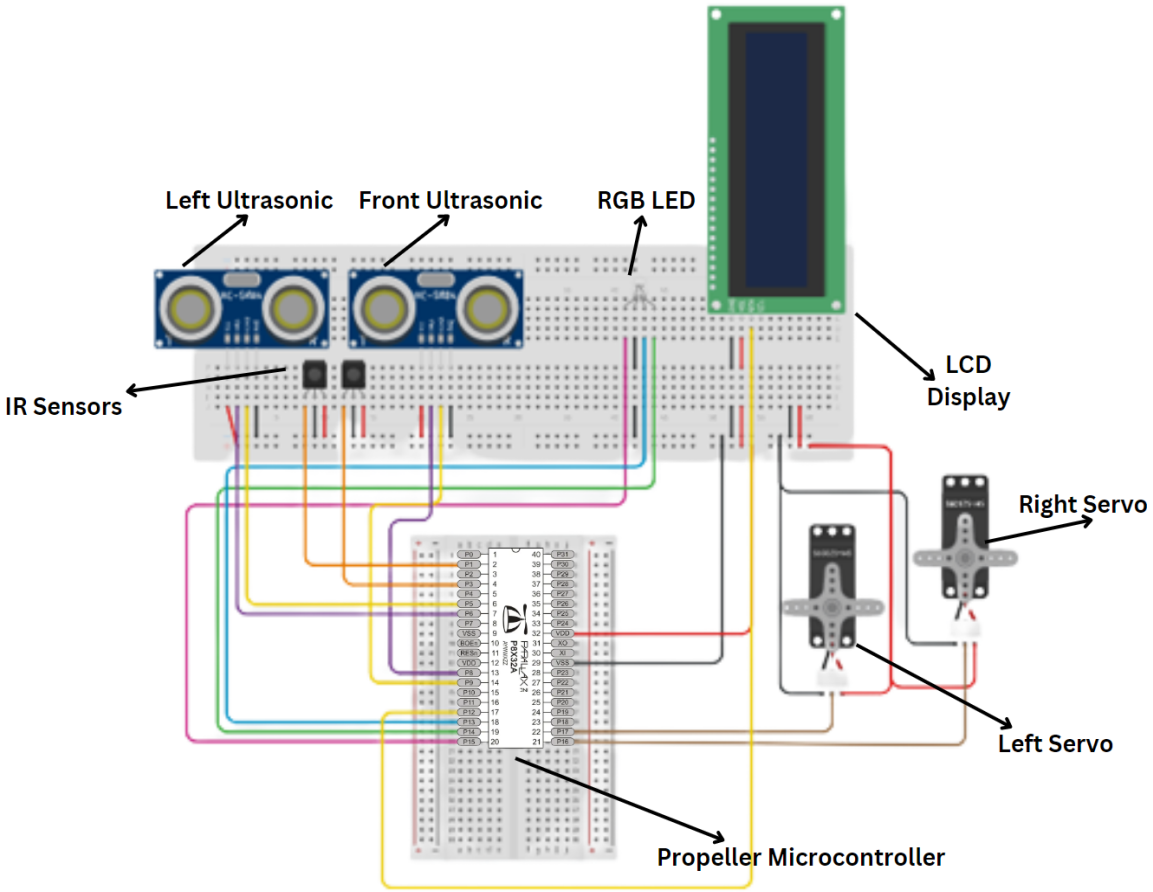
The ultrasonic sensor is used to identify any obstacles in front of the robot, while the robot is in the primary lane the ultrasonic sensor will prompt the robot to reroute if there is an obstacle in front of it. The same ultrasonic sensor is used to identify dynamic obstacles in lanes A and B and stop the robot till the obstacle has been removed. The Ultrasonic sensor mounted on the left side of the robot is used to identify the partially processed widget in lane A for pick up and drop it off at an assigned location in lane B.

The robot needs to move at a visibly faster speed in the primary lane and slow down when it goes into lane A or B.

An RGB LED is used to indicate the junctions (Blue), obstacles(red), pick-up (cyan), and drop-off(green). To indicate the distance between the pick-up and drop-off locations as a factor of 40, the RGB LED blinks Y number(violet) of times and then prints the distance on the LCD in centimeters (40 × Y).

Y = (Drop off distance - Pickup distance) / 40

Circuit diagram



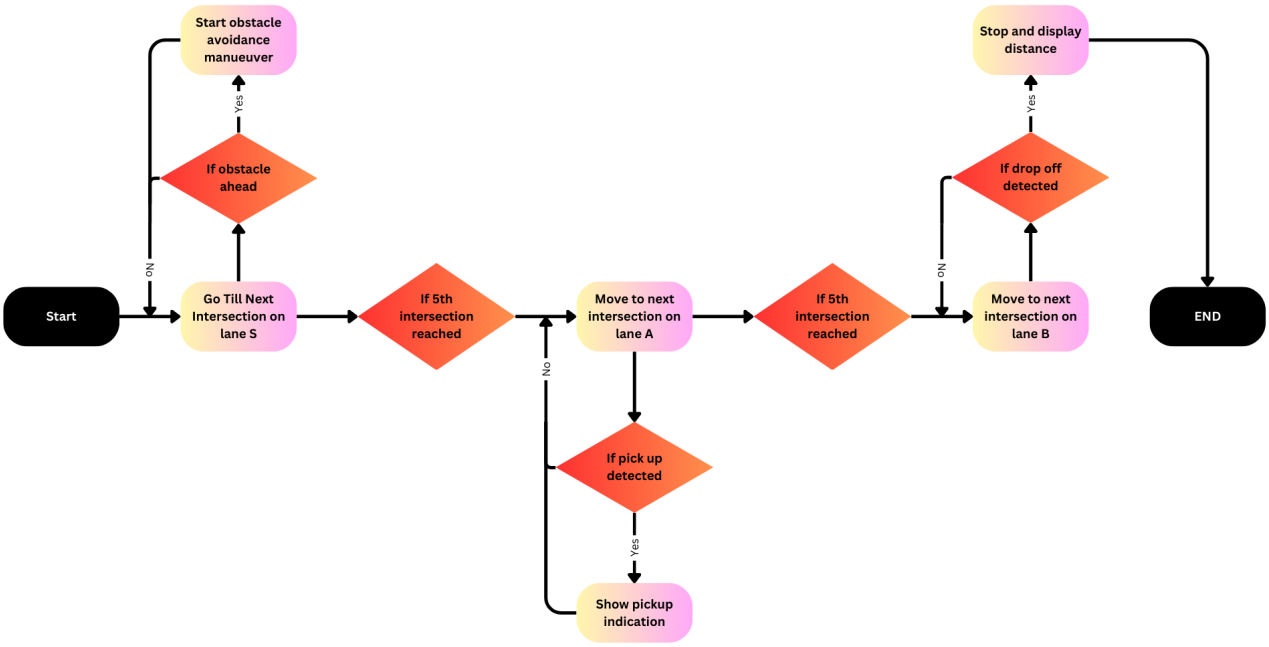
Power

Ground

Approach

1. Initiate forward motion from the starting point.
2. Check for Obstacles when the robot is at a junction on the primary lane, and blink the blue LED when it identifies a junction.
    2.1. If no obstacles are detected the robot must travel straight to junction i5 and turn right into lane A.
    2.2. If an obstacle is detected.
        2.2.1. Turn on the red LED.
        2.2.2. Turn left, travel to the junction to lane B, then turn right and use the distance from the front ultrasonic to halt when dynamic obstacles are placed while moving forward (must maintain a slower speed than in the primary lane) till the next intersection, and finally turn right at the intersection followed by a forward motion to make the robot reach the primary lane again.
        2.2.3. At this point, the robot will turn left and repeat step 2 until it reaches junction i5.
3. At i5 the robot must turn right, move forward till the intersection, and turn right into lane A.
4. Once in lane A, the robot must check for an object at each intersection.
    4.1. The robot starts using the distances from the front and left ultrasonic sensors while doing the line following.
    4.2. The front ultrasonic sensor is constantly checking for any dynamic obstacles and will stop if an obstacle is within a 10 cm range.
    4.3. If the left-ultrasonic sensor identifies an object, the robot pauses for 2 seconds and blinks the cyan-colored LED(pick-up) while making note of the intersections left to reach intersection A1.
    4.4. Once an object has been identified the robot will drive to intersection A1 and turn right while following step 4.2.
    4.5. The robot will then travel down the bi-directional lane till it reaches the intersection of lane B.
5. Once the robot enters lane B, it turns right.
    5.1. The robot will use the distance from the front and left IR ultrasonic sensors..
    5.2. The front IR sensor is constantly checking for any dynamic obstacles and will stop if an obstacle is within a 10 cm range.
    5.3. If the left-ultrasonic sensor identifies an object, the robot pauses for 2 seconds and blinks the green-colored LED(drop-off), and uses the intersections crossed until the drop-off location and uses the previous value calculated in 4.3 for approximating the total distance traveled between pickup and drop off.

Workflow



Method of Approach and Testing the Robot

- Testing Servo header file and its functions, testing and tuning motor speeds and direction after mounting
- Testing the two IR's separately in while loop in int main
- Testing IR's by assigning it to a separate COG to run it in parallel with the rest of the main code
- Testing Ultrasonic separately in while loop in int main
- Testing IR by assigning it to a separate COG to run it in parallel with the rest of the main code
- Testing the output of the IR's and ultrasonic together in int main after running them in separate COGs to make them function irrespective of what is going on in the main code
- Writing code and testing function for line following based on output from the IR's that will directly be used to condition the speed of the servos
- Writing and testing function for left turn
- Writing and testing function for a right turn
- Integrating the motion, right and left turn with the output of the front ultrasonic sensor for object detection and then maneuvering using the motion, right and left turn functions
- Including the intersection count to make it stop at the 5th intersection
- Writing and testing a separate function for line following in lane b for maneuvering obstacle in the center lane for halting until an obstacle is blocking the way and slowing down the speed of the line following

- Testing an extra ultrasonic sensor placed on the left of the robot and running it in a separate cog parallelly to test its output
- Writing and testing function for lane A where an object has to be picked up which will be detected using the ultrasonic sensor placed on the left side of the robot all the while doing the obstacle detection and halting in the front using the front ultrasonic sensor and line following.
- Testing if the robot stops at end of Lane A
- Writing extra lines of code for making the robot reach Lane B to drop off
- Reusing the function used for Lane A with some changes in the variables and getting it ready for drop off and testing
- Testing intersection detection using tri-color LED using a separate function that will always run in the COG parallelly
- Updating the earlier intersection detection LED function for all kinds of detection like obstacle detection, pickup detection and drop off detection with different colors
- Tried multiple methods for intersection counts between pickup and drop off locations for displaying the distance on the LCD

Problems Faced

1) **Blind right and left turns to calculated turning:**

To address this issue, a technique was implemented to monitor the number of changes detected by an IR sensor located at the front of the robot. Specifically, when the sensor detects two changes in its input signal, it indicates that a turn has been completed, whether it be a right or left turn. To differentiate between a right and left turn, the front right IR and front left IR sensors were used to track the direction of the turn. This method allows the robot to accurately detect and complete turns during its operation.

2) **Distance approximation between pickup and drop locations:**

The initial idea was to calculate the total distance using the total number of intersections detected along the way between pickup and drop and then display it as an approximation of the distance. If ‘n’ is the total number of intersections detected, then the distance would be 40 x n. The simple code line we implemented initially was

```
if (( left_ir == 1) && (right_ir == 1))
{
intersection = intersection + 1;
}
```

But the problem with this approach is that in the code that was implemented, the reading of the IR’s always run on a separate core. So, what happens is this piece of code keeps getting true for as long as the IRs are traversing through the thickness of the black line. This obviously gave inaccurate readings as high as 300 to 400 on the intersection counts. To counter this, we added another condition within this

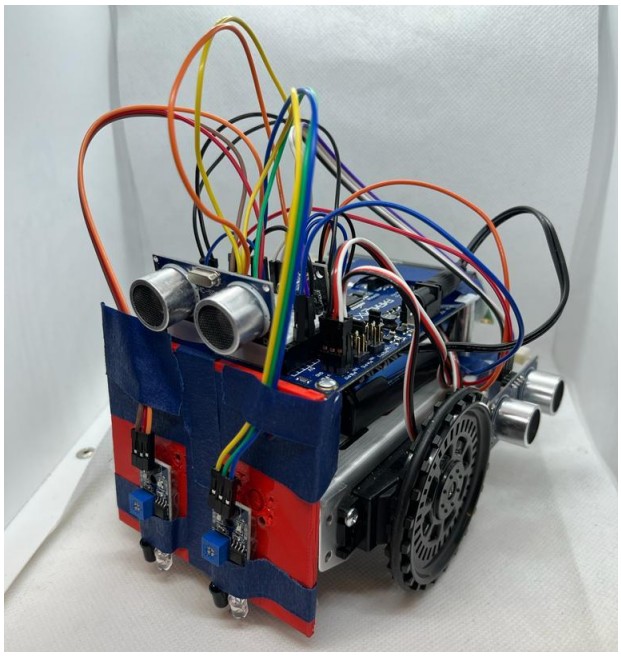
conditional statement which only went true if the last state of the IRs also changed, so that it does not take count while traversing through the black lines.

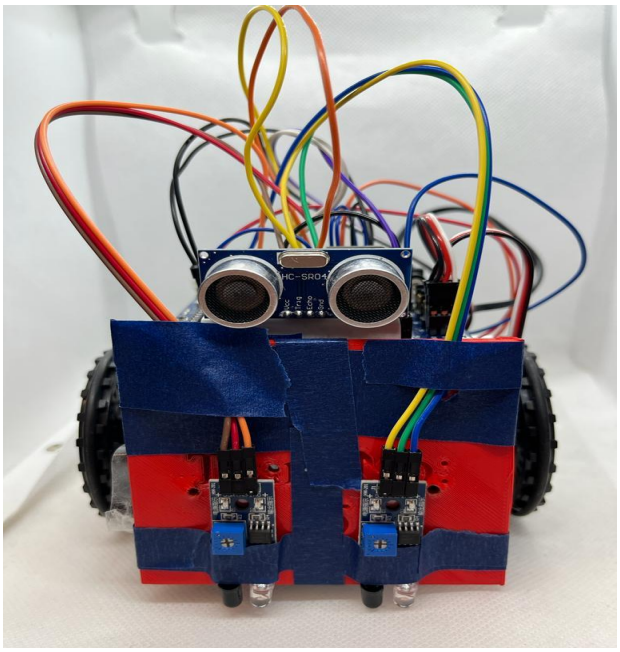
```
void distance_measurement_function(void *par)
{
        while(1)
        {
        printf("function started");
        if(final_counter == 1)
                {
                last_left_ir = 0;
                last_right_ir = 0;
                while(1)
                        {
                        printf("final counter \n");
                        if((left_ir == 1) &&(right_ir == 1))
                        {
                                printf("Intersection\n");
                                if(!((last_left_ir ==1 )&&(last_right_ir==1)))
                                {
                                        distance_counter = distance_counter + 1;
                                        printf("%d \n", distance_counter);
                                }
                                }
                                last_left_ir = left_ir ;
                                last_right_ir = right_ir;
                        }
                }
        }
}
```

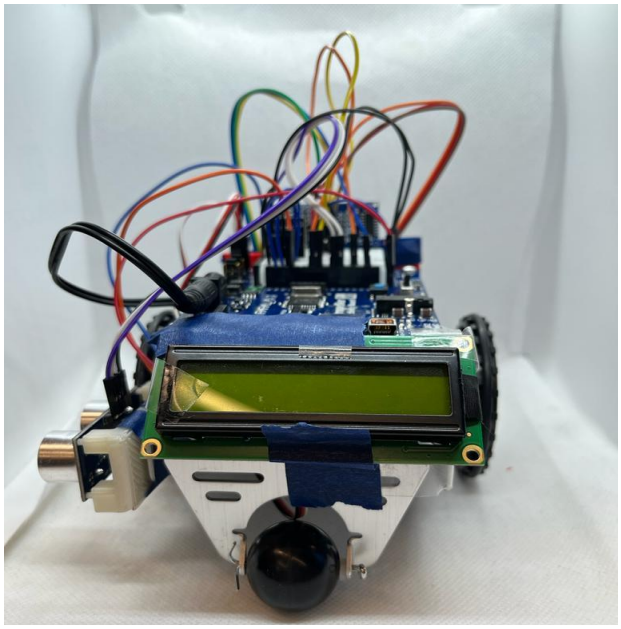
There was another problem with this approach. The nested if conditions were working fine but while making turns, these conditions are still satisfied thus taking extra counts sometimes on turns.

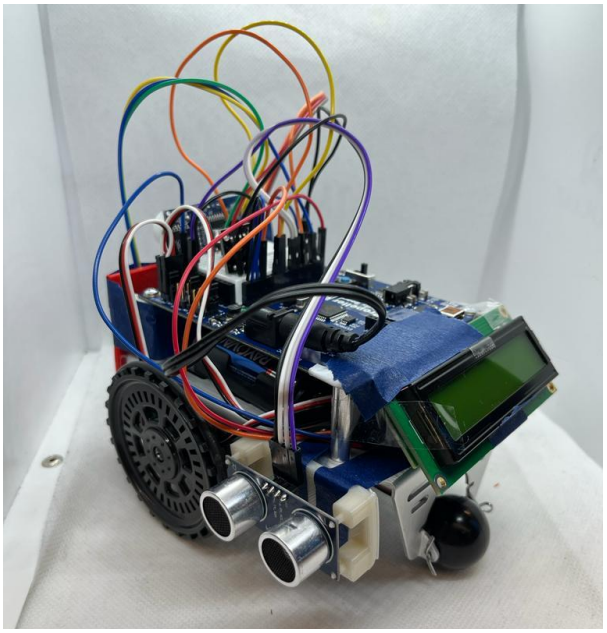
To avoid this together we decided to not go with a separate function that will run on a cog. The next and final approach is that we just take the count of the number of intersections already traveled, and subtract it by 5, which gives us the count left to reach the end of Lane A's junction. The intersections traveled on Lane B were kept track of, as well, and added to the initial number. We add 1 to it to include the middle intersection on the way to Lane B from Lane A. This approach was foolproof.

Robot Images









Scope of Improvement

In this project the robot traverses back to intersection A1 after picking the widget and then travels across the primary lane via the bi-directional lane to enter lane B at intersection B1 where it begins the process of identifying the first available parking spot in this lane. Alternatively, we can implement a more efficient method of finding the first available parking spot.

A. Using a search algorithm to find the shortest path from the pick-up spot to the drop-off spot. In this situation, we would have to provide the robot complete layout of the map including the start and end destination.
B. Use additional ultrasonic sensors. By adding an additional ultrasonic sensor to the right of the robot it can be programmed to take constant readings of its left and right while it moves on the primary lane. As soon as it identifies an available widget in lane A it turns towards it. The same can be done to identify the drop-off location in lane B. If the drop-off intersection is not registered before the robot turns to lane A, it resumes its search from the last intersection on the main lane that was taken before it turned right.

Conclusion

We developed an Automated Guided Vehicle that follows a line on the following grid. The AGV uses two IR sensors for the line following task and two ultrasonic sensors to identify dynamic obstacles and locations to pick up and drop-off locations for the widgets in lanes A and B respectively.

The team was able to successfully travel the primary lane while taking an alternative route when a dynamic obstacle was introduced. The team used a blue light to indicate intersections, red light to indicate obstacles, cyan light to indicate picking up a widget and a green light represents the widget being dropped off. The robot then indicates the distance between the pick-up and drop-off locations as a multiple of 40 cm on an LCD display. The robot also shows a visual difference in speed between the primary lane(fast) and lanes A & B (slower).

This project required us to use the Propeller microcontroller and was focused on creating smoother motions as compared to the previous Arduino projects as the propeller microcontroller uses 8-cores.