**Internship Project Report**

From: 01/02/2021 To: 06/03/2021

**Company**



**Rabiie Echriff**  

**Table of Contents**

**Introduction**3

**Working Principle**4

**Hardware Components**5

**Software Used**7

**Train Simulation**8

**Train Simulation Source Code**9

**Circuits**12

Ultrasound/Uno Side12

Calculating Coordinates13

Uno Source Code14

Train/Nano Side16

Nano Source Code17

Energy Problem20

**Assembly21**

**Result22**

**Conclusion22**

**Possible Solutions22**

**Introduction:**

**1-Objective:**

The aim of this project is to create a system that accurately tracks the location of a train and displays it in a simulated 3D environment in real-time, one that can be applied on larger scale, since Wi-Fi/radio distance measurement is unreliable on a small scale, for the purpose of this project, ultrasound sensors will be used instead, as the principle remains the same.

**2-Problem definition:**

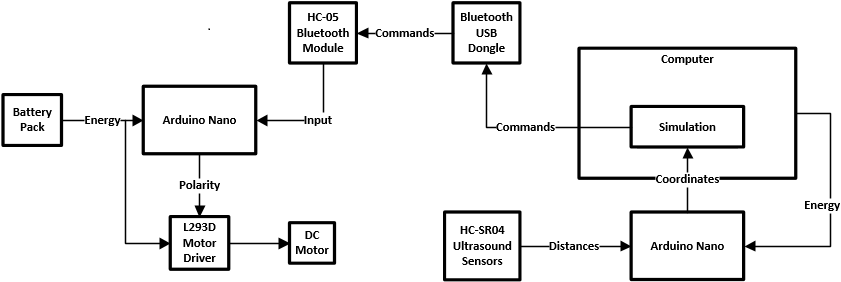
The current train system in Tunisia is very inefficient, and has many issues, mainly, there is a lack of coordination between trains, which leads to the train schedule being pretty much non-existent.

This project aims to eliminate this problem by accurately tracking the location, speed and direction of each train simultaneously, providing better coordination to all of them.

**3-Project Scope:**

The plan is to develop the system further to further so that we can track multiple trains at the same time, as well as control all of them remotely through and interface, but that will require the use of something other than ultrasound.

**4-Diagram:**



**Working Principle:**

A single HC-SR04 ultrasound sensor’s default function is simple, detecting whether there’s an object in front of it or not, but by measuring the time it takes for the ultrasound to bounce back from the object, we can actually calculate the distance between the sensor and the object…

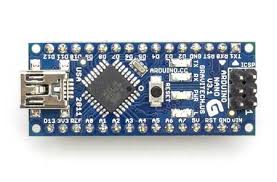
The sensor in this case will be a stand-in for a Wi-Fi tower, which works on the same principle of calculating the distance between it and the target based on how long it takes for the signal to return, since Wi-Fi distance measuring doesn’t work as accurately on a small scale, ultrasound will be used to simulate that effect.

From here we can use multiple sensors to calculate the coordinates of a single object at any given moment, the idea was to use 3 sensors positioned at 120° angles from each other to triangulate the position of the object in question (In our case a tory train) but at some point I realized that considering where the train will be placed, 2 sensors should do the job.

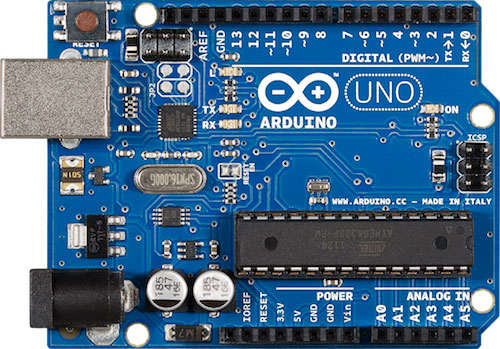
So in short, these 2 HC-SR04 will be sending data to an Arduino Uno board, which when provided with the sensors’ position, is able to calculate the train’s position and send it to a PC where further manipulations will take place, such as calculating the train’s speed and angle based on its previous position, as well as displaying the train in a simulated environment controlling it through Bluetooth.

**Hardware Components:**

-An Arduino Nano Board.



-An Arduino Uno Board.



-HC-05 Bluetooth transceiver.



-Two HC-SR04 Ultrasound sensors.



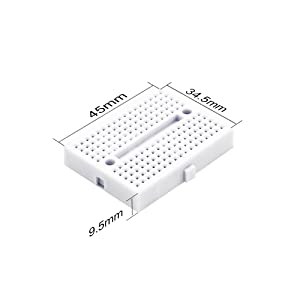
-USB 2.0 Cable Type A/B.



-Breadboard.



-FourSmall Self-Adhesive Breadboards.



-Male- To-Male Jumper Wire.



-1 kΩ & 2 kΩ Resistors.

-L293D Integrated Circuit.



-USB Bluetooth Dongle.



-Desoldering Pump.



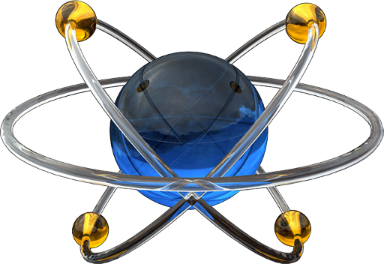


-Soldering Kit.



**Software Used:**

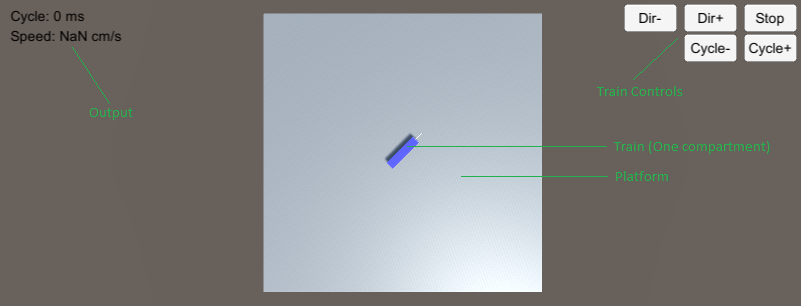
Unity3D: A commercial, free, cross-platform game engine that is widely used to develop countless 2D and 3D games, we’ll be using it to create a simplistic train simulation that receives data from the USB ports.

Proteus: A versatile tool used for many aspects of electronic design, since the circuits of this project are relatively simple, we’re only going to be using it for schematics.

Arduino IDE: The official tool used to write and compile code for Arduino boards, but can also be used for non-Arduino related serial communication (read & write), both of these features are very useful for this project.

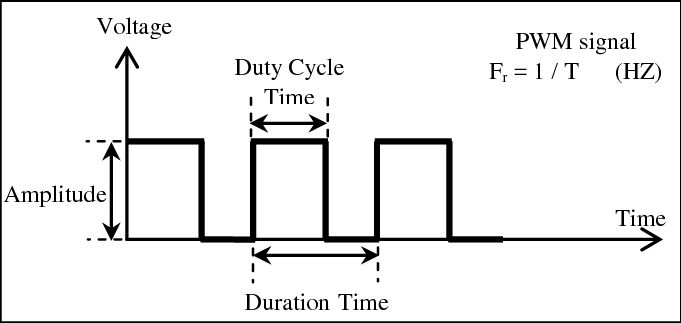
Notepad++: Great code editor that we’ll we using to write Unity scripts, no need for a compiler as Unity will handling that for us.

**Train Simulation:**



-This is the 3D simulation that receives the train’s coordinates from the system and uses it to update the simulated train’s position, as well as to calculate additional information, such as the train’s speed and angle, by comparing its previous position to its current one.

-The ‘Cycle’ value displayed in the top-left corner Is the duration of the PWM (Pulse-Width Modulation) used to indirectly control’s the train’s speed by feeding it voltage in interrupted bursts (I talk about this more in detail later).



-The buttons in the top-right corner are sued to control the train via Bluetooth, by sending commands such as turning the motor on and off, changing its direction, and increasing/decreasing it’s PWM.

*Note:* Train rotation display and a ‘Start’ button (Which was present in a previous version) are missing due to my own inadequacy.

**Train Simulation Source Code:**

1. *//Importing all the basic Unity classes, as the well as the one required for serial communication.*
2. **using** System.Collections;
3. **using** System.Collections.Generic;
4. **using** UnityEngine;
5. **using System.IO.Ports;**

8. **public** **class** TrainScript : MonoBehaviour
9. {
10. **public string portName1;**
11. SerialPort arduino; *//Input USB port for receiving train coordinates.*
13. **public** **string** portName2;
14. SerialPort bluetooth; *//Output USB port for sending commands to train.*

17. **public** **float** speed;
19. *//XY receives from serial communication so it can be stored in X AND Y respectively.*
20. **public float XY;**
21. **public** **float** X;
22. **public** **int** Y;
24. **public** **float** cycle;
26. Vector3 old\_coo; *//Two vectors for storing previous and current train coordinates.*
27. Vector3 new\_coo;
29. **float** inclination;
31. **bool** switcher; *//Unity for some reason can't communicate with two serial ports*
32. simultaneously, so **this** boolean will **switch** between the two ports each frame.
34. **public** **bool** engn\_on;
35. **public bool energy;**


39. **public** **float** next\_time1;
40. **public float next\_time2;**

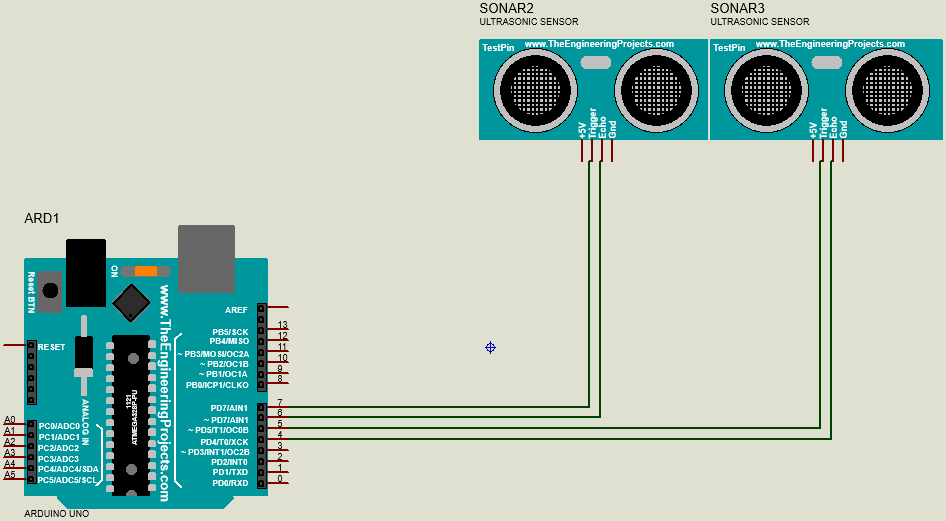
43. **void** Start()
44. {
45. ***//Assign the two serial ports for I/O communication.***
46. arduino = new SerialPort (portName1, 9600);
47. arduino.Open ();
49. bluetooth = new SerialPort (portName2, 9600);
50. **bluetooth.Open ();**
52. engn\_on=**false**; *//Set engine start as off.*
53. energy=**true**;
55. **switcher=true;**
56. }

59. **void** Update()
60. **{**
62. old\_coo=transform.position; *//Get train coordinates before updating (Old coordinates).*

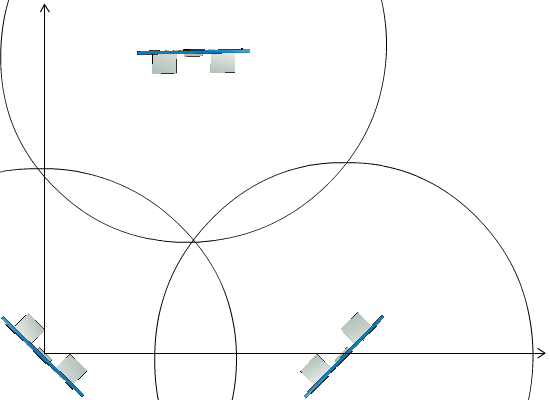
65. **switch(switcher){**
66. **case** **true**:
67. **if**( Time.time > next\_time1){ read\_arduino(); switcher=**false**; next\_time1 = Time.time + 0.5f; **break**; }
69. **case** **false**:
70. **if( Time.time > next\_time2){ read\_bluetooth(); switcher=true; next\_time2 = Time.time + 0.5f; break; }**
72. **default**: **break**;
73. }
74. *//Functions are only called every 0.5 seconds.*
76. transform.position = new Vector3(X, 0.5f, Y); *//Update coordinates.*
78. new\_coo=transform.position; *//Get train coordinates after updating (New coordinates).*
80. **if**(new\_coo!=old\_coo){
81. inclination=Mathf.Atan2(new\_coo.x-old\_coo.x,new\_coo.z-old\_coo.z) \* Mathf.Rad2Deg; *//Calculate rotation.*
82. transform.rotation = Quaternion.Euler(0, inclination, 0); *//Assign rotation.*
83. }
85. speed=(Mathf.Sqrt((new\_coo.x-old\_coo.x\*new\_coo.x-old\_coo.x)+(new\_coo.z-old\_coo.z\*new\_coo.z-old\_coo.z)))\*2; *//Calculate speed.*
86. }
87. *//Functions:*
88. *//Functions for sending commands to train Arduino board, which are interpreted based*
89. **on the ASCII value of the character sent.**
90. **public** **void** cycle\_plus(){bluetooth.Write ("+");bluetooth.Write ("d");} *//Increase PWM.*
91. **public** **void** cycle\_minus(){bluetooth.Write ("-");bluetooth.Write ("d");} *//Decrease PWM.*
92. **public** **void** forward(){bluetooth.Write ("f");} *//Set direction to forwards.*
93. **public** **void** backward(){bluetooth.Write ("b");} *//Set direction to backwards.*
94. **public void stop(){bluetooth.Write ("s");} *//Start/Stop***
96. *//Function for reading and handling USB input (Coordinates).*
97. **public** **void** read\_arduino(){
98. XY=**float**.Parse(arduino.ReadLine()); *//Serial port can only send one variable at a time.*
99. **X=Mathf.Floor(XY); *//Extracting X coordinates.***
100. Y=Mathf.RoundToInt((XY-X)\*100); *//Extracting Y coordinates.*
101. }
102. *//Function for handling USB ouput (Train controls).*
103. **public** **void** read\_bluetooth(){
104. **cycle=float.Parse(bluetooth.ReadLine());**
105. }
106. }

**Circuits:**

**Ultrasound/Uno Side:**

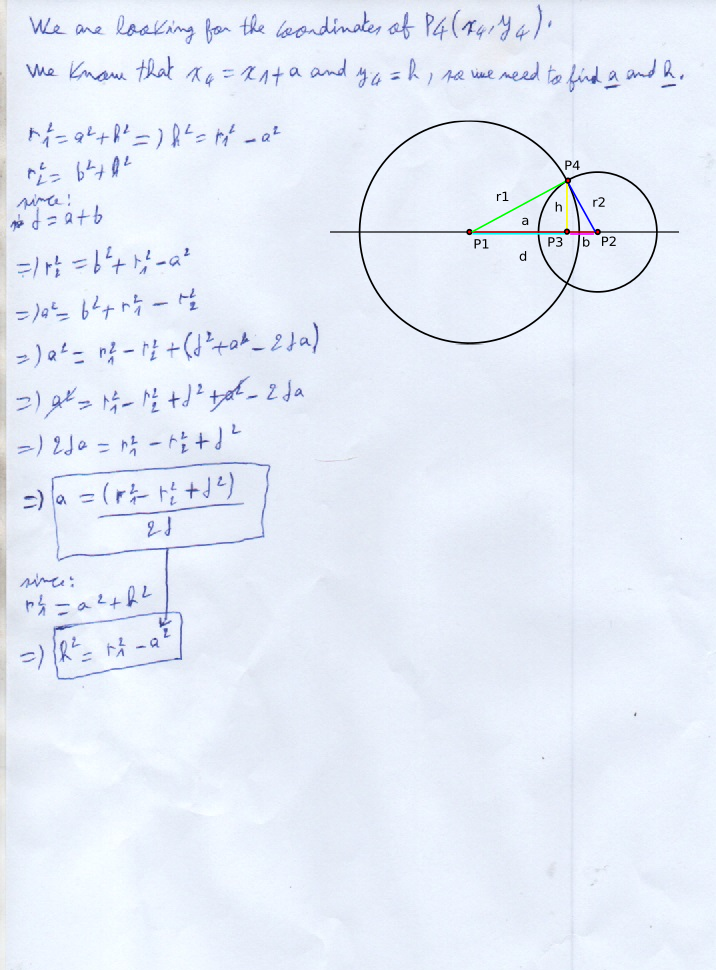
****

This is the circuit responsible for tracking the toy train, it’s very straightforward, the echo and trigger pins of each sensor are connected digital I/O pins on the Arduino, which in turn is connected to a PC through a USB cable (Not shown here).

This is roughly what the setup was originally going to be, the two bottom sensors, placed on the **X** axis, determine the two possible locations of the train *(represented by the two points in which the two bottom circles will be intersecting)*, and the top sensors will be responsible for determining which of those points is the correct one…

But since we know the train will always have a positive **Y** coordinate, the third (top) sensor becomes obsolete, all we need are two sensors angled in a way so that the train is always located in the intersection of both of their ranges (Roughly 30° degrees).

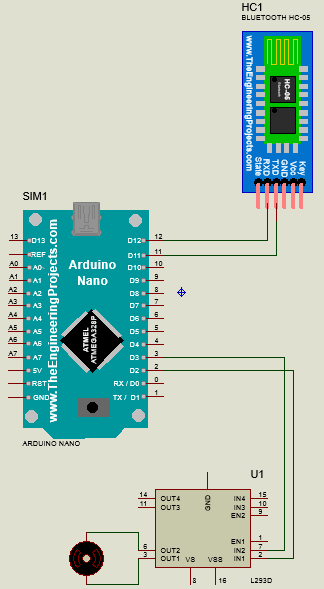
**Calculating Coordinates:**



**Uno Source Code:**

1. #include <math.h>
3. **int** ultrasound\_setup(**const** **int** echoPin, **const** **int** trigPin) { *//Function that sets sensor pins.*
4. pinMode(echoPin, INPUT);
5. pinMode(trigPin, OUTPUT);
6. }
8. **int** ultrasound\_loop(**const** **int** echoPin, **const** **int** trigPin) { *//Function that returns distance to object.*
9. **long** duration;
10. **int distance;**
11. digitalWrite(trigPin, LOW); *//Clear the trigPin.*
12. delayMicroseconds(2);
13. digitalWrite(trigPin, HIGH); *//Set the trigPin on HIGH state for 10 micro seconds.*
14. delayMicroseconds(10);
15. digitalWrite(trigPin, LOW);
16. duration = pulseIn(echoPin, HIGH); *//Read the echoPin & return the sound wave travel time in microseconds.*
18. distance= duration\*0.034/2; *//Calculate the distance.*
19. **return** distance; *// Print the distance on the Serial Monitor*
20. **}**
22. **void** send\_coordinates(**int** r1, **int** r2, **int** d) { *//Function that receives distance from object to each sensor, calculates its coordinates and sends them through serial port.*
23. **int** fx;
24. **int** x;
25. **float y;**
26. **String** ys;
27. **String** s;
29. x=(r1\*r1-r2\*r2+d\*d)/(2\*d);
30. delay(1);
31. fx=(r1^2-r2^2+d^2)/(2\*d);
32. delay(1);
33. y=sqrt(r1\*r1-x\*x);
34. delay(1);
35. ys=String(y/100);
36. delay(1);
37. ys.**remove**(0,2);
38. delay(1);
39. s=**String**(abs(x))+'.'+ys;
40. **delay(1);**
41. Serial.println(s);
42. delay(500);
43. }
45. int r1;
46. int r2;
48. int x1=0;
49. int x2=60;
50. int d=x2-x1;
52. String vString;
53. int velocity;
55. void setup() { *//Setup function.*
56. ultrasound\_setup(2,4);
57. ultrasound\_setup(8,10);
59. Serial.begin(9600); }
60. void loop() { *//Main loop.*
62. r2 = ultrasound\_loop(8,10);
63. delay(1);
64. r1 = ultrasound\_loop(2,4);
66. send\_coordinates(r1,r2,d);
67. }

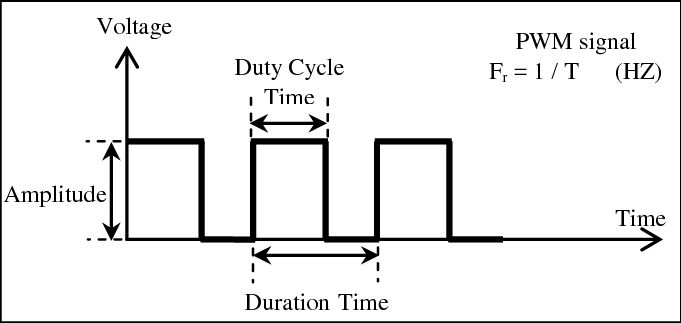
**Train/Uno Side:**

For the purpose of this project, we needed a tory train that can be controlled remotely to avoid interfering with the Ultrasound sensors’ input.

This is the circuit that is used to control the toy train, which is also fairly straight forward, the Arduino Nano board receives commands through the HC-05 Bluetooth transceiver (which can be sent through any Bluetooth device) and in turn sends commands to the train’s DC motor through the L293D circuit, which is a motor driver used to efficiently transmit commands and power to small motors such as this one.

The driver allows us to change the motor’s spinning direction by reversing its polarity, or increase/decrease its speed by varying the PWM.

Note: In both Arduino Schematics the power and ground connections are not included, that’s because this Arduino Proteus library doesn’t support those.

The way PWM works is by alternating the voltage fed to the motor between 0 and the required voltage (Amplitude) within a certain duration, in our case, the duration is 100 milliseconds, and the required voltage is around 3 Volts, the duty cycle time (DCT) will be varied by increasing or decreasing 10 milliseconds each time, so for example, if the DCT is something like 20 milliseconds, then the train’s speed will be slow, but if the DCT is equal to a 100, which is the full duration time, then the train will be receiving uninterrupted voltage and working at full speed.

**Nano Source Code:**

1. **int** in3 = 2; *//setting motor control pins.*
2. **int** in4 = 2;
3. **String** input;
4. **int** cycle = 0;
5. **char sign;**
6. **int** pinState1=LOW;
7. **int** pinState2=LOW;
8. **void** setup()
9. {
10. **Serial.begin(9600); *//Begin serial connection.***
11. pinMode(in3, OUTPUT); *//Set pin modes.*
12. pinMode(in4, OUTPUT);
13. }
14. **void** loop()
15. **{**
16. **if**(Serial.available() > 0)
17. {
18. input = Serial.read(); *//Receive Bluetooth input.*
20. **if(input == "45"){ *//Decide if the value received will be added to or subtracted from DCT.***
21. sign = '-';
22. }
23. **else** **if**(input == "43"){
24. sign = '+';
25. **}**
26. **else** **if**(input == "98"){ *//Set motor direction forward.*
27. pinState1=HIGH;
28. pinState2=LOW;
29. }
30. **else if(input == "102"){ *//Set motor direction backward.***
31. pinState1=LOW;
32. pinState2=HIGH;
33. }
34. **else** **if**(input == "115"){ *//Start/Stop*
35. **pinState1=LOW;**
36. pinState2=LOW;
37. }
38. **else**{
39. **if**(sign == '-'){
40. **cycle -= ((input.toInt())/10); *//Decrease duration based on input.***
41. }
42. **if**(sign == '+'){
43. cycle += ((input.toInt())/10); *//Increase duration based on input.*
44. }
45. **}**
46. }
47. digitalWrite(in3, pinState1); *//Set motor duration*
48. digitalWrite(in4, pinState2);
49. delay(cycle); *//Delay for DCT.*
50. **digitalWrite(in3, LOW); *//Stop motor***
51. digitalWrite(in4, LOW);
52. delay(100-cycle); *//Delay for res of duration.*
53. }

**Energy Problem:**

At this point in the project, I ran into an unexpected problem: How was I going to power both the train and the circuit installed on top of it?

Because normally, the train uses two 1.5V batteries connected in series (so 3V), but the Nano board requires at least 5V, and the L293D circuit has about a 2.6V drop on the output pins so I’m going to need at least 5.6V to power the whole thing, and even it would still be underpowered, and there was a delicate balance to keep in mind between finding a battery powerful enough and making sure it’s not too heavy for the toy train to carry around.

9V alkaline batteries such as this one seemed like the perfect choice initially, until I realized their true output is much lower than what’s being advertised and this isn’t the intended use for this kind of battery.

Someone on the Arduino Stack Exchange website recommended 7.4V Lithium batteries, but I couldn’t find any as they don’t appear to be available in Tunisia, after more research I found another solution:

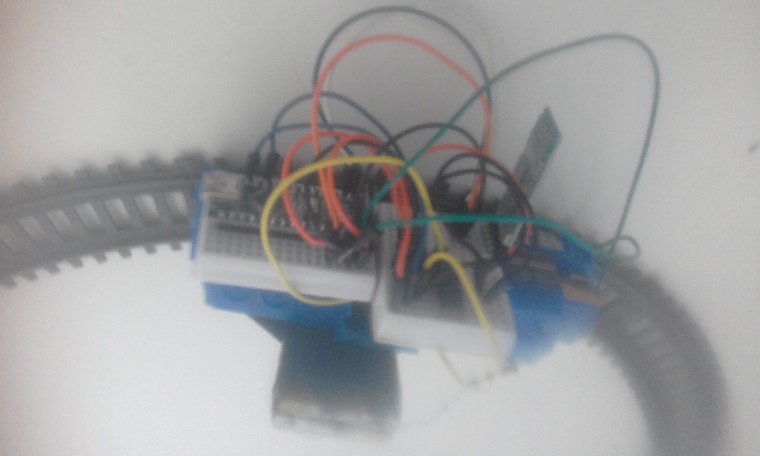
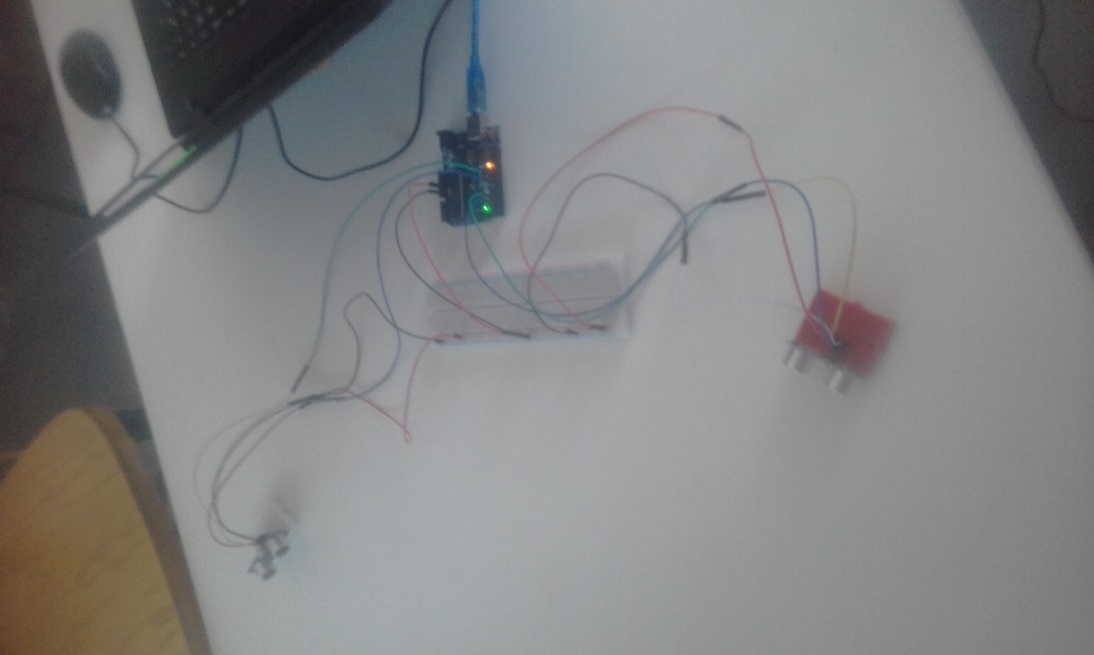
Standard Nokia phone batteries such as this one, output 3.7V, one of them on its own is enough to power the train’s DC motor, and two of them connected in series would form a 7.4V battery capable of powering the Nano board, the L293D circuit, and the motor in controls.

A small issue arises in the form of the batteries’ limiter circuits, which are the top part of the battery, where the pins are located, they need to be removed in order to not interfere with the voltage circulation between the two batteries, this guide by *TechBuilder* goes into more depth on how to do that, so I’ll just link it as to not plagiarize:

<https://www.instructables.com/Make-A-Cheap-Lithium-Battery-Pack/>

Once that was over the train worked with no issues.

**Assembly:**

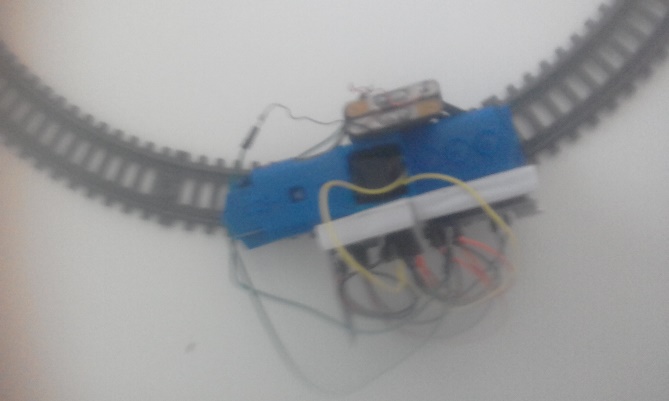


1-Ultrasound sensor setup.

2-Train circuit installed.



3-Train battery installed.



4-Train seen from above.

5-Train placed on tracks.

**Result:**-Train can be easily controlled through a phone, by sending the right strings of characters I can turn the train on and off, increase & decrease its speed and change its movement direction.

-Commands have a bit of a delay when sent through a computer as opposed to a phone, issue can stem either from Unity, or from the hardware itself.

-Ultrasound sensor device is able to locate train with relative accuracy, but only when it is stationary, once again, it is unclear if the issue is from the hardware itself or from Unity.

**Conclusion:**

-More optimal solutions are to be sought, either SR-04 ultrasound sensors aren’t the best tool for this job, or maybe a different platform other than Unity should be used for simulations.

-Even if this method was successful, it still will not be useful in the long run as it only allows the tracking of one, single-compartment train at a time.

**Possible Solutions:**

The real-time tracking issue can be due to a different number of factors, so many alternatives can be considered:

-Using a different, more optimized framework for simulations.

-Using a different kind of sensor for tracking.

-Using an entirely different method for tracking, such as.