



UNIVERSIDAD DEL BÍO-BÍO

CIMUBB

Ing. Civil En Informática

TUTORIAL

ROBOT PARA ROBOTOWN

Estudiante:

Marcelo Paz

Profesor supervisor:

Luis Vera

Fecha: 14 de enero de 2024



Índice

| | | |
|--------|---|----|
| 1. | Palabras del autor..... | 4 |
| 2. | Introducción..... | 5 |
| 2.1. | Contexto de la necesidad a resolver..... | 5 |
| 3. | Documentación de Hardware | 6 |
| 3.1. | WeMos D1 R32..... | 6 |
| 3.2. | Sensor de línea de 8 canales | 7 |
| 3.3. | Servo MG 996R..... | 7 |
| 3.4. | Herramientas Online utilizadas dentro del proyecto..... | 8 |
| 3.4.1. | ArUco markers generator (Kalachev, s.f.)..... | 8 |
| 3.4.2. | HTML Minifier (kangax, s.f.)..... | 8 |
| 4. | Instalación de software y librerías | 9 |
| 4.1. | Instalación de Arduino IDE | 9 |
| 4.2. | Librerías necesarias para el proyecto | 12 |
| 4.3. | Instalación de la herramienta sketch upload | 16 |
| 5. | Etapas del proyecto..... | 17 |
| 5.1. | Etapa 0: Movimiento del vehículo | 17 |
| 5.2. | Etapa 1: Seguimiento de líneas | 17 |
| 5.3. | Etapa 2: Coordinación de recorrido a realizar | 17 |
| 5.4. | Etapa 3: Reconocimiento de ArUcos | 17 |
| 5.4.1. | Etapa 3.a [Descartada] : Análisis de ArUcos con modulo OV7670 | 17 |
| 5.4.2. | Etapa 3.b: Análisis de ArUcos con sensor de línea..... | 17 |
| 5.5. | Etapa 4: Comunicación con proyecto Satélite para RoboTown..... | 17 |
| 5.6. | Etapa 5 (Experimental): Control desde satélite | 17 |
| 6. | Implementación | 18 |
| 6.1. | Diagramas de la ciudad..... | 18 |
| 6.2. | Diagrama del prototipo | 19 |
| 6.3. | Código WeMos D1 R32 | 20 |
| 6.4. | Código para la página web..... | 29 |
| 6.5. | Código Python comunicación WebSocket..... | 32 |
| 7. | Referencias | 40 |
| 8. | Repositorio de GitHub..... | 40 |



Índice de figuras

| | |
|---|----|
| Figura 1: WeMos D1 R32 | 6 |
| Figura 2: Sensor de línea | 7 |
| Figura 3: Servo MG 996R | 7 |
| Figura 4: Imagen referencia ArUco | 8 |
| Figura 5: Imagen referencia HTML Minifier | 8 |
| Figura 6: Descargar Arduino IDE Paso 1 | 9 |
| Figura 7: Descargar Arduino IDE Paso 2 | 9 |
| Figura 8: Instalar Arduino IDE | 10 |
| Figura 9: Buscar Arduino IDE del Microsoft Store | 10 |
| Figura 10: Instalar Arduino IDE del Microsoft Store | 11 |
| Figura 11: Demo RoboTown | 18 |
| Figura 12: Esquema final de RoboTown | 18 |
| Figura 13: Diagrama del prototipo | 19 |
| Figura 14: Fotografía RoboTown | 19 |
| Figura 15: Librerías | 20 |
| Figura 16: Pagina Web Control Robot | 30 |
| Figura 17: Captura de pantalla del control mediante el ArUco 99 | 39 |



1. Palabras del autor

El presente informe sin duda es largo, pero es necesario para que las imágenes adjuntas se vean bien y para explicar un paso a paso a detalle.

Si bien es cierto, el proyecto no se logró completar en su totalidad, se logró avanzar bastante, por lo que se decidió dejarlo en este punto para que en un futuro se pueda retomar y mejorar.

OBSERVACIONES FINALES:

- Algo tiene el pin 5 de la WeMos que siempre me marcaba una lectura en alto.
- El robot no pudo completar la ruta, buscaba como llegar al ArUco, pero los ángulos no se calculaban bien. Por lo tanto, esto es algo que se debe mejorar. Usando sensores de línea o un giroscopio en el esp32, o incluso se podría buscando soluciones en el mismo código.
- Agregar la función de dibujo de ruta con el centro del ArUco móvil.
- Parte de la lógica de los sensores de los sensores de línea analizando y escaneando los ArUcos se explica dentro del documento con diagramas.
- Diría que podría ser mejor ocupar motores simples de esos de 3V, y un encoder para controlar y registrar información de cuantas vueltas se dieron, etc. Para guardar esa información en el Datalogger. Para realizar estas mejoras sería necesario cambiar el prototipo, utilizando el menor espacio posible, pero teniendo la fuente de alimentación necesaria.



2. Introducción

Este documento ilustra la implementación de un proyecto de investigación y desarrollo centrado en la concepción de un vehículo autónomo de robótica aplicada.

Para poder desarrollar el dispositivo, se decide dividir el proyecto en etapas, principalmente para comprender bien el proceso, en estas etapas se utilizaron diferentes dispositivos entre ellos una WeMos D1 R32 que usa de base un microcontrolador ESP32 como unidad central de procesamiento, respaldado por un conjunto de sensores de línea, una cámara de visión OV7670 y 2 servomotores.

2.1. Contexto de la necesidad a resolver

En el contexto actual de rápido avance tecnológico y transformación digital, la automatización y la robótica emergen como pilares fundamentales para el progreso de diversas industrias y sectores. La capacidad de diseñar y desplegar sistemas robóticos avanzados y automatizados no solo redefine la eficiencia y la precisión en la producción y los procesos industriales, sino que también se convierte en un factor crítico para mantener la competitividad en un mundo cada vez más globalizado y digitalizado.

La automatización y la robótica han trascendido su papel tradicional en la manufactura y la producción para abarcar una amplia gama de aplicaciones, desde la atención médica y la logística hasta la exploración espacial y la agricultura. La capacidad de desarrollar y aplicar soluciones robóticas innovadoras no solo reduce costos y aumenta la productividad, sino que también puede abordar desafíos complejos, como la atención médica a distancia, la mitigación de riesgos en entornos peligrosos y la preservación del medio ambiente.

Estar en la vanguardia de la automatización y la robótica implica más que una ventaja competitiva; es una necesidad imperativa en un mundo en constante cambio. Las organizaciones y los profesionales que lideran en este ámbito no solo están contribuyendo al desarrollo tecnológico, sino que también están preparando el terreno para una sociedad más avanzada y eficiente en todos los aspectos. En este contexto, el presente informe marca un paso significativo hacia la exploración y materialización de soluciones robóticas autónomas y versátiles que tienen el potencial de impulsar el futuro de múltiples industrias y sectores.

3. Documentación de Hardware

3.1. WeMos D1 R32

D1 R32 Board Pinout

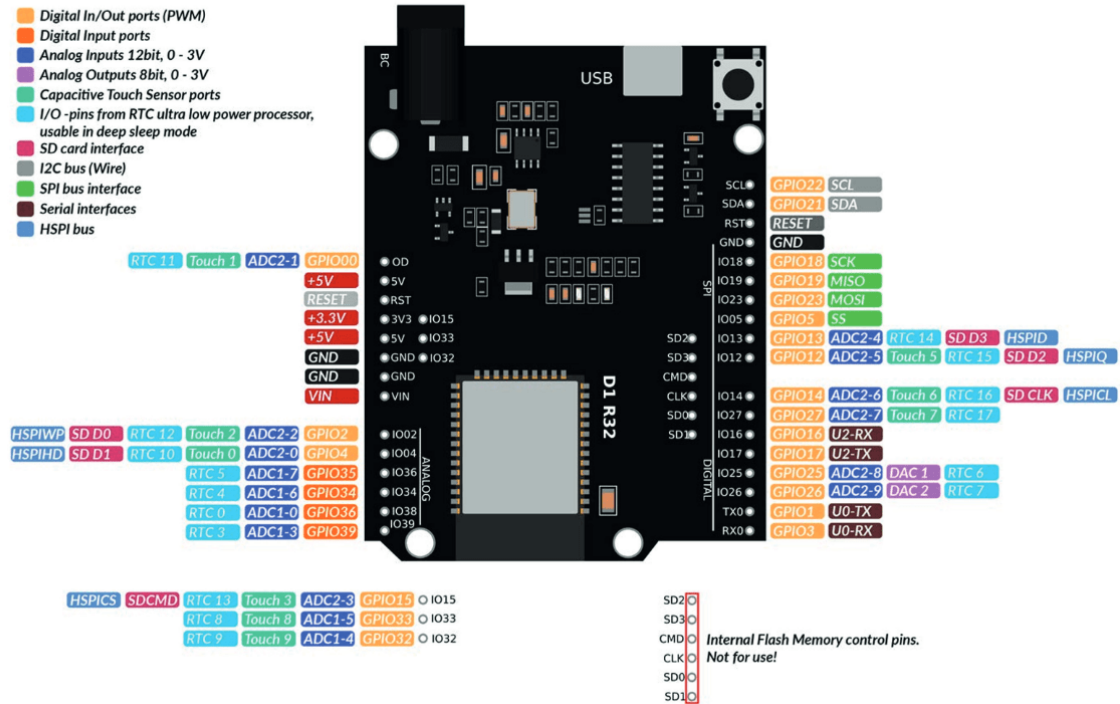


Figura 1: WeMos D1 R32

La placa WeMos D1 R32 (Villalpando, s.f.) es una tarjeta de desarrollo con forma de un Arduino Uno, está basada en el microcontrolador ESP32 WROOM 32 SMD el cual incorpora WiFi y Bluetooth.

3.2. Sensor de línea de 8 canales



Figura 2: Sensor de línea

El sensor de línea de 8 canales (paginaswebschile.com, s.f.) es un módulo que se conecta a una placa Arduino/Esp32 y permite detectar hasta 8 líneas en el suelo.

3.3. Servo MG 996R



Figura 3: Servo MG 996R

El Servo MG 996R (Altronics Chile SPA, s.f.) es un motor de rotación continua que se conecta a una placa Arduino/Esp32 y permite controlar la dirección de un vehículo.

3.4. Herramientas Online utilizadas dentro del proyecto

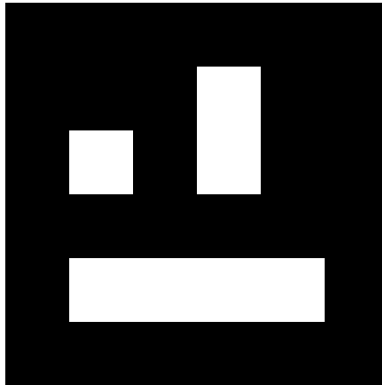
3.4.1. ArUco markers generator (Kalachev, s.f.)

ArUco markers generator!

Dictionary:

Marker ID:

Marker size, mm:



[Save](#) this marker as SVG, or [open](#) standard browser's print dialog to print or get the PDF.

Figura 4: Imagen referencia ArUco

3.4.2. HTML Minifier (kangax, s.f.)

HTML Minifier (v4.0.0)

```
<button style="visibility: hidden;"></button>
<a href="Retroceder"><button onmousedown="this.classList.add('pressed')" onmouseup="this.classList.remove('pressed')">Retroceder</button></a>
<button style="visibility: hidden;"></button>
</div>
</div>
<script>
</script>
</body>
</html>
```

Minify

```
<!doctypehtml><title>Control RoboTown</title><style>body{font-family:Arial,sans-serif;margin:0;padding:0;box-sizing:border-box}h1{font-family:Roboto,sans-serif;text-align:center;font-size:2.5em;font-weight:500;margin-bottom:20px}.container{display:flex;flex-direction:column;align-items:center}.column{display:flex;justify-content:center;flex-direction:column;align-items:center;width:100%;padding:20px}@media screen and (min-width:1080px){.container{flex-direction:row;justify-content:space-around}}.controls{margin:20px;display:grid;grid-template-columns:repeat(3,1fr);grid-gap:10px;width:60%;margin-top:20px;background-color:#f2f2f2;border-radius:10px;padding:20px}button{width:100%;aspect-ratio:1/1;padding:10px;border:none;border-radius:5px;background-color:#e0e0e0;box-shadow:2px 2px 5px rgba(0,0,0,.3);transition:transform .2s;background-color: #2s}button.pressed{background-color:#e0e0e0}button:hover{transform:translate(2px,2px)}button:active{transform:translate(5px,5px)}box-shadow:1px 1px 2px rgba(0,0,0,.3)}.google-forms-button{width:auto;height:auto;background-color:#f0f0f0;border-radius:5px;color:#202124;cursor:pointer;font-size:14px;font-weight:500;line-height:36px;padding:0 16px;text-align:center;text-decoration:none;text-transform:uppercase;transition:background-color 218ms;border-color 218ms,box-shadow 218ms}.google-forms-button:hover{transform:translate(2px,2px);background-color:#32cd32}.google-forms-button:active{background-color:#32cd32;transform:translate(5px,5px);box-shadow:0 1px 1px rgba(0,0,0,.1)}img{width:100%;height:auto;max-width:600px;border-radius:10px}form{display:flex;justify-content:center;width:auto;min-width:350px;max-width:550px;margin-top:20px;background-color:#f2f2f2;border-radius:10px;padding:20px}form label{display:flex;align-items:center;margin:5px}form input{border-radius:5px;border:1px solid #e0e0e0;margin:5px}</style><div class=Titulo><h1>Control Automovil de RoboTown</h1></div><div class=container><div class=column><div id=image><img alt=Imagen de ciudad RoboTown src=/imagen.jpg style=width:100%;height:auto></div></div><div class=column><form action=/Enviar id=myForm method=post><label for=rutaInputRuta:</label> <input name=rutaText placeholder=E3:1-2-3-7-10><br><input class=google-forms-button onclick=sendData() type=submit value=Enviar></form><div class=controls><a href=Izquierda><button
```

Original size: 5,597. Minified size: 3,403. Savings: 2,194 (39.20%).

Figura 5: Imagen referencia HTML Minifier

4. Instalación de software y librerías

4.1. Instalación de Arduino IDE

Se debe instalar dos IDE's para Arduino, se recomienda el uso del más actualizado para el desarrollo del código y el más antiguo solo para usar la "Esp32 Sketch Data Upload" que permite subir archivos a la memoria interna de la Esp32.

Arduino IDE para el desarrollo del código:

1. Se descargará el Arduino IDE (Arduino, s.f.) en la página oficial de Arduino.

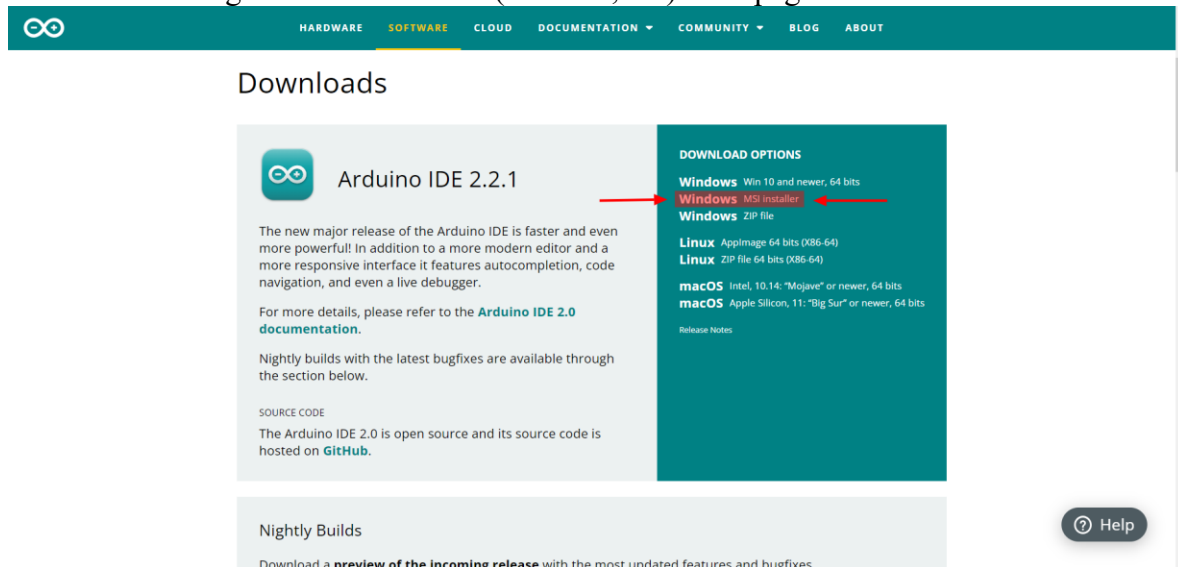


Figura 6: Descargar Arduino IDE Paso 1

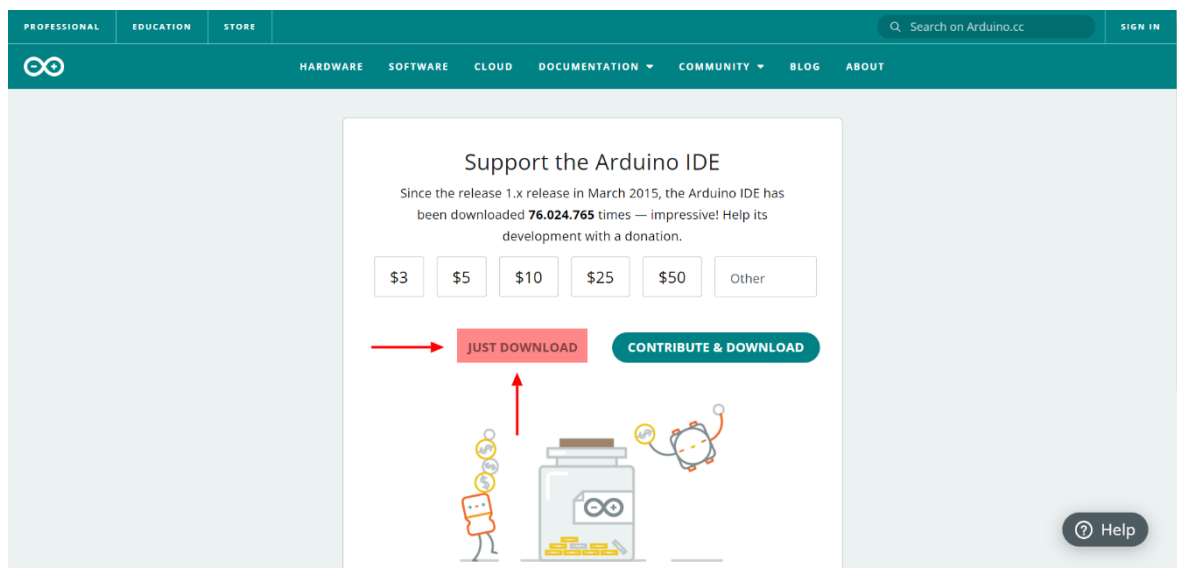


Figura 7: Descargar Arduino IDE Paso 2

2. Luego el archivo descargado se le da clic derecho e instalar.

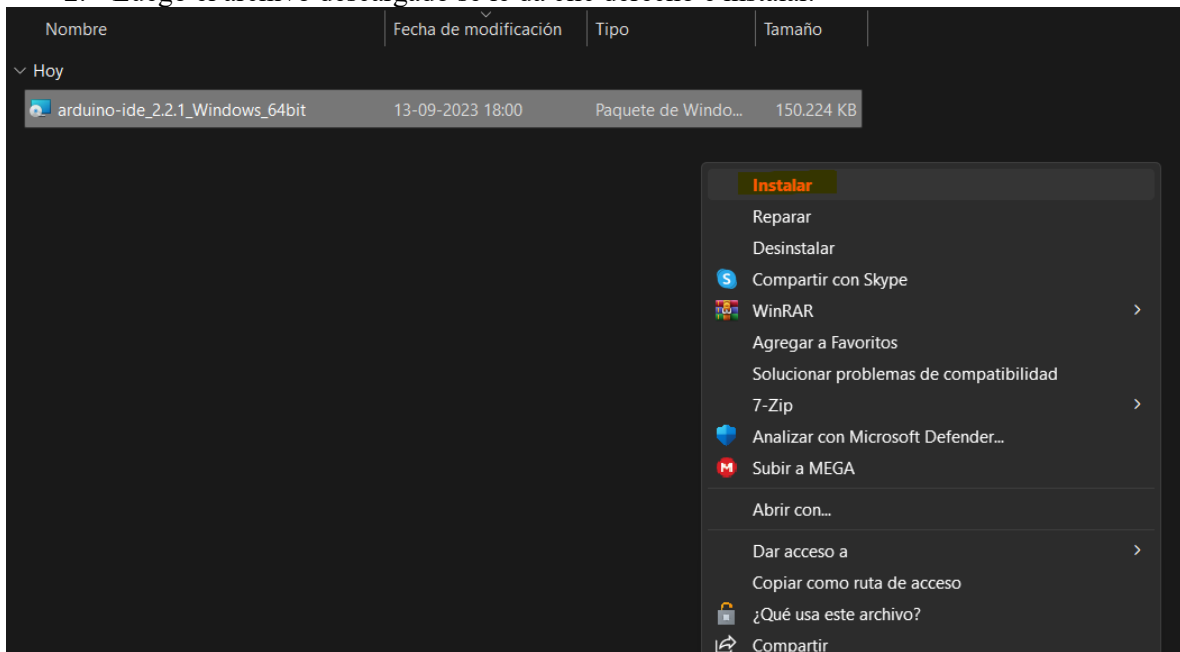


Figura 8: Instalar Arduino IDE

Arduino IDE para subir archivos a la memoria interna de la Esp32:

1. Se buscará el Arduino IDE en la Microsoft Store.

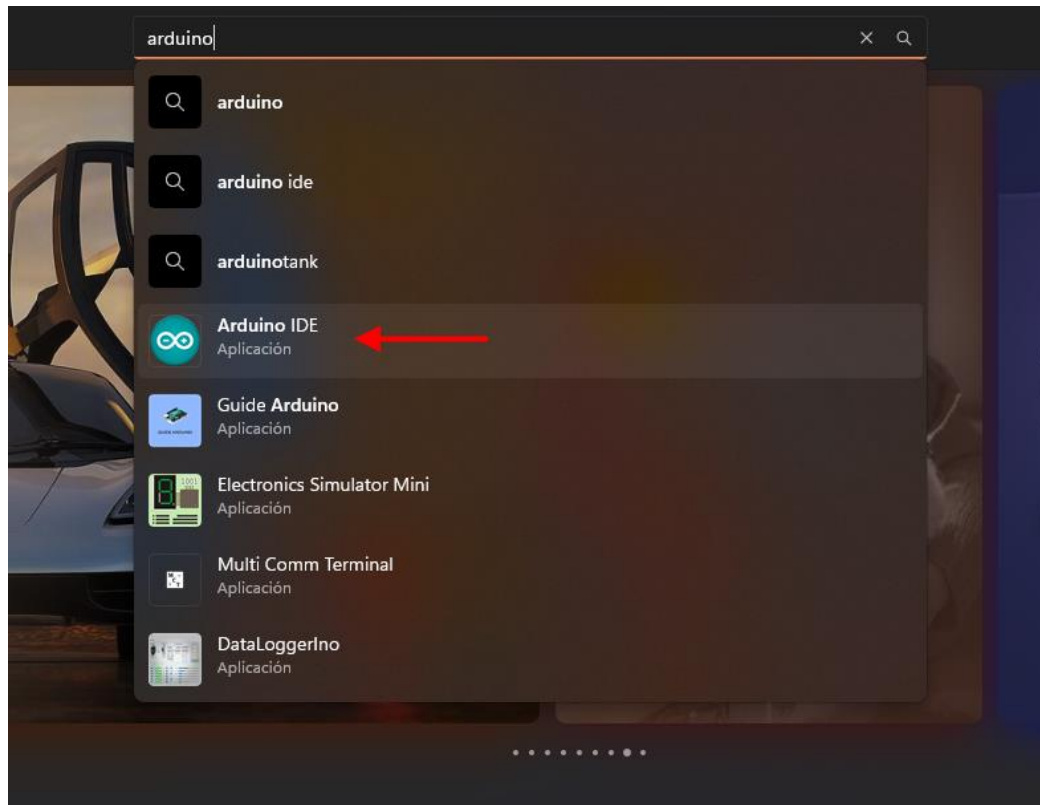


Figura 9: Buscar Arduino IDE del Microsoft Store

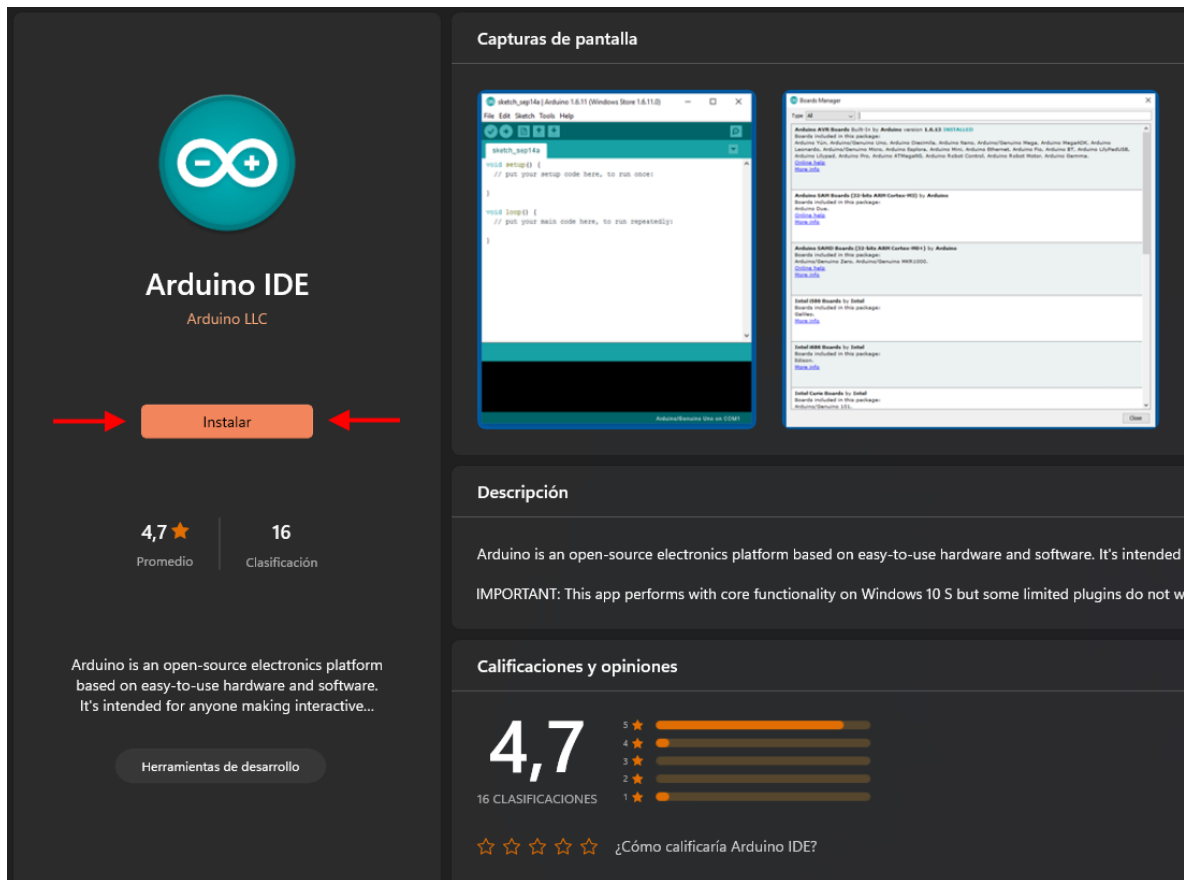
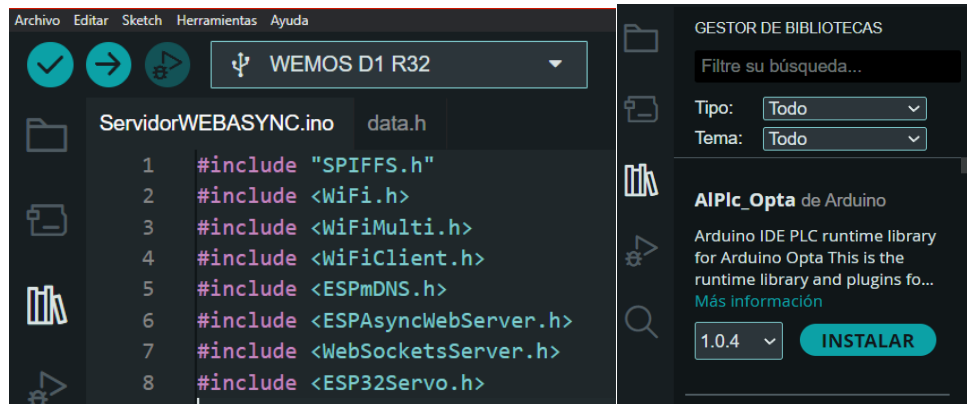


Figura 10: Instalar Arduino IDE del Microsoft Store

4.2. Librerías necesarias para el proyecto

Para la WeMos D1 R32:

Para descargar la gran mayoría de librerías podemos utilizar el gestor de bibliotecas del mismo IDE 2.2.1



1. Para instalar SPIFFS (se utiliza el core de esp32) para ello utilizamos el gestor de placas dentro del IDE. Y buscamos “esp32 espressif” y damos a instalar.

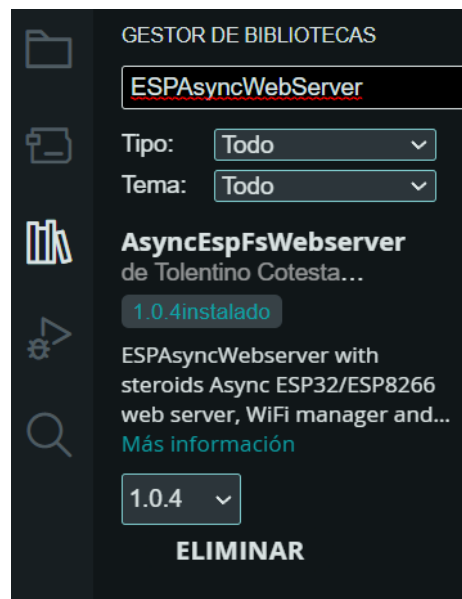


*Obs: ahora cuando conectamos la placa al computador podemos elegir la placa WeMos D1 R32 para cargar los códigos.

2. Librerías de control de WiFi, WiFiMulti y WiFiClient, buscamos “WiFi” dentro del gestor de bibliotecas del IDE



3. Para ESPmDNS, esta librería está incluido dentro del core de esp32 por lo que con realizar la instalación para el SPIFFS debería bastar.
4. Luego la librería ESPAsyncWebServer la encontramos dentro del gestor de bibliotecas.



5. Para lo de los WebSocketsServer se busca “WebSockets” dentro del gestor de bibliotecas.



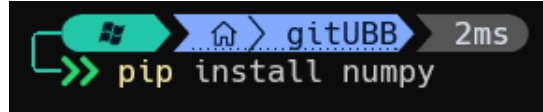
6. Y para finalizar la librería de Esp32Servo la encontramos dentro del gestor.



Para Python:

1. Descargamos la librería de numpy, el cual nos permite hacer operaciones matemáticas con mayor facilidad, utilizando la terminal/console y escribimos:

pip install numpy



2. Descargamos la librería para el WebSocket.

pip install websocket



3. Descargamos una librería para el control de eventos de teclado.

pip install keyboard



4. Descargamos la librería de opencv para controlar las cámaras y

pip install opencv-python



5. Y descargamos una versión muy específica por temas de incompatibilidad, pues Python cambia de versiones muy rápido que generan problemas con otras librerías.

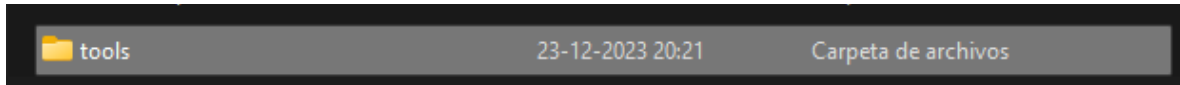
pip install opencv-contrib-python==4.6.0.66



4.3. Instalación de la herramienta sketch upload

Esta herramienta se utiliza en conjunto con SPIFFS o algún gestor similar, la herramienta se debe instalar en el IDE 1.8, pues el IDE 2.2.1 no tiene soporte para esta herramienta (Random Nerd Tutorials, s.f.).

1. Primero nos dirigimos a la carpeta raíz del Arduino y creamos una carpeta llamada “tools”. Como referencia la carpeta raíz puede ser donde guardas los proyectos del Arduino IDE.



2. Luego nos descargamos la última versión de la herramienta de ESP32FS desde (Dev, s.f.)



3. Nos llevamos el archivo a la carpeta “tools” y lo descomprimos.

| Nombre | Fecha de modificación | Tipo | Tamaño |
|-------------|-----------------------|---------------------|--------|
| ESP32FS | 23-12-2023 20:21 | Carpeta de archivos | |
| ESP32FS-1.1 | 23-12-2023 20:20 | Carpeta comprimi... | 8 KB |

4. Debería quedar algo así.

| Arduino > tools > ESP32FS > tool | | | |
|----------------------------------|-----------------------|---------------------|--------|
| Nombre | Fecha de modificación | Tipo | Tamaño |
| esp32fs | 23-12-2023 20:21 | Executable Jar File | 8 KB |

*Obs: luego lo utilizaremos para cargar los datos de la página web [31].



5. Etapas del proyecto

5.1. Etapa 0: Movimiento del vehículo

Se ocupa dos servomotores MG 996R y una rueda omnidireccional para lograr el movimiento del vehículo, el cual se controla usando la comunicación Serial.

5.2. Etapa 1: Seguimiento de líneas

Para esta etapa se supone que se debería ocupar el sensor de línea QTR de 8 canales, sin embargo, no funciona como se esperaba (ACTUALIZAR).

5.3. Etapa 2: Coordinación de recorrido a realizar

Para esta etapa se decidió ocupar la WeMos como un servidor Web.

5.4. Etapa 3: Reconocimiento de ArUcos

5.4.1. Etapa 3.a [Descartada] : Análisis de ArUcos con modulo OV7670

Objetivos:

1. Comprender el funcionamiento del módulo OV7670 (altronics, s.f.).
2. Aprender sobre los ArUcos (UCO, s.f.).
3. Analizar el funcionamiento de los ArUcos con el módulo OV7670.

Observación: Los objetivos 1 y 2 se lograron satisfactoriamente, sin embargo, el objetivo 3 no se logró debido a que el módulo OV7670 tiene muchos problemas al no poseer un buffer lo cual retrasa la sincronización, lo que genera imágenes borrosas, impidiendo el reconocimiento de los ArUcos, por lo que se decidió cambiar de modulo a un sensor de línea de 8 canales.

5.4.2. Etapa 3.b: Análisis de ArUcos con sensor de línea

Objetivos:

1. Comprender el funcionamiento del sensor de línea (paginaswebschile.com, s.f.).
2. Escanear ArUcos con el sensor de línea.
3. Analizar ArUco escaneado e interpretar la información.

Observación: Se creó solo la lógica, pero no se logra probar ni implementar por temas de hardware (no se pudo hacer funcionar los sensores de línea de 8 canales).

5.5. Etapa 4: Comunicación con proyecto Satélite para RoboTown

En esta etapa se realiza una comunicación bidireccional, entre WeMos y un programa en Python, utilizando WebSockets.

5.6. Etapa 5 (Experimental): Control desde satélite

Con otro practicante desarrollamos unos códigos para controlar el robot desde el computador dependiendo del ángulo que nos arroja el ArUco 99.

6. Implementación

6.1. Diagramas de la ciudad

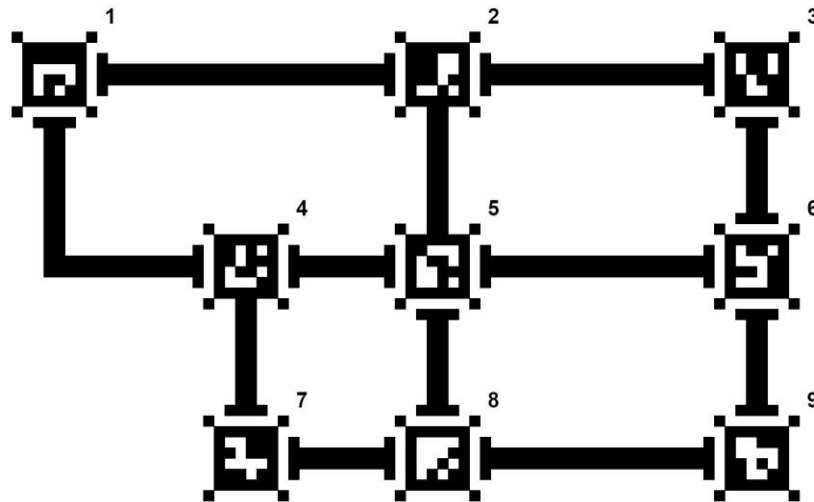


Figura 11: Demo RoboTown

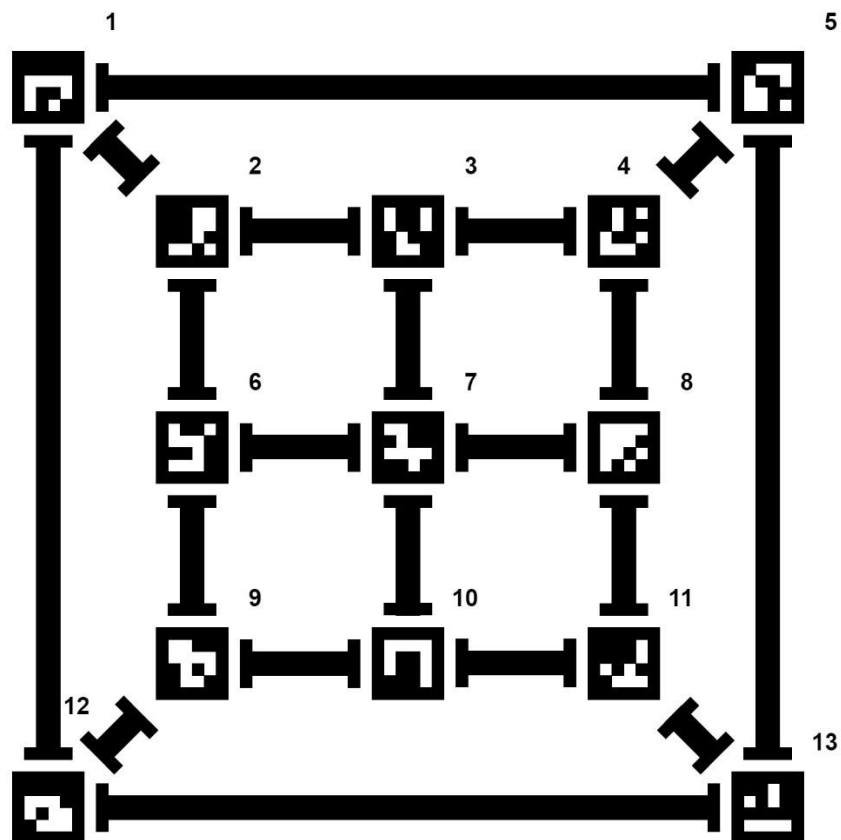


Figura 12: Esquema final de RoboTown

6.2. Diagrama del prototipo

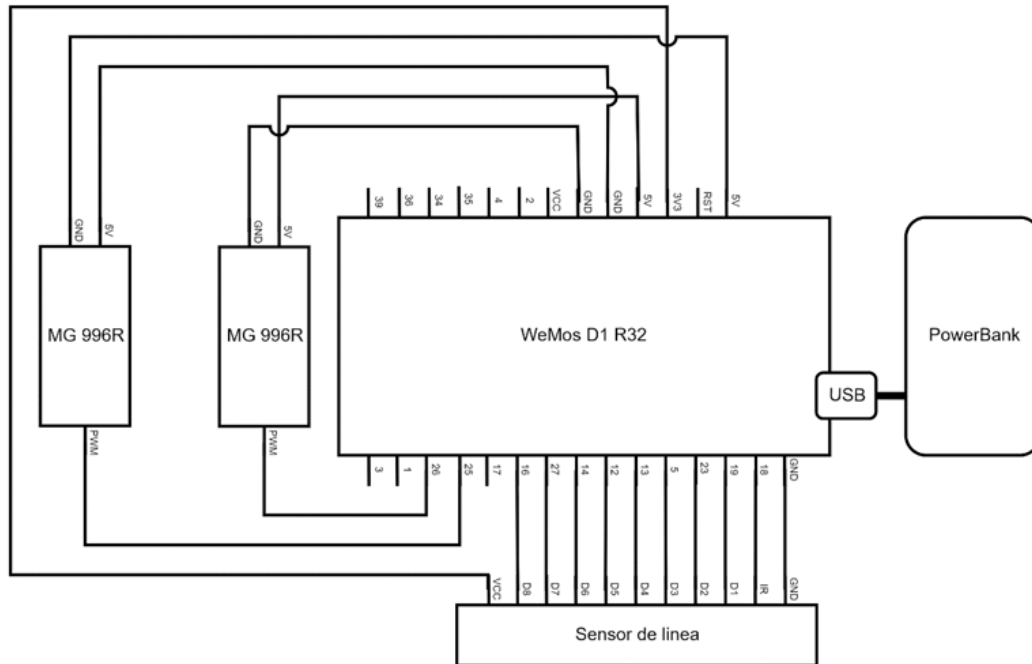


Figura 13: Diagrama del prototipo

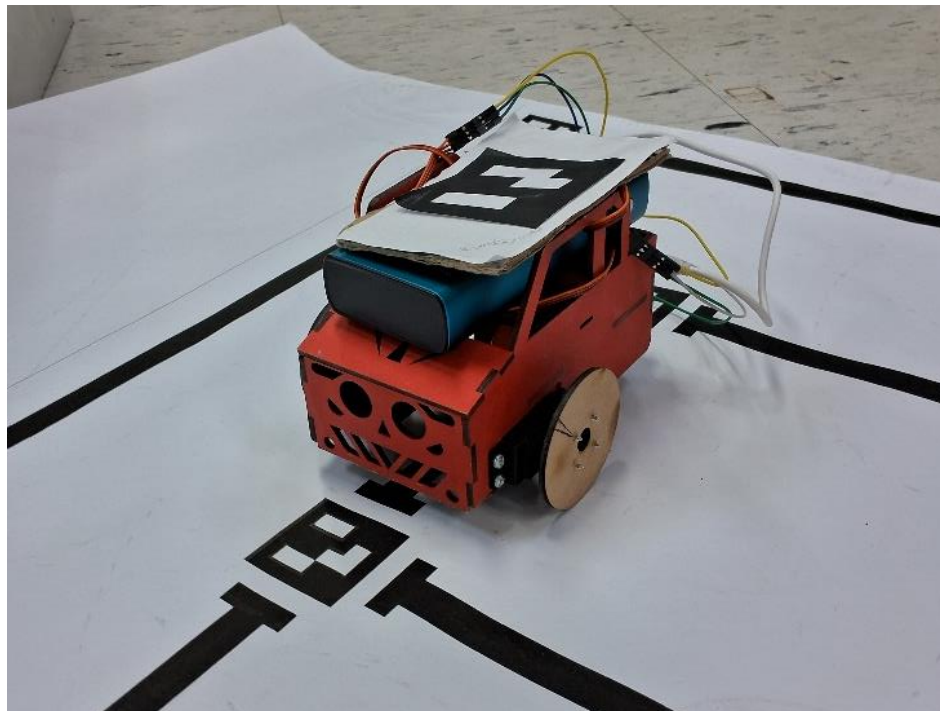


Figura 14: Fotografia RoboTown

6.3. Código WeMos D1 R32

```
1  #include "SPIFFS.h"
2  #include <WiFi.h>
3  #include <WiFiMulti.h>
4  #include <WiFiClient.h>
5  #include <ESPmDNS.h>
6  #include <ESPAsyncWebServer.h>
7  #include <WebSocketsServer.h>
8  #include <ESP32Servo.h>
```

Figura 15: Librerías

1. Se importa un archivo con las credenciales de las conexiones WiFi y se crea un mDNS para facilitar la conexión a futuro en WiFi's de prueba. Esto pues normalmente el IP puede cambiar dependiendo del proveedor de la conexión.

```
11 // Credenciales para WiFi
12 #include "data.h"
13 WiFiMulti wifiMulti;
14 const char* mDNS = "robotown";
```

2. Se define un servidor asincrónico por el puerto 80, para que cuando un cliente se conecte a ese servidor se puedan realizar las peticiones http GET y POST.
3. Además de definir un servidor websocket por el puerto 81, para a futuro conectarse a un código Python y tener una comunicación bidireccional.

```
16 // Servidores
17 AsyncWebServer server(80);
18 WebSocketsServer websocket = WebSocketsServer(81);
```

4. Se inician los pines de los servos y los objetos servomotores.

```
20 // Pines ServoMotores
21 const int servoPin1 = 25;
22 const int servoPin2 = 26;
23 Servo servoM1;
24 Servo servoM2;
```

5. Se definen los pines para el sensor de línea y su infrarrojo, además de definir variables para manejar el tiempo (no se utiliza esta parte porque el sensor de línea fallo, revisar a futuro).

```
26 // Pines Sensor de línea
27 int pinLedIR = 18;
28 int sensorPins[8] = {19, 23, 5, 13, 12, 14, 27, 16};
29 unsigned long tiempoInicial;
30 unsigned long tiempoActual;
```

6. Se crea una matriz para guardar lo escaneado por el sensor de línea. (no se utiliza)

```
32 // Matriz para guardar ArUco
33 int arucoEscaneado[4][4] = {
34     {0, 0, 0, 0},
35     {0, 0, 0, 0},
36     {0, 0, 0, 0},
37     {0, 0, 0, 0}
38 };
```

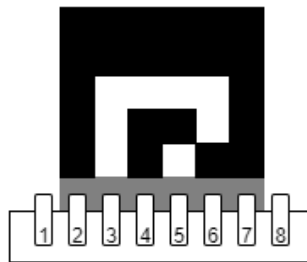
7. Se agrega la información de los ArUcos de 4x4 desde el 0 al 13, suponiendo que 0 es blanco y 1 es negro.

```
40 // Información ArUcos
41 const int ArUcos[14][4][4] = {
42     {
43         {0, 1, 0, 0},
44         {1, 0, 1, 0},
45         {1, 1, 0, 0},
46         {1, 1, 0, 1}
47     }, // 0
48     {
49         {1, 1, 1, 1},
50         {0, 0, 0, 0},
51         {0, 1, 1, 1},
52         {0, 1, 0, 1}
53     }, // 1
54     {
55         {1, 1, 0, 0},
56         {1, 1, 0, 0},
57         {1, 1, 0, 1},
58         {0, 0, 1, 0}
59     }, // 2
60     {
61         {0, 1, 1, 0},
62         {0, 1, 1, 0},
63         {1, 0, 1, 1},
64         {1, 0, 0, 1}
65     }, // 3
66     {
67         {1, 0, 1, 0},
68         {1, 0, 1, 1},
69         {0, 1, 1, 0},
70         {0, 0, 0, 1}
71     }, // 4
72     {
73         {1, 0, 0, 0},
74         {0, 1, 1, 0},
75         {0, 0, 1, 1},
76         {0, 0, 1, 0}
77     }, // 5
78     {
79         {0, 1, 1, 0},
80         {0, 0, 0, 1},
81         {1, 0, 1, 1},
82         {0, 0, 0, 1}
83     }, // 6
84     {
85         {0, 0, 1, 1},
86         {1, 0, 1, 1},
87         {0, 0, 0, 0},
88         {1, 1, 0, 1}
89     }, // 7
90     {
91         {1, 0, 0, 0},
92         {0, 1, 1, 0},
93         {0, 0, 1, 1},
94         {0, 0, 1, 0}
95     }, // 8
96     {
97         {0, 1, 1, 0},
98         {0, 0, 0, 1},
99         {1, 0, 1, 1},
100        {0, 0, 0, 1}
101     }, // 9
102     {
103         {0, 0, 1, 1},
104         {1, 0, 1, 1},
105         {0, 0, 0, 0},
106         {1, 1, 0, 1}
107     }, // 10
108     {
109         {0, 0, 0, 0},
110         {0, 0, 0, 1},
111         {0, 0, 1, 0},
112         {0, 1, 0, 1}
113     }, // 11
114     {
115         {0, 0, 1, 1},
116         {0, 0, 0, 1},
117         {1, 0, 1, 0},
118         {1, 0, 0, 1}
119     }, // 12
120     {
121         {1, 1, 0, 1},
122         {0, 1, 0, 1},
123         {1, 1, 1, 1},
124         {0, 0, 0, 0}
125     }, // 13
126 };
```

8. Se crea una función para escanear el ArUco, la cual debe controlar la velocidad de los servomotores. Esta función se debería llamar cuando los sensores del 2 al 7 reciben un valor 1, luego los sensores 3 a 6 deberían guardar los valores registrados en arucoEscaneado.

```

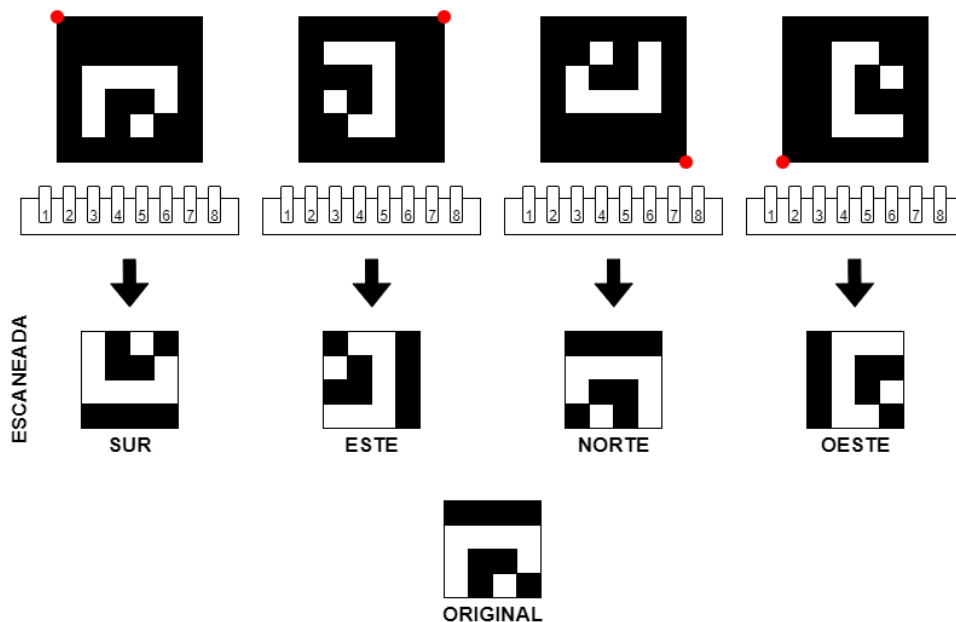
130 void escanear() {
131     return;
132 }
  
```



9. Se crea función para analizar el ArUco escaneado utilizando la referencia de los arucos previamente registrados (aunque esta función no se realizó, se tiene una lógica que tal vez podría funcionar).

```

134 void analizar( ) {
135     return;
136 }
  
```



*Obs: Depende de la dirección el valor que tendrá la matriz escaneada, por lo que es necesario obtener el ArUco original y compararlo 4 veces de formas diferentes.

10. Se crean funciones de movimiento de los servomotores (como actualización a futuro se podría poner aparte de tiempo, una velocidad de giro, ejemplo: 5 velocidades y dependiendo de ese valor poner condicionales).

```
139 void avanzar(int tiempo) {
140     Serial.println("Avanzar");
141     servoM1.write(180);
142     servoM2.write(0);
143     delay(tiempo);
144 }
145
146 void derecha(int tiempo) {
147     Serial.println("Derecha");
148     servoM1.write(180);
149     servoM2.write(180);
150     delay(tiempo);
151 }
152
153 void izquierda(int tiempo) {
154     Serial.println("Izquierda");
155     servoM1.write(0);
156     servoM2.write(0);
157     delay(tiempo);
158 }
```

```
160 void retroceder(int tiempo) {
161     Serial.println("Retroceder");
162     servoM1.write(0);
163     servoM2.write(180);
164     delay(tiempo);
165 }
166
167 void detener(){
168     Serial.println("Detener");
169     servoM1.write(90);
170     servoM2.write(90);
171 }
```

11. Se crea una función para controlar los eventos del websocket, dependiendo de si el cliente se desconecta, se conecta o recibe datos desde el código en Python.

```
173 // -----FUNCION--COMUNICACION--WEMOS--TO--PYTHON-----
174 void websocketEvent(uint8_t num, WStype_t type, uint8_t * payload, size_t length){
175     switch(type) {
176         case WStype_DISCONNECTED:
177             Serial.printf("[%u] Disconnected!\n", num);
178             break;
179         case WStype_CONNECTED:
180             Serial.printf("[%u] Connected from %s url: %s\n", num, websocket.remoteIP(num).toString().c_str(), payload);
181             break;
182
183         case WStype_TEXT:
184             Serial.printf("[%u] get Text: %s\n", num, payload);
185             String msg = String((char *)payload);
186
187             char* token;
188             char* rest = (char*)payload;
189
190             int tiempo;
191             String instruccion;
192             int i = 0;
193             while ((token = strtok_r(rest, " ", &rest))){
194                 // token ahora contiene una parte del payload separada por espacios
195                 Serial.println(token);
196                 if (i == 0) { // Si es el primer token, lo guardamos como una instruccion
197                     instruccion = token;
198                 }
199             }
200         }
```

```
199         if (i == 1) { // Si es el segundo token, lo guardamos como tiempo
200             tiempo = atoi(token); // Convertir el token a un entero
201         }
202         i++;
203     }
204     Serial.println(tiempo);
205     if (instruccion.equalsIgnoreCase("avanzar"))
206     {
207         avanzar(tiempo);
208         detener();
209     }
210     else if (instruccion.equalsIgnoreCase("retroceder"))
211     {
212         retroceder(tiempo);
213         detener();
214     }
215     else if (instruccion.equalsIgnoreCase("derecha"))
216     {
217         derecha(tiempo);
218         detener();
219     }
220     else if (instruccion.equalsIgnoreCase("izquierda"))
221     {
222         izquierda(tiempo);
223         detener();
224     }
225     break;
226 }
227 }
```

12. Se crea la función setup para cuando se inician las comunicaciones, se inicia comunicación Serial con una frecuencia de 115200. Luego se inician los servos con los pines dados anteriormente y se inicia la memoria interna no volátil de la WeMos.

```
230 void setup() {
231     // Comunicacion Serial
232     Serial.begin(115200);
233
234     // ServoMotores
235     servoM1.attach(servoPin1);
236     servoM2.attach(servoPin2);
237
238     // Memoria Interna
239     if(!SPIFFS.begin(true)){
240         Serial.println("An Error has occurred while mounting SPIFFS");
241         return;
242     }
```


13. Se añaden las credenciales desde “data.h”, para tener varias conexiones WiFi’s para conectarse, se define el WiFi como estático para obtener una IP estática desde el proveedor y se muestran los datos por la comunicación Serial.

```
244 // Conexión WiFi
245 wifiMulti.addAP(ssid_1, password_1);
246 wifiMulti.addAP(ssid_2, password_2);
247 wifiMulti.addAP(ssid_3, password_3);
248
249 WiFi.mode(WIFI_STA);
250 Serial.print("Conectando a Wifi ");
251 while (wifiMulti.run(3000) != WL_CONNECTED) {
252     Serial.print(".");
253 }
254 Serial.println("");
255 Serial.println("Conectado");
256 Serial.print("SSID: ");
257 Serial.print(WiFi.SSID());
258 Serial.print("IP: ");
259 Serial.println(WiFi.localIP());
```

14. Se inicia el servicio de mDNS para que en algunas conexiones se facilite la conexión a la pagina web utilizando “robotown.local” en vez de una IP.

```
261 // Inicio mDNS
262 if (!MDNS.begin("robotown")) {
263     Serial.println("Error configurando mDNS!");
264     return;
265 }
266 Serial.print("mDNS: ");
267 Serial.print(mDNS);
268 Serial.println(" configurado");
269 Serial.print("Ingresa al link: ");
270 Serial.print(mDNS);
271 Serial.println(".local");
272 Serial.println("¡¡NO SIEMPRE FUNCIONA EL mDNS, DEPENDE DEL PROVEEDOR!!");
273 MDNS.addService("http", "tcp", 80);
```

15. Se crean los request para que en la raíz cargue el Control.html y se le pasa la imagen.jpg desde la memoria no volátil con SPIFFS.

```
275 // Servidor Web Asincrono
276 server.on("/", HTTP_GET, [](AsyncWebServerRequest * request) {
277     request->send(SPIFFS, "/Control.html", String(), false);
278 });
279
280 server.on("/imagen.jpg", HTTP_GET, [](AsyncWebServerRequest * request) {
281     request->send(SPIFFS, "/imagen.jpg", String(), false);
282 });
```

16. Se crean mas request para dar una instrucción de movimiento utilizando en el Control.html botones que generan los HTTP GET además de redirigir a la página inicial.

```
284 server.on("/IzquierdaN", HTTP_GET, [](AsyncWebServerRequest * request) {
285     izquierda(700);
286     detener();
287     request->send(SPIFFS, "/Control.html", String(), false);
288 });
289
290 server.on("/Avanzar", HTTP_GET, [](AsyncWebServerRequest * request) {
291     avanzar(1000);
292     request->send(SPIFFS, "/Control.html", String(), false);
293 });
294
295 server.on("/DerechaN", HTTP_GET, [](AsyncWebServerRequest * request) {
296     derecha(1050);
297     detener();
298     request->send(SPIFFS, "/Control.html", String(), false);
299 });
301 server.on("/Izquierda", HTTP_GET, [](AsyncWebServerRequest * request) {
302     izquierda(1000);
303     request->send(SPIFFS, "/Control.html", String(), false);
304 });
305
306 server.on("/Detener", HTTP_GET, [](AsyncWebServerRequest * request) {
307     detener();
308     request->send(SPIFFS, "/Control.html", String(), false);
309 });
310
311 server.on("/Derecha", HTTP_GET, [](AsyncWebServerRequest * request) {
312     derecha(1000);
313     request->send(SPIFFS, "/Control.html", String(), false);
314 });
315
316 server.on("/Retroceder", HTTP_GET, [](AsyncWebServerRequest * request) {
317     retroceder(1000);
318     request->send(SPIFFS, "/Control.html", String(), false);
319 });
```

17. Se crea un request para obtener desde el cliente (usuario que envió los datos utilizando el botón verde de la página web) los datos ingresados el rutaText para luego enviarlo por el websocket al código en Python y procesar las instrucciones, además de redirigir a la página inicial.

```
321     server.on("/Enviar", HTTP_POST, [])(AsyncWebServerRequest * request) {
322         if(request->hasParam("rutaText", true))
323         {
324             String msgRuta = request->getParam("rutaText", true)->value();
325             Serial.print("Ruta: ");
326             Serial.println(msgRuta);
327
328             // Obtener la dirección IP del cliente
329             IPAddress remote_ip = request->client()->remoteIP();
330             Serial.print("Cliente IP: ");
331             Serial.println(remote_ip);
332
333             // Enviar el mensaje al cliente correspondiente
334             for(uint8_t i = 0; i < websocket.connectedClients(); i++) {
335                 IPAddress client_ip = websocket.remoteIP(i);
336                 if(client_ip == remote_ip) {
337                     websocket.sendTXT(i, msgRuta.c_str());
338                     break;
339                 }
340             }
341         }
342         request->send(SPIFFS, "/Control.html", String(), false);
343     });
```

18. Se inicia el servidor asincrónico con los request creados anteriormente.
19. Se inicia el websocket con un evento para comprobar la comunicación por el puerto 81.

```
345     // Inicio del servidor
346     server.begin();
347
348     // Inicio del websocket
349     websocket.begin();
350     websocket.onEvent(webSocketEvent);
```

20. Se inician los pines del sensor de línea como entrada de datos, además de iniciar el infrarrojo como salida y en alto. Para finalizar el setup se registra un tiempo inicial (que podría usarse en los sensores de línea).

```
352 // Se inician los pines del sensor como entrada de datos
353 for (int i = 0; i < 8; i++)
354 {
355     pinMode(sensorPins[i], INPUT);
356 }
357
358 // Se inicia el IR del sensor como salida en alto
359 pinMode(pinLedIR, OUTPUT);
360 digitalWrite(pinLedIR, HIGH);
361
362 // Se inicia un tiempo
363 tiempoInicial = millis();
364 }
```

21. En la función loop se registra el tiempo actual (que se podría utilizar para controlar la información de los sensores de línea sin acudir a un delay e interrumpir la comunicación del socket) mientras se ejecuta el websocket en un loop recibiendo y enviando datos.

```
367 void loop(){
368     tiempoActual = millis();
369     // Agregar logica de los sensores de linea
370     websocket.loop();
371 }
```

6.4. Código para la página web

1. Se crea un archivo Control.html con un style con código CSS (se podría utilizar un archivo style.css) y se crea un título para mostrarle al usuario

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  |   <title>Control RoboTown</title>
5  >   <style>...
134  |   </style>
135  </head>
136  <body>
137  |   <div class="Titulo">
138  |       <h1>Control Automovil de RoboTown</h1>
139  |   </div>
```

2. Se crea un container para posicionar una imagen de la ciudad RoboTown en una columna izquierda y el control a la derecha y, también hacer mediante CSS hacer la página responsiva dependiendo del dispositivo.

```
140  <div class="container">
141  |
142  |   <div class="column">
143  |       <div id="image">
144  |           
146  |       </div>
147  |   </div>
```

3. En la otra columna se agrega un mini formulario para enviar la ruta a la WeMos.

```
148  |   <div class="column">
149  |       <form id="myForm" action="/Enviar" method="post">
150  |           <label for="rutaInput">Ruta:</label>
151  |           <input type="text" name="rutaText" placeholder="EJ:1-2-3-7-10"><br>
152  |           <input class="google-forms-button" type="submit" value="Enviar"
153  |               onclick="sendData()">
154  |       </form>
```

4. Además de crear un control con varios botones con diferentes instrucciones de movimiento.

```

156 <div class="controls">
157
158 <a href="IzquierdaN"><button onmousedown="this.classList.add
('pressed')" onmouseup="this.classList.remove('pressed')">Izquierda
90</button></a>
159
160 <a href="Avanzar"><button onmousedown="this.classList.add('pressed')
" onmouseup="this.classList.remove('pressed')">Avanzar</button></a>
161
162 <a href="DerechaN"><button onmousedown="this.classList.add('pressed')
" onmouseup="this.classList.remove('pressed')">Derecha 90</button></
a>
163
164 <a href="Izquierda"><button onmousedown="this.classList.add
('pressed')" onmouseup="this.classList.remove('pressed')">Izquierda</
button></a>
166 <a href="Detener"><button onmousedown="this.classList.add('pressed')
" onmouseup="this.classList.remove('pressed')">Detener</button></a>
167
168 <a href="Derecha"><button onmousedown="this.classList.add('pressed')
" onmouseup="this.classList.remove('pressed')">Derecha</button></a>
169
170 <button style="visibility: hidden;"></button>
171
172 <a href="Retroceder"><button onmousedown="this.classList.add
('pressed')" onmouseup="this.classList.remove('pressed')
">Retroceder</button></a>
173
174 <button style="visibility: hidden;"></button>
175 </div>
176 </div>
177 </div>
179 <script>
180
181 </script>
182 </body>
183 </html>

```

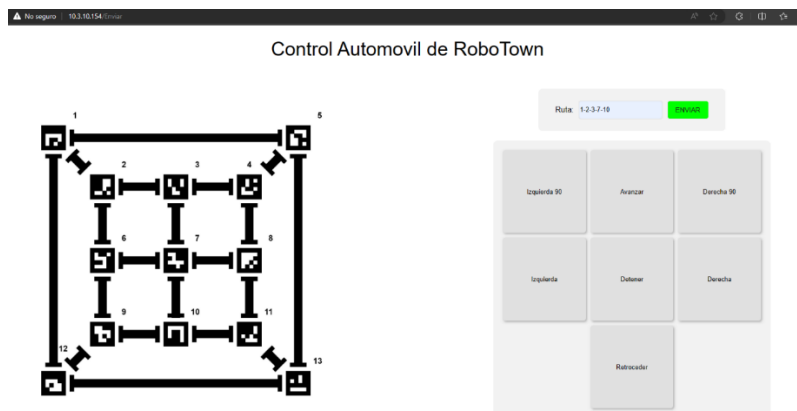
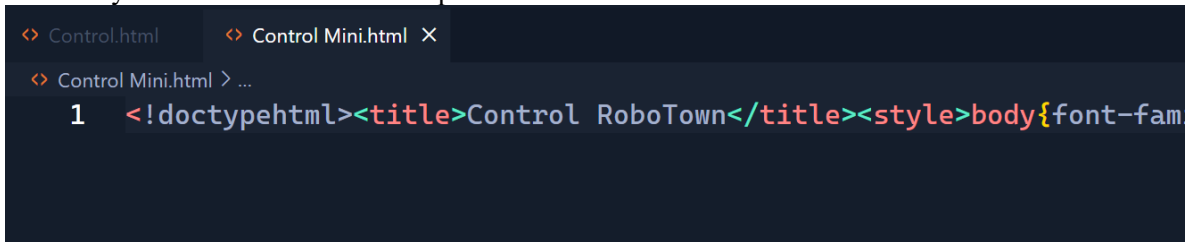


Figura 16: Pagina Web Control Robot

- Como paso opcional se podría minificar la el archivo.html, para ahorrar memoria interna utilizando (kangax, s.f.) la cual es una herramienta online para eliminar todos los espacios y tabulaciones innecesarias para HTML.



```

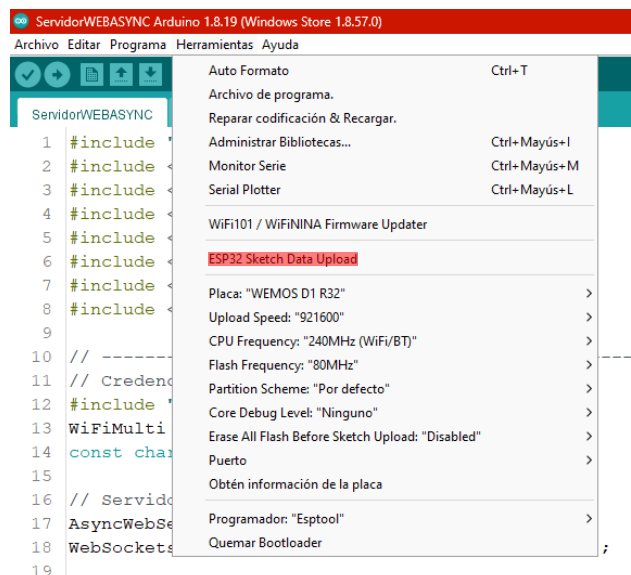
1 <!doctypehtml><title>Control RoboTown</title><style>body{font-fam
  
```

- Creamos una carpeta llamada data dentro del proyecto, y pegamos el Control.html e imagen.jpg.

| Nombre | Fecha de modificación | Tipo | Tamaño |
|-----------------|-----------------------|-----------------------|--------|
| data | 12-01-2024 0:36 | Carpeta de archivos | |
| data | 12-01-2024 0:08 | Archivo de origen ... | 1 KB |
| ServidorWEBASYN | 12-01-2024 0:08 | Archivo INO | 10 KB |

| Nombre | Fecha de modificación | Tipo | Tamaño |
|---------|-----------------------|--------------------|--------|
| Control | 12-01-2024 0:20 | Opera GX Web Do... | 6 KB |
| imagen | 06-01-2024 23:04 | Archivo JPG | 93 KB |

- Abrimos la versión 1.8 de Arduino IDE para utilizar la herramienta descargada anteriormente, para luego cargar los datos del proyecto dentro de la memoria interna de la WeMos.



6.5. Código Python comunicación WebSocket

1. Se importan las librerías necesarias dentro de un archivo main.py

```
1  import numpy as np
2  import websocket
3  import time
4  import threading
5  import numpy as np
6  import keyboard
7  import cv2
8  from cv2 import aruco
9  from collections import deque
10 from queue import Queue
```

2. Se define la IP de la WeMos y utilizamos el puerto 81, para la comunicación websocket.

```
12  IP = "10.3.10.154"
13  puerto = 81
```

3. Definimos ruta y cola_mensajes como globales (más adelante se utilizan hilos para tener múltiples bucles y poder compartir información entre funciones) también definimos una ruta para que funcione como una fila y una cola de mensajes para procesar cómodamente el envío de datos a la WeMos.

```
15  global ruta
16  ruta = deque()
17  global cola_mensajes
18  cola_mensajes = Queue()
```

4. Variables globales para designar el ArUco móvil.

```
21  # global aruco_id_fijo
22  # aruco_id_fijo = 1 # ID del aruco fijo
23  global aruco_id_movil
24  aruco_id_movil = 99 # ID del aruco móvil
```

5. Designamos una distancia tope para estimar la interacción entre un ArUco fijo y un ArUco móvil (en pixeles).

```
27  global umbral_distancia
28  umbral_distancia = 150 #su unidad es pixeles
```


6. Definimos una función para recibir los mensajes y procesar el formato para separarlo por instrucciones para luego guardarlo, pues el formato establecido es 1-2-3-4.

```
32 def mensajeEntrante(ws, message):
33     print("Recibido: " + message)
34     instrucciones = message.split("-")
35     for i in instrucciones:
36         print(i)
37         if i.isdigit():
38             ruta.append(i)
39     print(ruta)
```

7. Definimos una función para enviar mensajes a la WeMos utilizando una cola de mensajes para evitar enviar demasiados mensajes y sobrecargar la comunicación WebSocket, además de que si un mensaje de la cola es “salir”, terminamos la comunicación.

```
41 def enviarMensajes(ws):
42     while True:
43         # Esperar a que haya un mensaje en la cola
44         msg = cola_mensajes.get()
45         if msg == "salir":
46             finConexion(ws, 0, "Conexión cerrada de forma segura")
47             break
48         else:
49             ws.send(msg)
50             time.sleep(1)
```

8. Definimos funciones necesarias para procesar errores dentro de la conexión y para finalizar la conexión.

```
52 def error(ws, error):
53     print("Error: " + str(error))
54
55 def finConexion(ws, close_status_code=0, close_msg="Conexion cerrada"):
56     if close_status_code == 0:
57         print(f"[INFO] {close_msg}, code: {close_status_code}")
58     ws.close()
```

9. Se crea una función para iniciar la conexión del websocket con un mensaje y crear un hilo para mantener activo el bucle del envío de datos.

```
60 def inicioConexion(ws):
61     print("[INFO] Conexión abierta")
62     print("\n")
63     ws.send("Iniciando comunicación...")
64
65     # Crear e iniciar el hilo de envío de mensajes
66     t = threading.Thread(target=enviarMensajes, args=(ws,))
67     t.start()
```

10. Se crea una función para crear la comunicación websocket, asociarle todas sus funciones bases e iniciar un bucle indefinido de comunicación.

```
69 def comunicacionWebSocket():
70     ws = websocket.WebSocketApp(f"ws://{IP}:{puerto}/",
71                                on_message = mensajeEntrante,
72                                on_error = error,
73                                on_close = finConexion)
74     ws.on_open = inicioConexion
75     ws.run_forever()
```

11. Se crea una función para controlar la detección de ArUco y poder controlar el termino del programa al presionar la tecla "q".

```
79 def controlMedianteArUco():
80     time.sleep(1)
81     print("[INFO] Inicializando control", end="")
82     for i in range(3):
83         print(".", end="")
84         time.sleep(1)
85     print("\n")
86
87     while True:
88         if keyboard.is_pressed('q'):
89             exit()
90         while len(ruta) > 0:
91             detectarAruco(ruta)
```

12. Función de detección de ArUco e instrucciones de movimiento en base a una ruta recibida.

```
94 def detectarAruco(ruta):
```

13. Descomposición de ruta en instrucciones y transformación de estas a entero.

```
95     instruccion = ruta.popleft()
96     instruccion=int(instruccion)
97     print(f'Procesando: {instruccion}')
```

14. Definición del diccionario de ArUcos y creación de parámetros.

```
99     dictionary = aruco.Dictionary_get(aruco.DICT_4X4_100)
100    parameters = aruco.DetectorParameters_create()
```

15. Inicializar captura de video.

```
102    cap = cv2.VideoCapture(0) # Puedes ajustar el parámetro según tu
    configuración
```

16. Asignación de medidas en metros de ArUcos fijo y en movimiento, respectivamente.

```
104    markerLength_af = 0.04755
105    markerLength_am = 0.1 #medida del aruco movil
106    movil_en_movimiento = False
107    aruco_fijo_detectado = False
```

17. Asignar valores a la matriz y coeficientes de distorsión (se consigue haciendo el tutorial de calibración).

```
109    # Asume que tienes la matriz de la cámara y los coeficientes de distorsión
110    camera_matrix = np.array([[239.51027567, 0, 314.1138314 ],[ 0, 238.47458721,
284.9145289 ],[0, 0, 1]])
111    dist_coeffs = np.array([-0.10410461, 0.29094547, -0.00622743, -0.00109586, 0.
99861016])
116    instruccion_procesada = False
```

18. Bucle que termina cuando una instrucción fue completada.

```
117    while not instruccion_procesada:
118        ret, frame = cap.read()
119        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) # Convierte la imagen a
    escala de grises
```

19. Función para detectar los arucos en cámara

```
121    # Detecta los arucos en la imagen
122    corners, ids, rejectedImgPoints = aruco.detectMarkers(gray, dictionary,
    parameters=parameters)
```

20. Condicional para saber si el ArUco que se va a mover está detectado.

```
124         # Comprueba si el aruco móvil está en movimiento
125         if ids is not None and aruco_id_movil in ids:
126             movil_en_movimiento = True
127         else:
128             movil_en_movimiento = False
```

21. Condicional para saber si el ArUco solicitado como “instrucción” es detectado.

```
130         if ids is not None and instruccion in ids:
131             print(f"Buscando...{instruccion}")
132             aruco_fijo_detectado = True
```

22. Obtención del índice del ArUco fijo, para luego conseguir la esquina de este.

```
134         aruco_fijo_index = np.where(ids == instruccion)[0][0] # Obtén el
                        índice del aruco fijo en los IDs detectados
135         corners_fijo = corners[aruco_fijo_index] # Obtén las esquinas del
                        aruco fijo
```

23. Si el ArUco móvil es detectado, se obtendrá el índice y su esquina.

```
137         # Comprueba si el aruco móvil y el aruco fijo están suficientemente
                        cerca
138         if movil_en_movimiento:
139             aruco_movil_index = np.where(ids == aruco_id_movil)[0][0] #
                        Obtén el índice del aruco móvil en los IDs detectados
140             corners_movil = corners[aruco_movil_index] # Obtén las esquinas
                        del aruco móvil
```

24. Cálculo de distancia entre la esquina del ArUco móvil y el ArUco fijo “instrucción”.

```
142         # Calcula la distancia entre el aruco móvil y el aruco fijo
143         distancia = np.linalg.norm(corners_movil - corners_fijo)
144         print(distancia)
```

25. Estimación de la posición de los ArUcos (fijos y móviles) para luego calcular el ángulo de rotación con respecto al ArUco fijo.

```
146         # Estima la pose del aruco fijo
147         rvecs_fijo, tvecs_fijo, _ = aruco.estimatePoseSingleMarkers
                        (corners_fijo, markerLength_af, camera_matrix, dist_coeffs)
148
149         # Estima la pose del aruco móvil
150         rvecs_movil, tvecs_movil, _ = aruco.estimatePoseSingleMarkers
                        (corners_movil, markerLength_am, camera_matrix, dist_coeffs)
151
152         # Calcula el ángulo de rotación del aruco móvil con respecto al
                        aruco fijo
153         angle = np.arctan2(tvecs_movil[0][0][1] - tvecs_fijo[0][0][1],
                        tvecs_movil[0][0][0] - tvecs_fijo[0][0][0])
```

26. Pasar el ángulo de radianes a grados.

```
155     angle = np.degrees(angle) # Convierte el ángulo a grados
156     print("Ángulo de rotación: ", angle)
```

27. Condicional para saber si la distancia calculada es menor al umbral.

```
158     if distancia < umbral_distancia: # Ajusta el umbral de distancia
                                         según tu configuración
```

28. Si es menor, da instrucciones de movimiento mediante cola_mensajes y avisa que llegó al ArUco instruido, luego registra que la instrucción se procesó .

```
159         print("Aruco móvil llegó al aruco fijo")
160         cola_mensajes.put("avanzar 500")#determinar cuando avanzar
                                         una vez se acerca
161         time.sleep(1)
162         cola_mensajes.put("detener")#luego de avanzar detenerse
163         time.sleep(1)
164         instruccion_procesada = True
165     else:
```

29. Si es mayor o igual, aún no llega al ArUco entonces, en base al ángulo de rotación se decide que acción realizar y lo comunica con cola_mensajes hasta llegar al ArUco.

```
166         print("Aruco móvil en movimiento")
167         if angle < 25 and angle > -25:
168             cola_mensajes.put("avanzar 1000")
169             time.sleep(1)
170         elif angle >= 25:
171             cola_mensajes.put("izquierda 1000")
172             time.sleep(1)
173         elif angle <= -25:
174             cola_mensajes.put("derecha 1000")
175             time.sleep(1)
```

30. Si el ArUco fijo “instrucción” no fue detectado, avisa que no está y lo mantendrá en bucle.

```
176     else:
177         aruco_fijo_detectado = False
178         print("Aruco fijo no detectado")
179         cola_mensajes.put("detener")
180         time.sleep(1)
181         instruccion_procesada = True
```

31. Dibuja los ArUcos detectados por la cámara.

```
183     frame_markers = aruco.drawDetectedMarkers(frame, corners, ids)
```

32. Asignar nombre a la ventana que aparecerá.

```
185 cv2.imshow('Detección de arucos', frame_markers)
```

33. Condicional para que con la letra “q” se cierre la ventana y por tanto esa instrucción.

```
187 if cv2.waitKey(1) & 0xFF == ord('q'):
188     break
```

34. Una vez se llega se cerrarán las ventanas.

```
190 cap.release()
191 cv2.destroyAllWindows()
```

35. Creamos la función principal e iniciamos 2 hilos, uno para la función de comunicación y otra para el control de las operaciones mediante ArUcos (como dije anteriormente es necesario utilizar hilos para tener varios bucles indefinidos dentro de un mismo programa en Python.

```
194 if __name__ == '__main__':
195
196     # Crear los hilos
197     t1 = threading.Thread(target=comunicacionWebSocket)
198     t2 = threading.Thread(target=controlMedianteArUco)
```

36. Iniciamos los hilos y dejamos en un bucle la función principal esperando que se finalice el programa al presionar la tecla “q”.

```
200     # Iniciar los hilos
201     t1.start()
202     t2.start()
203
204     while True:
205         if keyboard.is_pressed('q'):
206             # Establecer los eventos para detener los hilos
207             print("[INFO] Deteniendo hilos...")
208             time.sleep(1)
209             cola_mensajes.put("salir")
210             break
211     # Esperar a que ambos hilos terminen
212     t1.join()
213     t2.join()
```

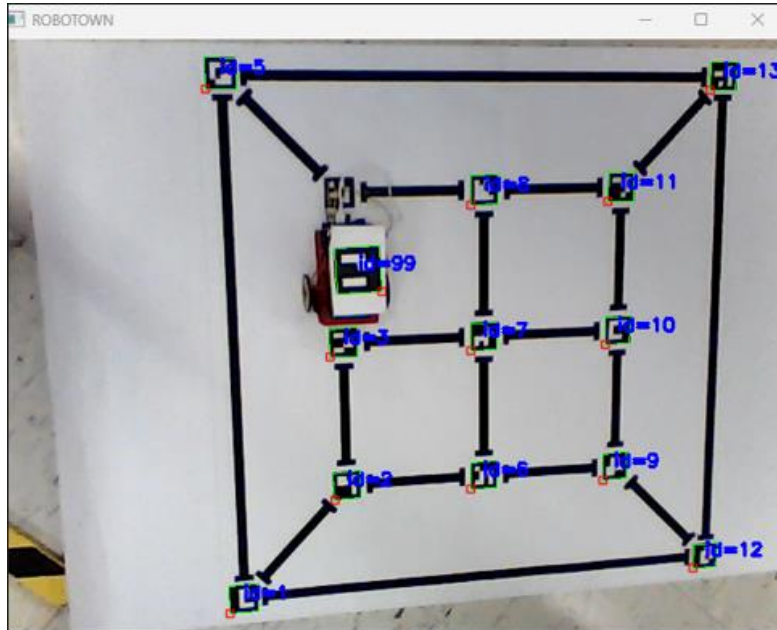


Figura 17: Captura de pantalla del control mediante el ArUco 99

7. Referencias

altronics. (s.f.). *Cámara OV7670 para Arduino*. Obtenido de <https://altronics.cl/camara-ov7670-arduino>

Altronics Chile SPA. (s.f.). *Servo MG 996R*. Obtenido de <https://altronics.cl/servo-mg996r-180>

Arduino. (s.f.). *Descarga Arduino IDE*. Obtenido de <https://www.arduino.cc/en/software>

Dev, M. N. (s.f.). *Esp32FS*. Obtenido de <https://github.com/me-no-dev/arduino-esp32fs-plugin/releases/>

espressif. (s.f.). *Github SPIFFS*. Obtenido de <https://github.com/espressif/arduino-esp32/tree/master/libraries/SPIFFS>

Kalachev, O. (s.f.). *Online ArUco markers generator*. Obtenido de <https://chev.me/arucogen/>

kangax. (s.f.). *HTML minifier*. Obtenido de <http://kangax.github.io/html-minifier/>

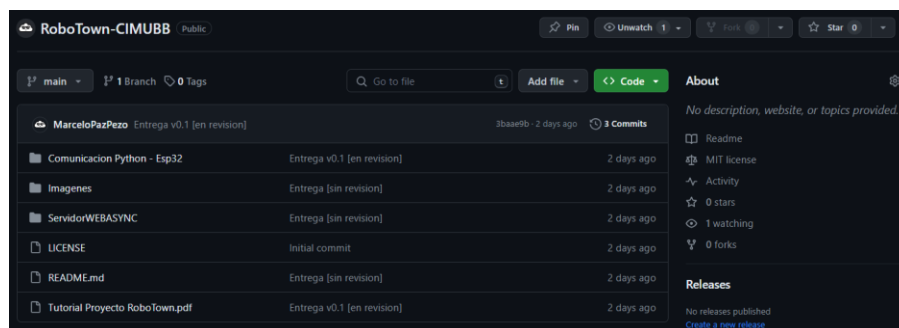
paginaswebschile.com. (s.f.). *Sensor de línea Analogo QTR8*. Obtenido de <https://ardumotica.cl/producto/sensor-de-linea-analogo-qtr-8/>

Random Nerd Tutorials. (s.f.). *Install ESP32 Filesystem Uploader in Arduino IDE*. Obtenido de <https://randomnerdtutorials.com/install-esp32-filesystem-uploader-arduino-ide/>

UCO. (s.f.). *ArUco: a minimal library for Augmented Reality applications based on OpenCV*. Obtenido de <https://www.uco.es/investiga/grupos/ava/portfolio/aruco/>

Villalpando, J. A. (s.f.). *Wemos D1 R32 ESP32*. Obtenido de http://kio4.com/arduino/100_Wemos_ESP32.htm

8. Repositorio de GitHub



[RoboTown-CIMUBB](#)