Term Project

Line Following Robot for Detecting Friendlies and Enemies with Obstacle Detection

Group 19

1) **Dajr Alfred**
2) **Mohammed Nauman Shariff**
3) **Gordon Oboh**

05/16/2022

NYU

Contents

Term-Project 1

Contents 2

1. Introduction 3

Brief Description of the scenario 3

1.1 Problem Objective and its Solution 3-4

2. Methodology 5

2.1 Circuit Diagram and explanation 5-6

2.2 Pin Representation 7

3. Software Session 8

3.1 Raspberry Pi Functions and Description 8

3.2 Propeller Functions and Description 9

3.3 Arduino Code 9

3.4 Propeller Code 10

4. Hardware 11

4.1 Raspberry Pi Activity Board 11

4.2 Raspberry Pi V4 Camera 12

4.3 Parallax Propeller Activity Board 14

4.4 Ultrasonic Sensor 15

4.5 Pololu IR sensor Array 16

5. Result and Conclusion 17

NYU

1. Introduction

Brief Description of the scenario

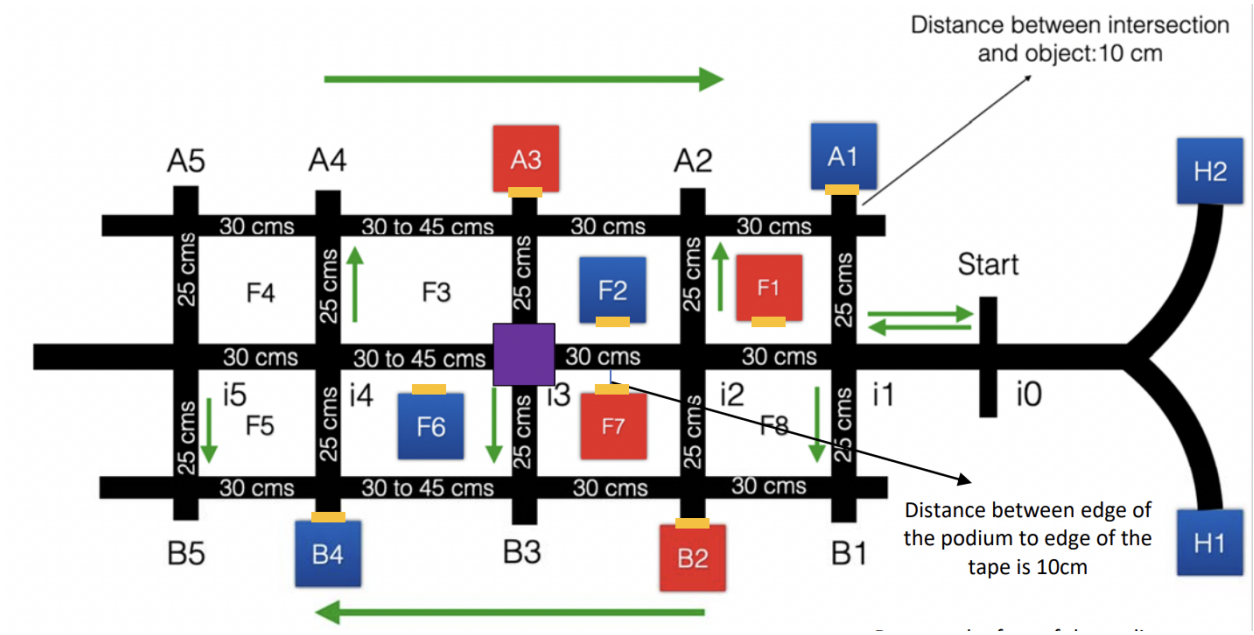


Figure 1.1: The arena, A grid-like structure

1.1 Problem Objective and its Solution

Our task is to create an autonomous boe-bot that will scour Manhattan's streets for hostile intruders (also known as enemies) by not harming the Friendlies. The robot should follow the outlined course while avoiding the obstacles in the way by obeying the traffic laws of New York City.

Figure 1 depicts the map's outline in a grid-like form, similar to the Manhattan and Brooklyn streets. There are two home locations (H1 and H2) from which the robot enters the track. There are 16 intersections (i0 to i5, B1 to B5, and A1 to A5) with ten possible delivery destinations and one obstacle that is randomly placed in either i2, i3, or i5. Each street's traffic flow is represented by green arrows, with the center lane being bi-directional.

To follow the line and detect junctions, our line following robot uses an eight-channel digital IR Reflectance sensor. IIf the leftmost IR sensor detects black tape, the Arduino compensates by shifting the robot to the right. If the IR sensor on the right detects black tape, the Arduino compensates by moving to the left. The bot follows the tape's movement using this technology but instead of pausing to scan, the propeller's multicore architecture allows the robot to complete all of these tasks at once, or at least faster than a single-core microcontroller.

2. Methodology

2.1 Circuit Diagram and explanation

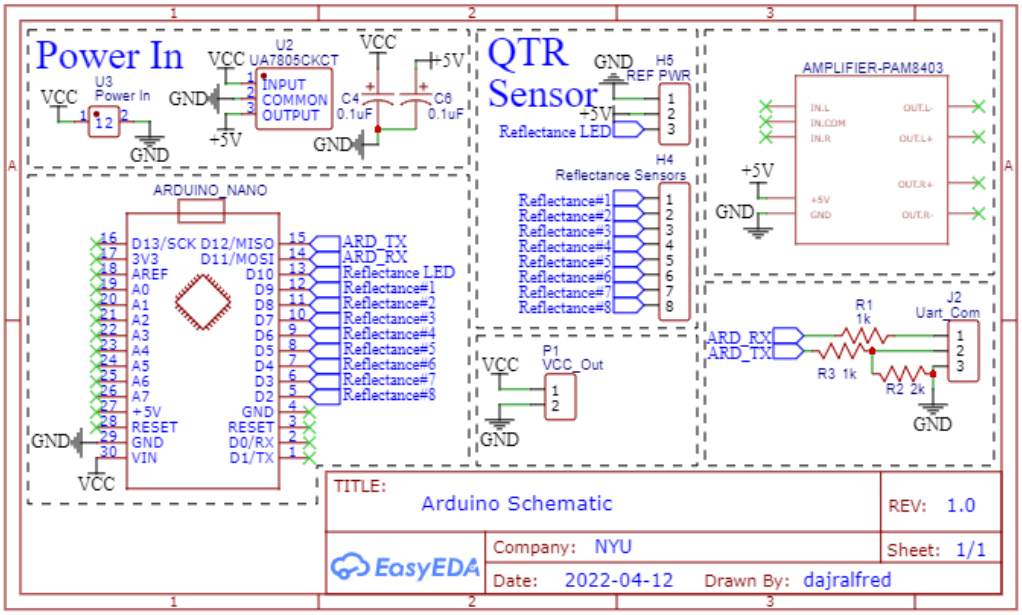


Figure 2.1: Arduino Circuit Diagram

Figure 2.1 shows how the Arduino Nano is connected to the QTR sensor and the Pins that are used to send the data via Serial Communication.

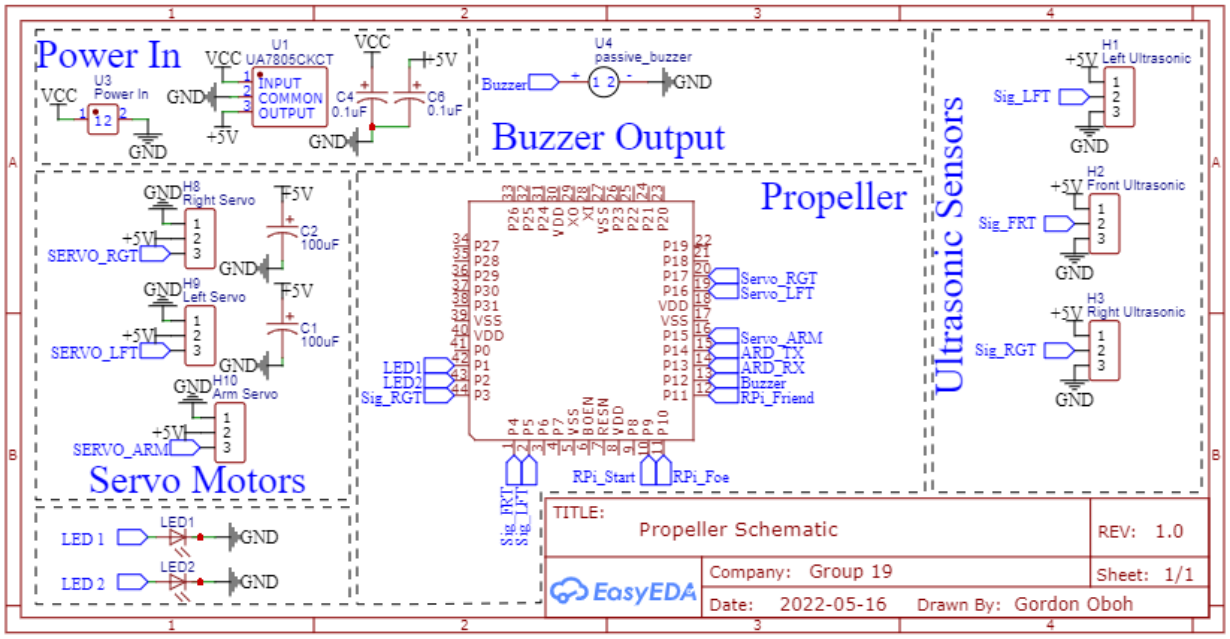


Figure 2.2: Propeller Circuit Diagram

Figure 2.2 shows how the Propeller is connected to the servo motors, LEDs, buzzer, and Ultrasonic sensors and the pins used for Serial communication with the Arduino Nano.

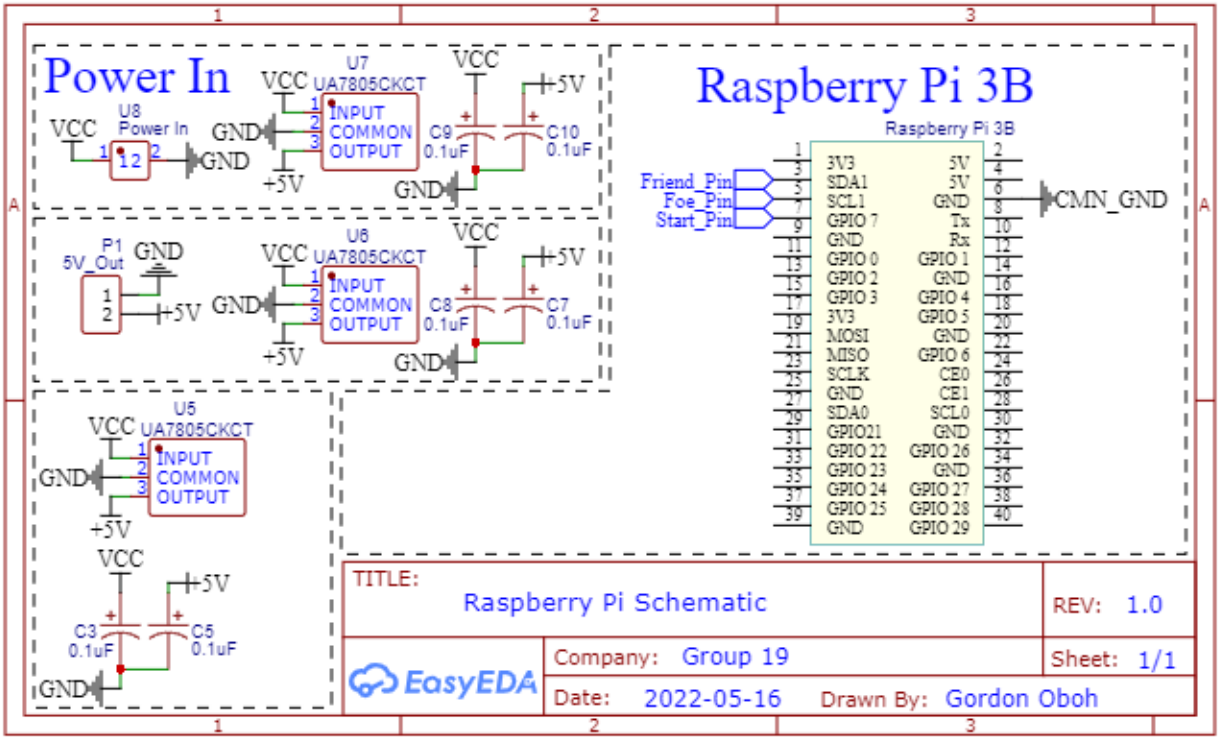


Figure 2.3: Raspberry Pi Circuit Diagram

Figure 2.3 shows how connections for the Raspberry Pi, the Raspberry pi has a power circuit to give it appropriate power at 5V, it's also important that it shares a common ground with the propeller, or it might have problems with communicating.

2.2 Pin Representation

Arduino Connection:

- Pin D2 - Reflectance Pin 8.
- Pin D3 - Reflectance Pin 7.
- Pin D4 - Reflectance Pin 6.
- Pin D5 - Reflectance Pin 5.
- Pin D6 - Reflectance Pin 4.
- Pin D7 - Reflectance Pin 3.
- Pin D8 - Reflectance Pin 2.
- Pin D9 - Reflectance Pin 1.
- Pin D10 - Reflectance LED Pin.
- Pin D11 - UART Receiver Pin.
- Pin D12 - UART Transmitter Pin.

Propeller Connection:

- Pin 1 - LED 1.
- Pin 2 - LED 2.
- Pin 3 - Signal Pin for Right Ultrasonic sensor.
- Pin 4 - Signal Pin for Front Ultrasonic sensor.
- Pin 5 - Signal Pin for Left Ultrasonic sensor.
- Pin 9 - RPi Trigger.
- Pin 10 - Foe Pin.
- Pin 11 - Friend Pin.
- Pin 12 - Buzzer.
- Pin 14 - UART Transmitter Pin.
- Pin 13 - UART Receiver Pin.
- Pin 15 - Signal Pin for Arm Servo.
- Pin 16 - Signal Pin for Left Servo Motor.
- Pin 17 - Signal Pin for Right Servo Motor.

Raspberry Connection:

- Pin 3 - Friend Pin, used to tell Propeller if a friend is encountered.
- Pin 5 - Foe Pin, used to tell Propeller if a foe is encountered.
- Pin 7 - Start Pin, this pin is used by the Propeller to tell the Raspberry when to scan for enemies.
- Pin 6 - Common Ground with Propeller

An Arduino Nano was used due to space constraints and to act as a slave to a Propeller Master, because of Arduino's open-source nature it has support for the QTR Sensory library, unlike the propeller. Instead of trying to "reinvent the wheel", it was better to use the Arduino and its access to the needed library to read the data from the QTR sensor. Later send that data via a Serial Communication to the Propeller, while the propeller uses this data in its decision making and branching.

Using a camera, The Raspberry Pi scans for friends or foes, the information is passed to the Propeller, the propeller then attacks or continues to move on.

3. Software Session

3.1 Raspberry Pi Functions and Description

On raspberry pi we chose to use python as our programming language, we imported the necessary libraries and declared a FriendOrFoe function, this function takes TagID as a parameter and is then used to determine if a tag is a Friendly or enemy, a sound is played depending on the categorization and the Friend or Foe Pin is set high as way of communicating the information to the Propeller, the remaining part of the code allows us to initialize the raspberry pi to scan the aruco tag and get a tag id, we set the fps of the camera to 5fps, this helped to reduce the number of times the Raspberry mistakenly triggered.

3.2 Propeller Functions and Description

getPos(); This function starts a cog and the cog is used to read the values of the QTR sensor sent by the arduino nano via serial communication to Propeller.

getDist(); This function uses the built-in built command *ping_cm* to read the length of the return pulse and calculate the time in centimeters.

soundBuzzer(); This function starts a cog and is used to trigger the buzzer and then the cog is stopped at the end.

lineFollower(); This function starts a cog and is used to ensure the robot follows the line, the function compares the difference of the present position from the desired position to a defined error threshold, and then the cog is stopped at the end.

detectIntersection(); This function compares the 8 values from the QTR sensor to a reference threshold and returns a 0 or 1.

intersectionSequence(); This function triggers the buzzer.

Follow(); This function is used to start the cog for *lineFollower()*.

spinServos(); This function uses a switch statement as a conditional to select the speed and direction of the servo, below is a brief explanation of the result of each case.

- **Case 0**; this case uses the *servo_set* command to stop the servos.
- **Case 1;** this case uses the *servo_speed* command to make the servos move the robot forward.
- **Case 2**; this case uses the *servo_speed* command to make the servos spin the robot left on its wheel axis.
- **Case 3**; this case uses the *servo_speed* command to make the servos spin the robot right on its wheel axis.
- **Default Case**; this case sets the switch expression to case 0.

This function runs on an independent cog, uses a while loop to make it repeat, and is also the backbone for the functions below.

Forward(); it takes one parameter for the speed and passes that to *spinServos()* with the switch expression *motorDir* set to 1.

TurnLeft(); it takes one parameter for the speed and passes that to *spinServos()* with the switch expression *motorDir* set to 2.

TurnRight(); it takes one parameter for the speed and passes that to *spinServos()* with the switch expression *motorDir* set to 3.

Stop(); it takes one parameter for the speed and passes that to *spinServos()* with the switch expression *motorDir* set to 0.

Turn180(); it takes no parameter and uses the switch expression *motorDir*(set to 3) with *pause()* command to make the robot spin 180° on its wheel axis.

Turn90L(); it takes no parameters and uses the switch expression *motorDir*(set to 3) with *pause()* command to make the robot turn left on another lane.

Turn90R(); it takes no parameters and uses the switch expression *motorDir*(set to 2) with *pause()* command to make the robot turn right on another lane.

3.3 Arduino Code

The Arduino code is used to read the QTR sensor and send the data to the Propeller via Serial communication. The code initializes 11 pins, Pin 1 to 8 for each array on the QTR sensor, Pin 10 for the emitter control pin, Pin 11 and 12 for Receiving and Transmitting data respectively. In the setup part, the QTR sensor is set to a digital mode and there's a calibration sequence to allow the QTR sensors to be calibrated for each run. The Loop section is used to keep reading the values from each sensor pin.

3.4Propeller Code

The Propeller uses a switch statement to control the flow of the program and trigger an appropriate response based on inputs and conditions, below is a brief explanation of the result of each case;

- **Case 0**; this case gets QTR Sensor data from the Arduino and uses that to follow the line from H1 or H2 to intersection i0.
- **Case 1**; this case triggers the front ultrasonic and it's used to scan for obstacles in front of the robot, once the robot moves beyond intersection i0.
- **Case 2**; the front ultrasonic is used to approach an obstacle, then this case is activated and it returns the robot to a i1 and then it branches off to the B1.
- **Case 3 and 4**; these case statements use the left ultrasonic sensor to scan the delivery locations for objects and an object counter is incorporated. These case statements decide when to switch from B lane to A lane and when to loop around the entire track. If all 2 objects are found then the robot stops, else if the robot loops around the track and doesn't sense all the objects, it then goes to B5 to look for the final object,
- **Default Case**; this case ends Line following and stops the robot.

4. Hardware

4.1 Raspberry Pi 3B

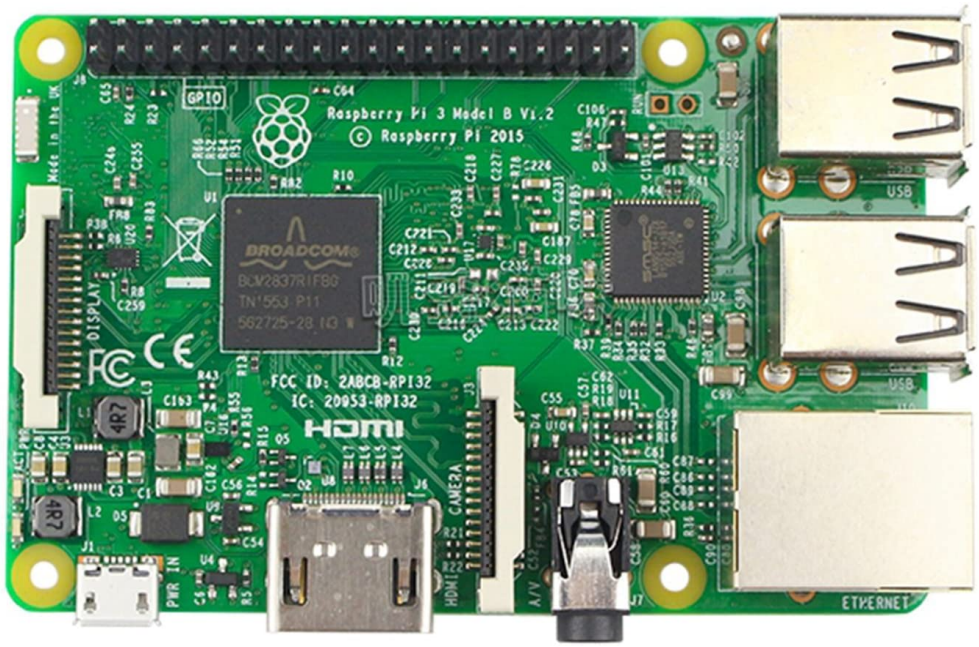


Figure 4.1: Raspberry Pi Computer Board

The Raspberry Pi is a low-cost computer whose size is equivalent to a credit card. It connects to a computer monitor or TV and utilizes a conventional keyboard and mouse. It utilizes Broadcom BCM2837 SoC. It has low power consumption and is well-suited for mobile computing. The processor is based on ARM Cortex-A53 which is running at 1.2Ghz (CPU). The GPU features a fast 3D core that can be accessed via OpenGL ES2.0 and OpenVG, and it can connect microcontrollers like Arduino to the Raspberry Pi via low-level peripherals.

In the accompanying term project, we use a Raspberry Pi device with a camera to detect Arco code, which tells us which tags are enemy and which are friends.

4.1 Raspberry Pi V2 Camera

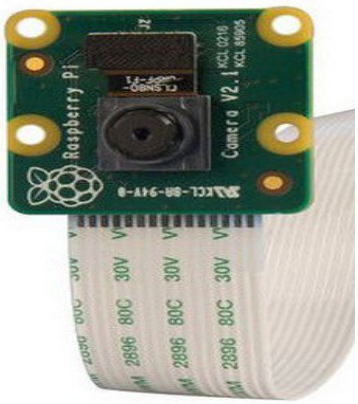


Figure 4.2: Raspberry Pi V2 Camera

The Camera Module 2 can capture both video and still images in high definition. The Raspberry Pi Camera Module v2 is a custom made add-on board for Raspberry Pi with an 8 megapixel Sony IMX219 image sensor and a fixed focus lens. It supports static photos up to 3280 x 2464 pixels and video up to 1080p30, 720p60, and 640x480p90.

Features:

- On-board fixed focus lens
- Dimensions: 25mm x 23mm x 9mm
- Weight: just over 3g
- A ribbon cable connects to the Raspberry Pi board.

4.2 Parallax Propeller Activity Board

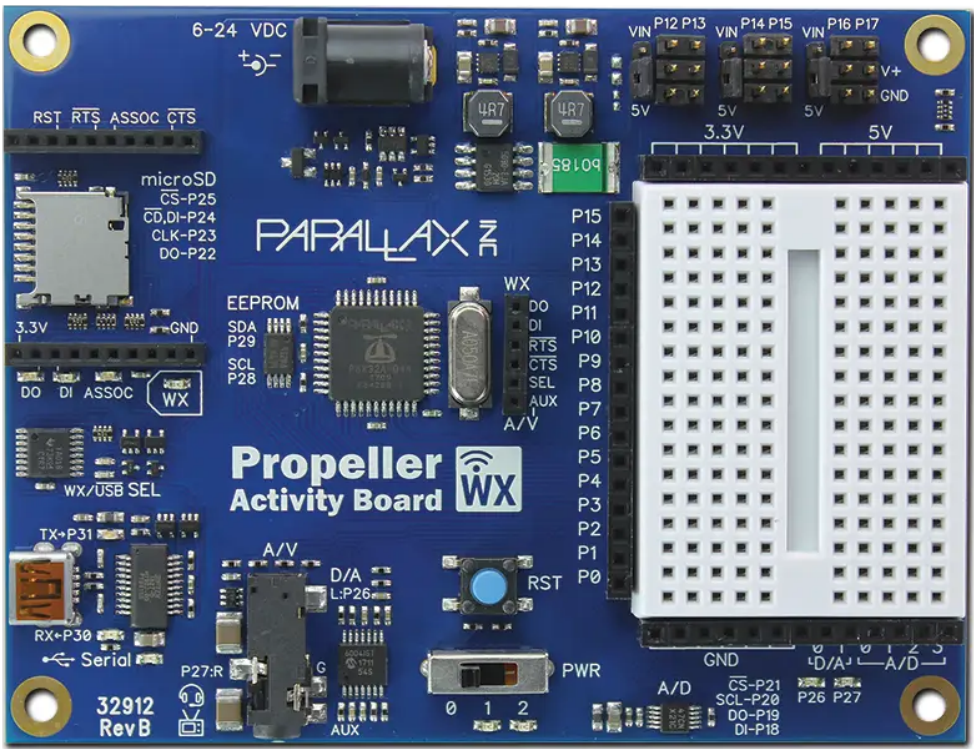


Figure 4.1: Parallax Propeller 2 Activity Board WX

A P8X32A-Q44 microprocessor powers the Propeller Activity Board WX.;It requires an external power supply of 6-24V. There are three separate memory areas to consider:

- Global RAM - 32KB
- Global ROM - 32KB
- Cog RAM - 2KB per Cog

Each Cog can access all 32 Pins and use them as inputs or outputs; the method set direction() is used to do so., The first argument is a pin number, while the second parameter is a state(1 or 0)., set output() uses the same syntax as set direction to set the pin to high or low ().

The Propeller connects to the Arduino as a master, receiving and processing the QTR sensor data.

4.3 Arduino Nano

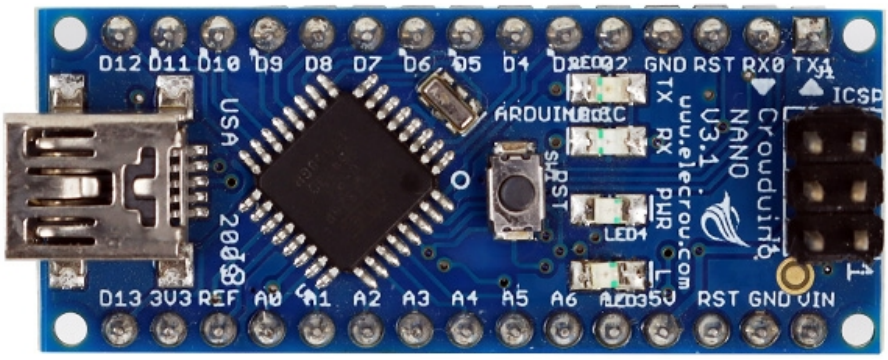


Figure 4.2: Arduino Nano Microcontroller

The Arduino Nano is fueled by a 6-20V unregulated power supply and a Mini-B USB connection. It uses the same ATmega328 microcontroller as the Arduino Uno. It has three separate memory kinds.

- Flash memory - 32kb.
- EEPROM - 1kb.
- SRAM - 2kb.

The Arduino serves as a slave to the Propeller, transferring data from the QTR Sensor to it.

4.4 Ultrasonic Sensor

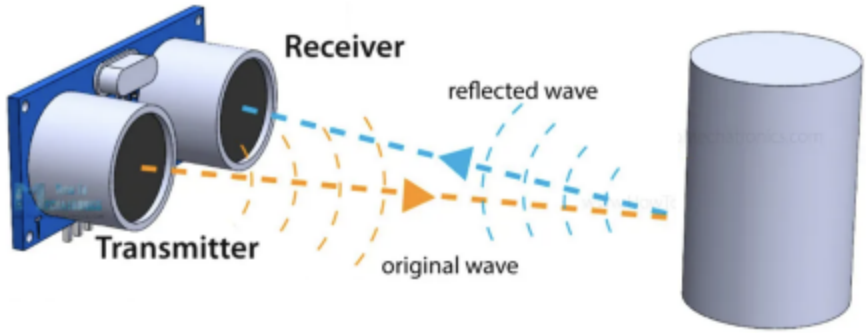


Figure 4.3: Ultrasonic waves transmitted and received by an Ultrasonic Sensor

The distance is measured with the HC-SR04 Ultrasonic sensor, which emits a 40kHz Ultrasound frequency. The Ultrasound waves bounce back to the receiver end if any object is detected along the way, as shown in Figure 3.3. The halving time necessary for Ultrasound waves to detect the object and bounce back to the receiver is used to compute the object's distance. The formula below is used to calculate it.

$$Distance = \frac{Speed\ of\ sound\ in\ air \times Time\ from\ transmitter\ to\ the\ receiver}{2}$$

For example: Find the distance of the object if the time required to detect the object is 254 µs. Given: Speed of Sound in Air = 340 m/s = 0.034cm/µs

Time =1300 µs

Distance $= \frac{Speed \times Time}{2}$

$= \frac{0.034 \times 254}{2}$

=4.318 cm

General Pin Representation of Ultrasonic Sensor:

- Pin 1 - Vcc
- Pin 2 - Trigger
- Pin 3 - Echo
- Pin 4 - Ground

We're utilizing three ultrasonic sensors in this project to detect the object and barriers in three separate directions (Right, Front and Left).

4.5 Pololu IR sensor Array

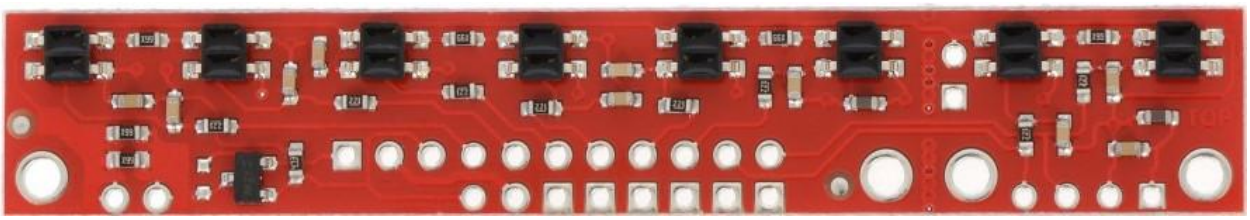


Figure 4.4: QTR-8RC Reflectance Sensor Array

The IR sensor array, which includes eight infrared LEDs, is primarily used for sensing light and dark points on a plane, which is ideal for line-following robots. Some of the important features of QTR-8RC are pairing with LEDs for reduced current consumption and MOSFET for power savings. Each of the eight sensors has its own digital I/O measurable output. Aside from that, no analog to digital converter is required.

Specifications:

- Dimensions: 2.95" x 0.5" x 0.125" (74.93mm x 12.7mm x 3.175mm)
- Required operating Voltage : 3.3-5.0 V
- Required supply current: 100 mA
- Output format: 8 digital I/O-compatible signals
- Optimal sensing distance: 0.125" (3 mm)

We're utilizing the Pololu IR sensor array in this project to detect the black line (tape) and follow the robot along the track. The IR sensor must be calibrated, which can take up to 7-10 seconds. Each IR returns a number ranging from 0 to 1000 for maximum and minimum reflectiveness. Each IR LED is momentarily passed over the black tape before returning to the white paper background; this is done as many times as feasible for better calibration. The last chapter covered the Pin representation of the Pololu sensor's connection to the Arduino Nano board.

5. Result and Conclusion

After the implementation of the line following robot with obstacle avoidance and threat detection, 4 friendlies and 4 foes are kept at different locations (B5 and A4) and an obstacle was placed at the center lane (i2). In our demonstration video, the robot starts from a home location and continues down the center lane until it encounters an obstacle and then returns back on the center lane to i1. After that, it takes a turn to B1, and then scans for intruders between B1 to B4, it then makes a turn on i4 and moves to A4, and scans from A4 to A1.

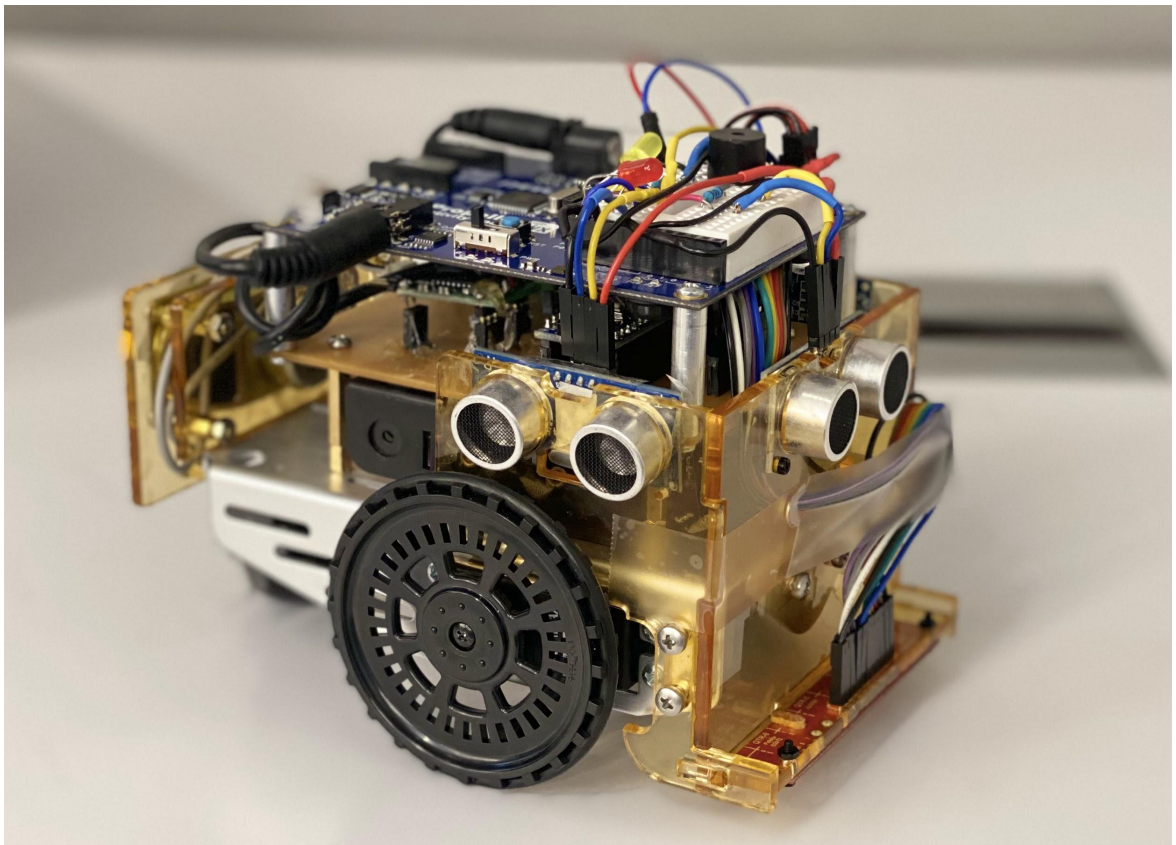


Figure 5.1: Picture of the Line Following Robot with Obstacle avoidance threat detection and an arm to eliminate intruders.