UNIVERSIDAD NACIONAL DE CUYO – FACULTAD DE INGENIERÍA

CONTROL DE AUTOMÓVIL MEDIANTE GESTOS

PROYECTO FINAL – REALIDAD VIRTUAL

21 DE MARZO DEL 2022

COSTARELLI AGUSTÍN
Legajo 10966
MANZANO Claribel Estefanía
Legajo 11362

1. RESUMEN

En el siguiente informe se desarrolla la presentación diseño y construcción de un automóvil impreso en 3D controlado mediante diferentes gestos realizados por el usuario con sus manos. Se describirán las herramientas de hardware y software utilizadas y se explicará su desarrollo completo para su posible recreación.

2. ÍNDICE

Tabla de contenido

3. INTRODUCCIÓN

El presente proyecto se centra en el desarrollo de un vehículo o automóvil impreso en 3D controlado por el movimiento de las manos del usuario o mediante consignas por reconocimiento de códigos ArUco.

El objetivo es lograr este control a distancia, de manera inalámbrica, sin necesidad de que el usuario se encuentre delante de una computadora y que sólo se realice mediante gestos determinados, creando una experiencia agradable de manejo.

El listado de objetivos entonces sería:

- Lograr la captura y procesamiento de imágenes de los gestos realizados por las manos del usuario, transformando estos en datos de velocidad;
- Añadir gestos para comenzar o pausar el proceso;
- Utilizar ArUco para realizar funciones determinadas;
- Construir un automóvil de dos ruedas motorizadas;
- Lograr la comunicación serial entre la PC y un Arduino Mega;
- Establecer la comunicación entre el Arduino Mega y el Arduino Uno ubicado en el automóvil mediante los módulos nRF24l01;
- Accionar los motores según las órdenes recibidas.

En las siguientes secciones del informe se describirán distintos aspectos del desarrollo del entorno. Paso a paso se detallará cómo reproducirlo de forma completa. Se empezará con una breve descripción técnica del proyecto y de las herramientas utilizadas para su fabricación. Luego se mencionará como se realiza el hardware y el software necesario para su funcionamiento. Finalmente se mostrarán los resultados del proyecto haciendo uso completo de todo lo explicado.

4. PRESENTACIÓN FORMAL Y TÉCNICA

4.1. Descripción detallada del proyecto

Como se mencionó anteriormente, uno de los objetivos principales del proyecto es lograr el manejo del automóvil de manera inalámbrica. Esto se logra mediante la comunicación entre varios dispositivos de hardware utilizados.

Primeramente, se captan las manos del usuario con una cámara, ya sea la webcam de una notebook o la de un teléfono celular. Para lograr utilizar esta última, se hace uso de una aplicación llamada DroidCam, instalada tanto en el teléfono como en la computadora. Ambos dispositivos deben estar conectados a la misma red de wifi para que la comunicación se logre.

Estas imágenes serán tomadas por el código que fue desarrollado en Python, utilizando distintas librerías, especialmente OpenCV, la cual facilita el procesamiento de imágenes captadas por la cámara. Gracias a esto se realiza en tiempo real el reconocimiento de cada mano, distinguiendo si se trata de la derecha o de la izquierda, ya que cada una de ellas modificará variables diferentes. Para que estos comandos funcionen, ambas manos deben estar presentes dentro de la imagen captada y una vez detectada su presencia, el algoritmo podrá funcionar. También en lugar de las manos se puede presentar frente a la cámara marcadores ArUco los cuales mostrarán por pantalla una imagen en realidad aumentada y la misma tendrá asociada un comando predefinido que será enviado al vehículo.

Con respecto a los gestos, de la mano izquierda se tomará su posición angular desde una línea horizontal que pasa por el medio de la pantalla, si este ángulo es positivo, la velocidad del vehículo aumentará de manera proporcional, en el ángulo cero el vehículo estará detenido y con uno negativo, retrocederá.

Concerniente a la mano derecha se hará algo similar. Se toma su posición angular con respecto a una línea vertical en la mitad de la parte derecha de la pantalla. Si el ángulo es positivo, el vehículo aumentará la velocidad de la rueda izquierda y comenzará a girar a la derecha, mientras que cuando el ángulo sea negativo, hará lo contrario aumentando la velocidad de la rueda derecha y girando a la izquierda. Finalmente, al posicionar la mano cerca del ángulo cero, el vehículo no tendrá ningún ángulo de giro.

Estos movimientos son proporcionan una experiencia agradable de manejo ya que son bastante instintivos de realizar para el usuario, lo cual era otro de los objetivos de este proyecto.

Según la posición de las manos entonces, se definirán las velocidades de las ruedas del vehículo, siendo un 60% aportada por la mano izquierda y un 40% por el posible giro indicado por la mano derecha. Ambas velocidades se enviarán como dato desde el algoritmo realizado en Python hacia un Arduino Mega o Atmega2560 conectado al ordenador por puerto serie, mediante comunicación UART. Los datos son recibidos por el Arduino y se procesan para luego ser enviados automáticamente hacia el Arduino UNO que se encuentra en el automóvil, mediante un transmisor de radiofrecuencia llamado NRF24L01. Una vez recibidos los datos en el Arduino UNO, son nuevamente procesados y finalmente dan la orden recibida a los motores, para variar su velocidad.

Es así que el usuario podrá lograr el control del vehículo con sus propias manos, de manera inalámbrica y con los datos de las velocidades impresos en la pantalla para que pueda ser consciente de los movimientos que está realizando.

4.2. Descripción técnica de las herramientas utilizadas

En este apartado se describen las herramientas utilizadas para el desarrollo del proyecto. Se especifica el software y el hardware empleado en su construcción. También se muestra una pequeña introducción para su uso y se definen los conceptos básicos necesarios.

4.2.1.Descripción del Software

- **OpenCV en Python**

Para el reconocimiento de manos se utiliza OpenCV (Open Source Computer Vision Library) es una librería de software de Machine Learning y Computer Vision de código abierto y multiplataforma. OpenCV se creó para proporcionar una infraestructura común para aplicaciones de visión por computadora y para acelerar el uso de la percepción de la máquina en los productos comerciales [1]. OpenCV está totalmente desarrollado en C++, orientado a objetos y con alta eficiencia computacional. Su API es C++ pero incluye conectores para otros lenguajes: como Python, Java, Matlab, entre otros. En este proyecto se implementará en Python.

Figura 1 – OpenCV como librería de Python

De esta librería el módulo más importante para nuestro proyecto es el de seguimiento de manos “HandTrackingModule”. Una vez recibida la imagen por cámara este módulo puede rápidamente realizar el reconocimiento y seguimiento de manos para lo cual hace uso de 21 marcas que aparecen al ser reconocidas, los mismos están ubicados en puntos específicos como se muestran en la Figura 2.

Gracias a esta marcación sobre las manos es posible diferenciar entre mano izquierda y derecha, que cantidad de dedos se encuentran levantados y así como un enmarcado de la región de las manos como se muestra en la Figura 3, esto permite calcular el centro de las manos.

Tomando como referencia el centro de la mano izquierda, se trazó hacia el centro vertical izquierdo de la pantalla una recta, midiendo el ángulo de la misma respecto de la horizontal y se utilizó como consigna de velocidad. A su vez desde el centro de la mano derecha se trazó a la mitad inferior derecha de la pantalla una recta a la cual se le mide el ángulo respecto la vertical para así obtener una medida de consigas de giro izquierda o derecha, esto se puede apreciar en la Figura 4.

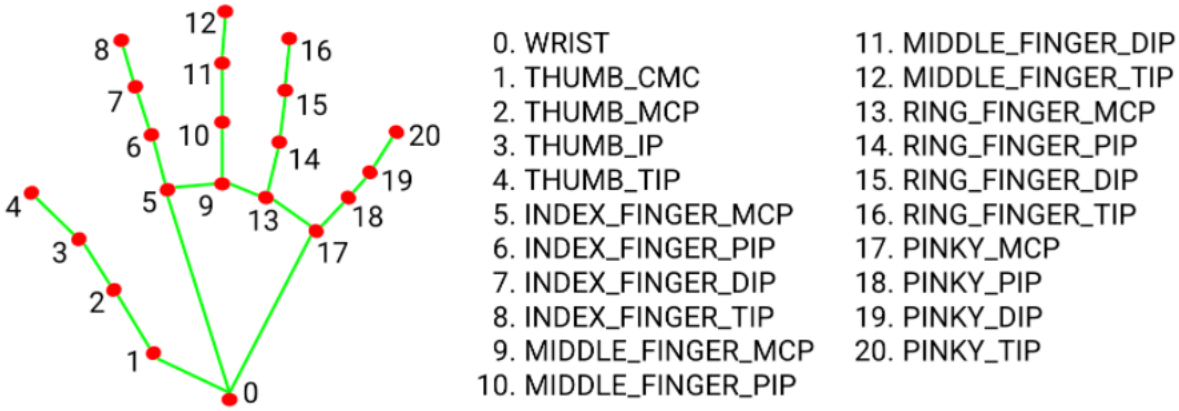

Figura 2 – Marks listadas en "lmList"

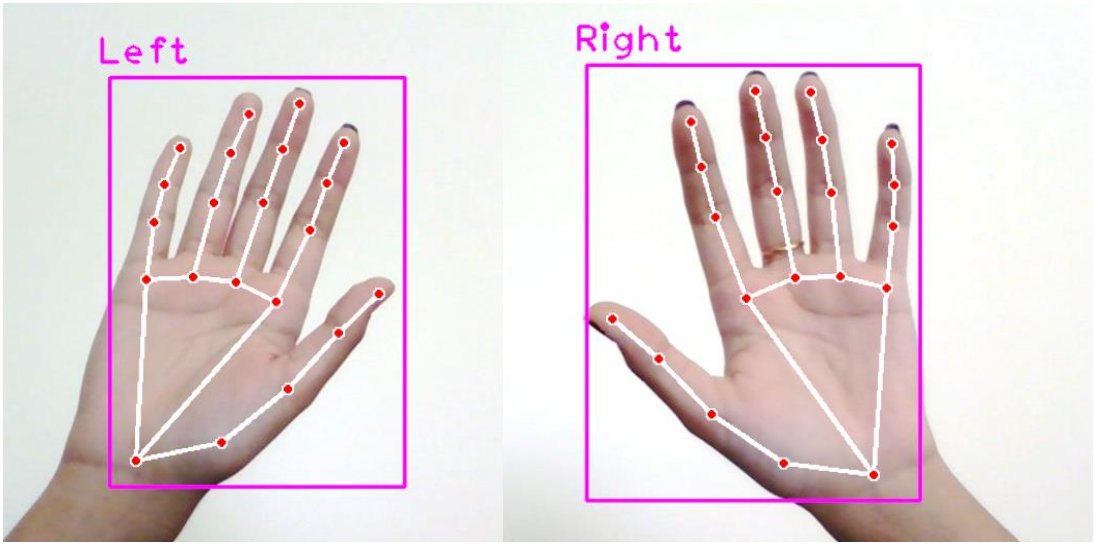

Figura 3 – Reconocimiento de mano derecha e izquierda

Gracias al uso de todas estas herramientas se desarrolló el método de medición de la posición angular de las manos anteriormente descripto. Se optó por este procedimiento ya que, si bien la librería posee muchas ventajas con respecto al seguimiento de manos, también posee algunas desventajas que hay que saber sobrellevar.

Al principio, se buscó realizar la aceleración del automóvil mediante el acercamiento de la mano izquierda a la pantalla, o si no, al abrir o cerrar la mano, pero se descubrió que el rectángulo o bounding box cambia de tamaño tanto al cerrar o abrir la mano, como al realizar un acercamiento o alejamiento de la mano a la pantalla, lo cual nos hizo renunciar a estas ideas debido a que al elegir una, se vería influenciada por la otra. Es por eso que se decidió implementar la posición angular, que no se ve influenciada de ninguna manera por el tamaño de la bounding box.

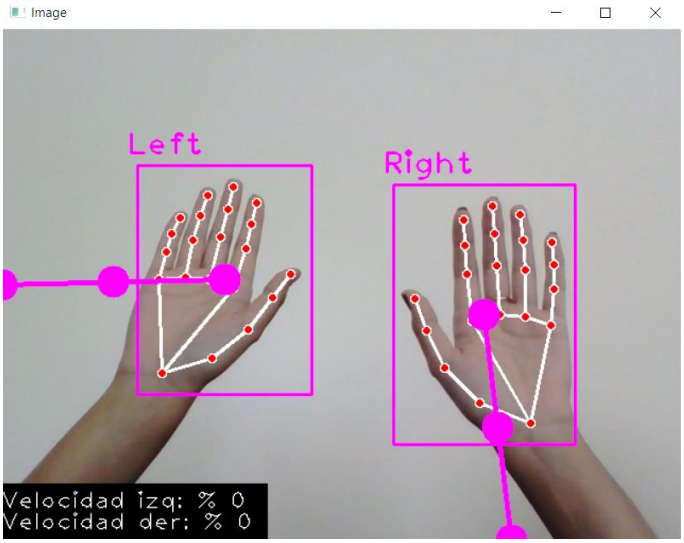

Figura 4 – Posiciones angulares en valor cero

Como segunda opción para la experiencia se implementaron los marcadores ArUco los cuales también pueden ser procesados por la librería de OpenCV y a los mismos se les asignó una consigna específica.

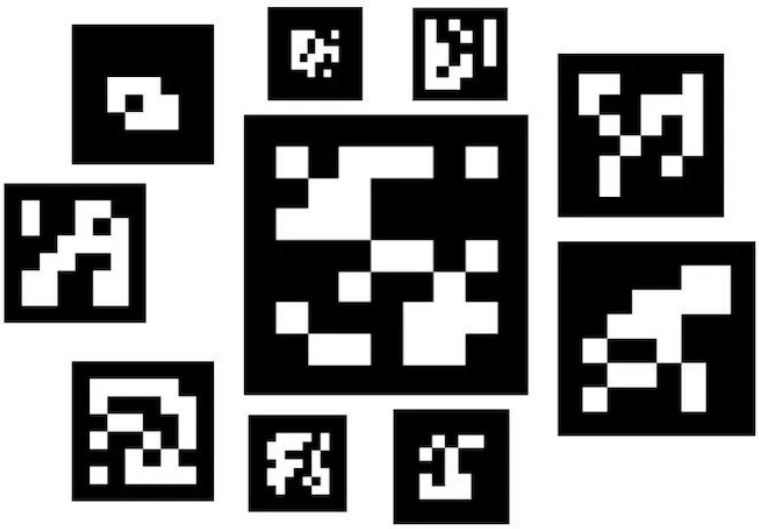

Figura 5 – Marcadores ArUco

Las consignas asignadas a los marcados fueron tres, en primer lugar, la de parada del sistema que entrega velocidades nulas a los motores, en segundo lugar, la consigna de girar en el lugar otorgando a los motores misma potencia, pero con giro en direcciones contrarias y, por último, la consigna de avanzar con ambos motores hacia adelante.

Estos marcadores son utilizados para la incorporación de realidad aumentada, cuando un marcador es tomado por la cámara no sólo entrega una consigna, sino que superpone sobre el marcador una imagen definida como se puede ver en la Figura 6, se trabajó para que no importe el ángulo de la cámara, mientras todo el marcador sea visible la lectura será efectuada.

Se utilizaron ArUco de 6x6, IDE 1, 2 y 3 correspondiente a cada imagen de la figura 7.

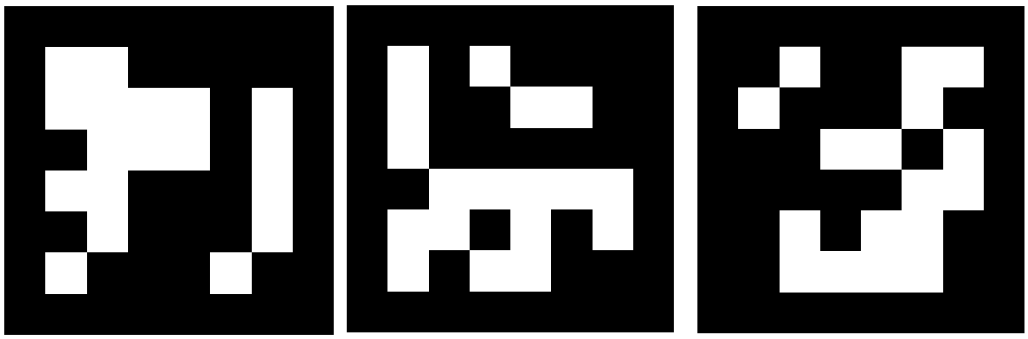

Figura 6 – ArUco 6x6 de IDE 1, 2 y 3

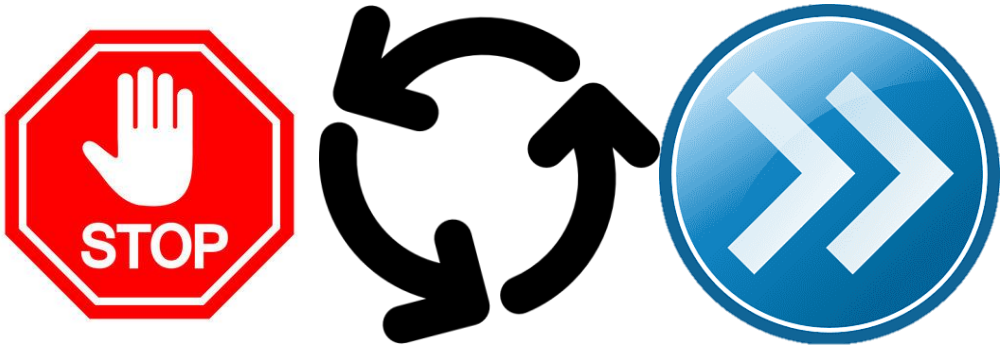

Figura 7 – Imágenes correspondientes a ArUco de IDE 1, 2 y 3

- **Arduino Software**

Figura 8 – IDE Arduino

El entorno de desarrollo integrado (IDE) de Arduino es una aplicación de código abierto y multiplataforma (para Windows, macOS, Linux) que está escrita en el lenguaje de programación Java. Se utiliza para escribir y cargar programas en placas compatibles con Arduino, pero también, con la ayuda de núcleos de terceros, se puede usar con placas de desarrollo de otros proveedores. El IDE de Arduino admite los lenguajes C y C++. [2]

4.2.2. Descripción del Hardware

- **Arduino Mega (Atmega2560)**

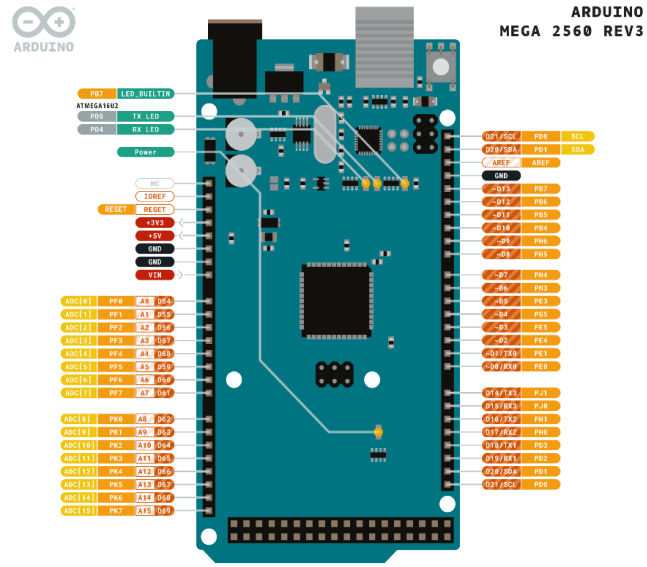

Figura 9 – Esquema Arduino Mega 2560

El Arduino Mega 2560 es una placa de microcontrolador basada en el ATmega2560. Tiene 54 pines de entrada/salida digital (de los cuales 15 se pueden usar como salidas PWM), 16 entradas analógicas, 4 UART (puertos seriales de hardware), un oscilador de cristal de 16 MHz, una conexión USB, un conector de alimentación, un cabezal ICSP, y un botón de reinicio. [3]

Este Arduino es utilizado para recibir los datos directamente desde Python mediante una comunicación serie. Estos datos son las velocidades derecha e izquierda de las ruedas y son recibidas con un formato de seis dígitos. Estos luego son transformados en dos números que se encuentran en el intervalo [-100, 100] para ser enviados nuevamente hacia el Arduino UNO ubicado en el automóvil. Esta comunicación se realiza mediante dos transceptores de radiofrecuencia llamados nRF24l01.

- **Arduino UNO**

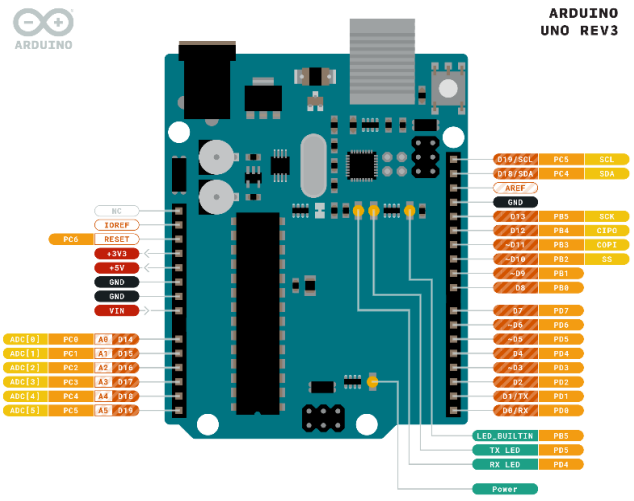

Figura 10 – Esquema Arduino UNO

Arduino UNO es una placa de microcontrolador basada en el ATmega328P. Tiene 14 pines de entrada/salida digital (de los cuales 6 pueden usarse como salidas PWM), 6 entradas analógicas, un resonador de cerámica de 16 MHz, una conexión USB, un conector de alimentación, un cabezal ICSP y un botón de reinicio. [4]

Este Arduino está integrado en el automóvil, recibe los datos desde el Arduino Mega mediante el transceptor de radiofrecuencia nRF24l01, estos dos números recibidos reciben un último procesamiento para luego ser enviados directamente hacia los motores.

Considerando que se reciben dos números iguales, es decir que el vehículo no girará, podemos decir que, en el rango de los números positivos, los motores harán que el automóvil se desplace hacia adelante; si son negativos, irá hacia atrás y en el cero se detendrá. Ahora considerando velocidades diferentes, si la velocidad de la rueda derecha es mayor que la de la izquierda (v_der>v_izq), entonces se realizará un giro a la izquierda y a la inversa (v_izq>v_der), se girará a la derecha.

- **Módulo NRF24L01**

Figura 11 – Esquema del módulo nRF24l01

Este módulo puede funcionar como transmisor o como receptor de radio-frecuencia. Utiliza la banda de 2,4 GHz y puede operar con velocidades de transmisión de 250 kbps hasta 2 Mbps. El módulo puede usar 125 canales diferentes (desde 2,4 GHz a 2,525 GHz, separados cada 1 MHz), lo que da la posibilidad de tener una red de 125 módulos que funcionen con independencia uno de otro en un solo lugar, sin producir interferencia entre sí. Cada canal puede tener hasta 6 direcciones, es decir, cada unidad puede comunicarse con hasta otras 6 unidades al mismo tiempo.

El módulo debe ser alimentado con una tensión entre 1,9V y 3,6V, por lo que fue conectado a la salida de 3,3V del Arduino.

Tres de sus pines son para la comunicación SPI: MOSI, MISO y SCK, que deben conectarse a los pines de la interfaz SPI del Arduino. Los pines CSN y CE se pueden conectar a cualquier pin digital de la placa Arduino, y su función es configurar el módulo en modo de espera o activo, así como para alternar entre modo de transmisión o de comando. El último pin, IRQ, es un pin de interrupción que no es necesario utilizar. [5]

- **Puente H**

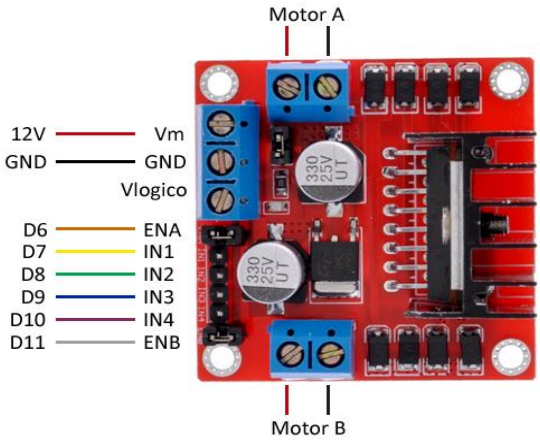

Figura 12 – Esquema de Puente H

El puente H es un circuito electrónico que permite a un motor eléctrico DC girar en ambos sentidos, avanzar y retroceder.

Los puentes H ya vienen hechos en circuitos integrados, pero también se pueden construir a partir de componentes electrónicos. Un puente H se construye con cuatro interruptores (mecánicos o mediante transistores). Cuando los interruptores INT1 e INT4 están cerrados (INT2 e INT3 abiertos) se aplica una tensión haciendo girar el motor en un sentido. Abriendo los interruptores INT1 e INT4 (cerrando INT2 e INT3), el voltaje se invierte, permitiendo el giro en sentido inverso del motor.

El circuito Puente H no solo permite un funcionamiento on/off del motor, a plena potencia en un sentido o en el otro, sino que también ofrece un modo de controlar la velocidad al quitar los jumpers de ENA y ENB y al entregarles voltaje por esos pines. [6]

El puente H es alimentado con una batería de 7,4V y 1500mAh. Cabe destacar que cuenta con un regulador de tensión que da una salida de 5V, la cual fue destinada a alimentar el Arduino UNO.

- **Motores DC**

Figura 13 – Conjunto Motor DC y rueda

Lo motores utilizados son motores de corriente continua o DC, cuentan con engranajes con doble eje y una reducción de 1:48. El eje es plano y tiene un orificio interior de 1.9mm. Trabajan con un voltaje de 3V a 6V y su velocidad nominal es de 100rpm. [7]

5. EXPLICACIÓN PASO A PASO

En esta sección se detallará el proceso de desarrollo del proyecto, tanto su construcción como el desarrollo de los códigos. El proyecto abarca archivos desarrollados en los lenguajes: Arduino (C++) y Python con la librería OpenCV.

5.1. Construcción del automóvil

Para llevar a cabo el proyecto se construyó un automóvil desde cero, empezando con su impresión 3D, luego con su ensamblaje y conexión y finalmente con su programación.

5.1.1. Impresión en 3D

El diseño del automóvil escogido fue originalmente descargado de la web, específicamente de la página de Thingiverse, en la bibliografía se puede encontrar el link de descarga. [8]

Luego, se procedió a modificar el diseño para adaptarlo a nuestras necesidades, es decir, un automóvil de dos ruedas motorizadas y una rueda loca al frente.

En total se imprimieron cuatro piezas de material PLA: la base; el chasis, el techo y el paragolpes, que en total tuvieron un tiempo de impresión de 25 horas.

A continuación, se muestran algunas fotos del proceso de impresión.

Figura 14 – Impresión 3D del chasis y del piso

5.1.2. Ensamblaje y conexión

El proceso de ensamblaje fue bastante sencillo, por un lado, se unió el soporte interior con la base del automóvil y por otro, las ruedas se ensamblaron a los motores. Ambos conjuntos fueron atornillados entre sí y también la rueda loca, dando el resultado mostrado en la siguiente figura.

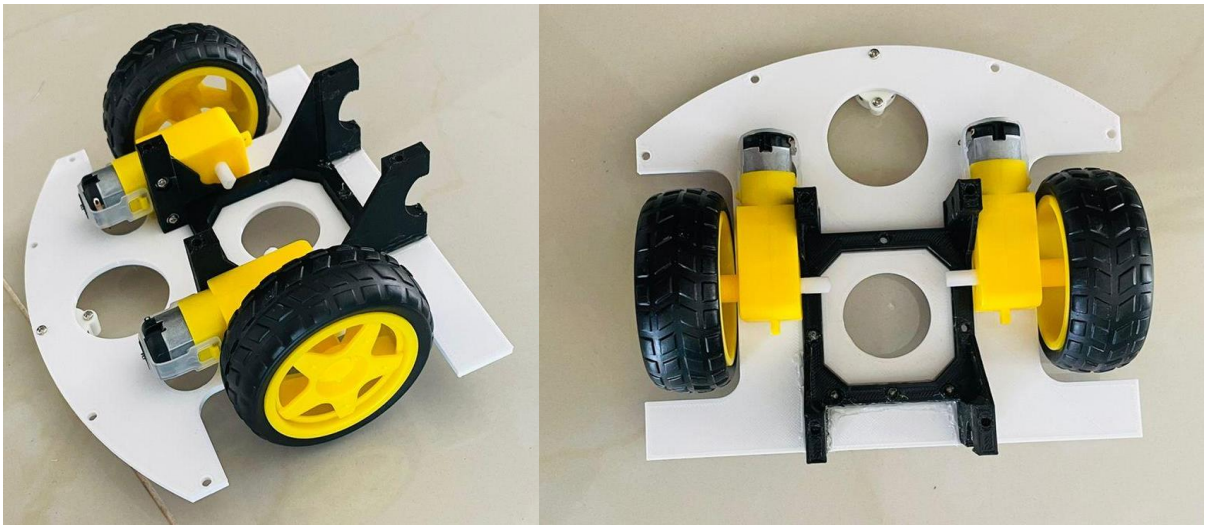

Figura 15 – Ensamblaje inicial

Para continuar, se agregó el puente H en el soporte interior y se conectó a los motores y a la batería siguiendo el esquema que se mostrará en la Figura 18.

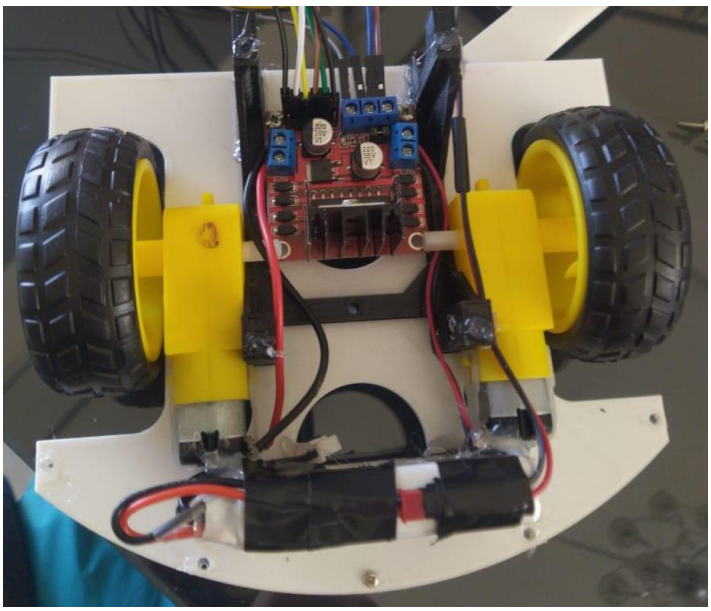

Figura 16 – Agregado del puente H y la batería

Luego se unió el paragolpes y el techo, el cual posee una ranura diseñada para atornillar el Arduino UNO.

Finalmente se realizaron todas las conexiones y se ubicó el módulo nRF24l01 en la parte posterior del automóvil, obteniendo como resultado final el mostrado en la siguiente imagen.

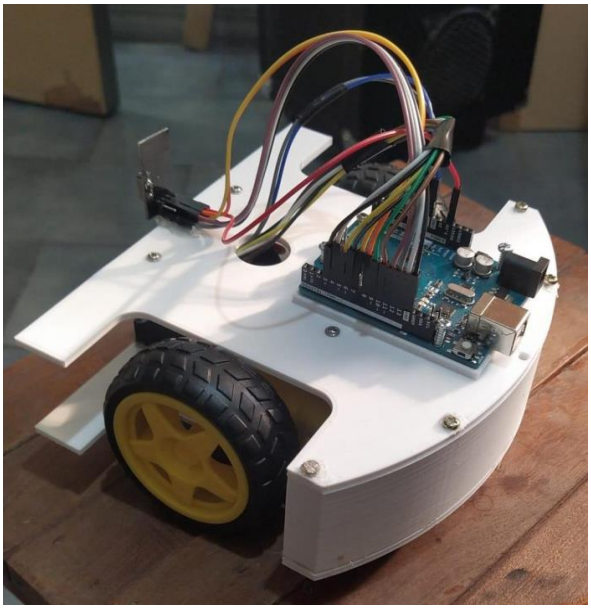

Figura 17 – Resultado final

A continuación, se muestran los esquemas de conexión utilizados, tanto en el automóvil con el Arduino UNO, como el del Arduino Mega.

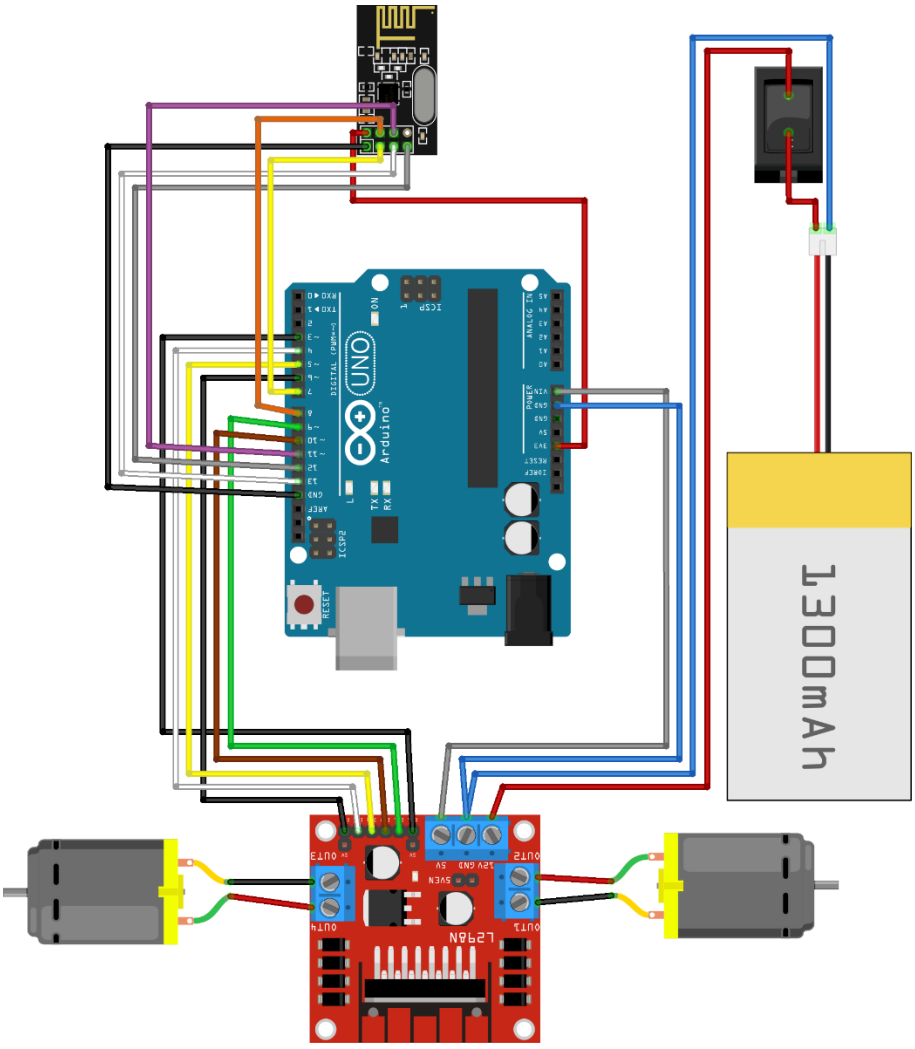

Figura 18 – Esquema de conexión del automóvil

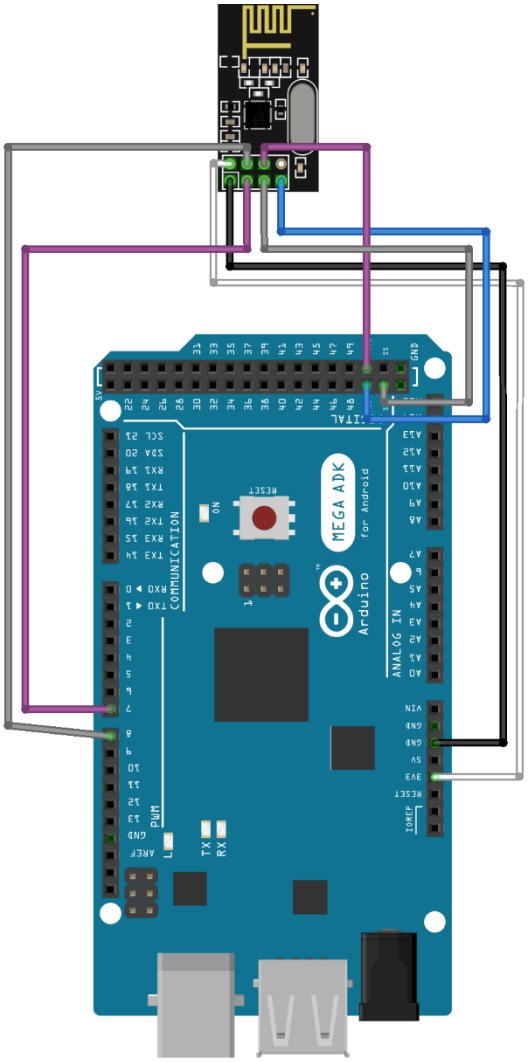

Figura 19 – Esquema de conexión del Arduino Mega

5.2. Desarrollo código Python

Se desarrolló un código en Python llamado "RV_final.py", que será el encargado de tomar las imágenes y realizar el procesamiento de las mismas. En este apartado se irá desglosando el código realizado a modo de guía para que pueda ser interpretado por el lector.

- **Librerías de importancia**

- **cv2**: Librería de OpenCV para Python.

- **cvzone**: Este es un paquete de visión computacional que facilita el procesamiento de imágenes y las funciones de IA. En el núcleo, utiliza las bibliotecas OpenCV y Mediapipe. Posee diferentes módulos y entre ellos están: 60 FPS Face Detection, Hand Tracking, Pose Estimation, Face Mesh Detection, entre otras.

- **serial**: Proporciona el acceso al puerto serial. Proporciona backends para Python que se ejecutan en Windows, Linux, OSX, entre otras.

- **multithreading**: Librería que permite el uso de hilos, haciendo que un programa ejecute múltiples operaciones simultáneamente en el mismo espacio de proceso.

● Desarrollo

Para lograr la comunicación serial con el Arduino Mega, se utilizó la librería "serial", definiendo el objeto "serialArduino" cuyo puerto de comunicación seleccionado es el "COM5" a una baudrate de 115200.

```
10    serialArduino = serial.Serial("COM5", 115200, timeout=1)
```

Figura 20 – Establecimiento de la comunicación serial

Se inicializa la cámara y se definen las dimensiones de la resolución, a su vez en parámetros se inicializa un detector de códigos ArUco y en diccionario se especifica las dimensiones del mismo

```
11    cap = cv2.VideoCapture(0)
12    cap.set(3, 640)
13    cap.set(4, 480)
14    parametros = cv2.aruco.DetectorParameters_create()
15    diccionario = cv2.aruco.Dictionary_get(cv2.aruco.DICT_6X6_100)
```

Figura 21 – Establecimiento de cámara y códigos ArUco

La variable "cad" serán los datos a enviar y se definen en la siguiente línea. Dado que la lógica en Python es obtener valores de velocidad entre -100 y 100 esto representa un problema para la comunicación al enviar un signo negativo. Se decide sumar 100 a las velocidades de esta manera el rango a enviar de consignas será entre 0 y 200. Este valor cuando sea recibido por el controlador de los motores restara 100 para obtener el valor original y proceder a su procesamiento.

```
27    cad = str(w_der + 100) + "," + str(w_izq + 100)
```

Figura 22 – Variable "cad" a enviar

Se prosiguió definiendo la función "func()" que escribe en el puerto especificado cada cierto tiempo. La función va a ejecutarse en un hilo paralelo llamado "t1". Esto debió hacerse ya que se presentaban diferentes problemas de comunicación y aunque esto añadió un mayor tiempo de latencia, fue la mejor solución que se pudo realizar. Es importante que cuando se inicia un hilo generar un modo de finalizarlo en este caso el hilo al ser ajustado en "daemon" finalizará cuando el proceso principal termine.

```
31  def func():
32      while True:
33          time.sleep(1)
34          serialArduino.write(cad.encode('ascii'))
35          print(cad)
36
37
38  t1 = threading.Thread(target=func)
39  t1.daemon = True
```

Figura 23 – Definición de la función func() y uso de threadings

Se inicializa un objeto de detección de manos el cual reconoce máximo dos manos por cámara, también se asigna un valor relacionado a la certeza de lo que se está detectando es realmente una mano.

```
29  detector = HandDetector(detectionCon=0.8, maxHands=2)
```

Figura 24 – Definimos el objeto "detector"

En un bucle while procedemos a captar las imágenes y a procesarlas. Primero tomamos la entrada de la cámara con "cap.read()", la invertimos con "cv2.flip()" y detectamos las manos con la función "detector.findHands()".

```
43  while True:
44      success, img2 = cap.read()
45      img = cv2.flip(img2, 1)
46      hands, img = detector.findHands(img, flipType=False)
```

Figura 25 – Bucle while

Para la detección de los marcadores ArUco es necesario primero convertir la imagen tomada por la cámara a escala de grises para después poder procesarla con el detector de ArUco y obtener información de la ubicación y los IDS asociados a dichos marcadores.

```
47      gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
48      esquinas, ids, bads = cv2.aruco.detectMarkers(img, diccionario, parameters=parametros)
```

Figura 26 – Tratamiento de la imagen y obtención de datos de ArUco

En esta parte aparecen dos condicionales if, cuyas condiciones son la presencia de dos manos y si esto no sucede, las velocidades de ambas ruedas se anularán. A continuación, se toman datos de las manos, como su centro, la bounding box, si es la mano izquierda o la derecha y cuantos dedos levantados posee. Aquí se muestra sólo para una.

```
50      if hands:
51          if len(hands) == 2:
52              hand1 = hands[0]  # Mano 1
53              lmList1 = hand1['lmList']
54              centerPoint1 = hand1["center"]
55              bbox1 = hand1['bbox']
56              handType1 = hand1["type"]
57              fingers1 = detector.fingersUp(hand1)
```

Figura 27 – Toma de datos de las manos

Las siguientes 25 líneas se encargarán de definir dos variables, “ang_vel” que será el ángulo que establezca la mano izquierda con respecto a una línea que pasa por la mitad de la pantalla y “ang_giro”, que indica el ángulo que la mano derecha tiene con una línea que pasa por la mitad de la parte derecha de la pantalla.

Luego procesamos estas variables, comenzamos con “ang_vel” que será igual a cero mientras se mantenga en el rango de [-10; 10] y su valor máximo será de 40. (Se muestra sólo avance).

```
101             if ang_vel < -10:  # avanzar
102                 if ang_vel < -40:
103                     ang_vel = -40
104                 w_der_A = (ang_vel * (-1) - 10) * 2
105                 w_izq_A = (ang_vel * (-1) - 10) * 2
```

Figura 28 – Procesamiento de “ang_vel”

Se realiza un procedimiento similar para la variable “ang_giro”, con el mismo rango para el valor cero y con un máximo de 30. Cabe destacar que la que será afectada, sólo será la velocidad de una de las ruedas dependiendo el sentido de giro. (Se muestra sólo giro a la derecha).

```
111             if ang_giro > 10:  # giro derecha
112                 if ang_giro > 30:
113                     ang_giro = 30
114                 w_der_B = (ang_giro - 10) * 2
115                 w_izq_B = 0
```

Figura 29 – Procesamiento de “ang_giro”

La suma de las velocidades de cada rueda obtenidas A con “ang_vel” y B con “ang_giro” se suman para llegar a las velocidades finales de cada rueda dadas por “w_der” y “w_izq”.

```
125             w_izq = int(w_izq_A + w_izq_B)
126             w_der = int(w_der_A + w_der_B)
```

Figura 30 – Obtención de las velocidades de cada rueda

Como segundo método de asignación de consignas siendo el de detección de marcadores ArUco primero se comprueba si en la imagen ingresada por cámara se encuentra alguno, esto se realiza revisando si en la variable de “ids” se encuentra alguno.

De ser así se dibuja el contorno del marcador sobre la imagen procesada, además se toma la posición de donde se encuentran las 4 esquinas del marcador.

```
136     if np.all(ids != None):
137         aruco = cv2.aruco.drawDetectedMarkers(img, esquinas)
138 
139         c1 = (esquinas[0][0][0][0], esquinas[0][0][0][1])
140         c2 = (esquinas[0][0][1][0], esquinas[0][0][1][1])
141         c3 = (esquinas[0][0][2][0], esquinas[0][0][2][1])
142         c4 = (esquinas[0][0][3][0], esquinas[0][0][3][1])
```

Figura 31 – Detección de ArUco

En función del marcador detectado es la consigna enviada como se describió anteriormente. En este paso se asigna en función del número de “ids” una imagen .jpg que se encuentra almacenada en la misma carpeta que el programa.

```
144         copy = img
145         if 1 in ids:
146             imagen = cv2.imread("1.jpg")
147             start = False
148         if 2 in ids:
149             imagen = cv2.imread("2.jpg")
150             start = True
151             w_izq = -30
152             w_der = 30
153         if 3 in ids:
154             imagen = cv2.imread("3.jpg")
155             start = True
156             w_izq = 30
157             w_der = 30
```

Figura 32 – Lectura de imágenes

Se realiza un arreglo con los puntos de las esquinas del marcador y se realiza otro arreglo que contiene el tamaño de la imagen cargada para el respectivo marcador.

```
161         puntos_aruco = np.array([c1, c2, c3, c4])
162
163         puntos_imagen = np.array([
164             [0, 0],
165             [tamaño[1] - 1, 0],
166             [tamaño[1] - 1, tamaño[0] - 1],
167             [0, tamaño[0] - 1]
168         ], dtype=float)
```

Figura 33 – Arreglos de ArUco e Imágenes

Se hacen coincidir las esquinas de la imagen con las del marcador para lograr el efecto de realidad aumentada. Mediante perspectiva se aplica una transformación a la imagen esto se hace con el fin de que cuando la cámara no esté enfocando totalmente perpendicular al marcador la imagen se deforme en función de cómo es captado el marcador por la cámara.

```
170         h, estado = cv2.findHomography(puntos_imagen, puntos_aruco)
171
172         perspectiva = cv2.warpPerspective(imagen, h, (copy.shape[1], copy.shape[0]))
173         cv2.fillConvexPoly(copy, puntos_aruco.astype(int), 0, 16)
174         img = copy + perspectiva
```

Figura 34 – Posición de la imagen sobre el ArUco

En caso de que no se encontraran marcadores en la imagen por cámara se asume que el proceso está en pausa

```
176     elif np.all(ids == None) and (len(hands) != 2):
177         start = False
```

Figura 35 – Ausencia de ArUco

Cuando el proceso encuentra que “start” es “False”, se imprime en pantalla la palabra PAUSADO.

```
192     if start == False:
193         cv2.putText(img, "PAUSADO", (220, 240), cv2.FONT_HERSHEY_PLAIN, 3, (255, 0, 255), 3)
```

Figura 36 – Impresión de PAUSADO

Por último, se agrega un rectángulo de fondo negro en la parte inferior izquierda de la pantalla para que las indicaciones de la consigna de velocidad puedan ser mejor observadas y se indica la programa mostrar por pantalla la imagen final después de haber sido procesada.

```
195     cv2.rectangle(img, (0, 480), (250, 427), (0, 0, 0), -1)
196     cv2.putText(img, text_1 + str(w_izq), (0, 450), cv2.FONT_HERSHEY_PLAIN, 1.5, (255, 255, 255), 1)
197     cv2.putText(img, text_2 + str(w_der), (0, 470), cv2.FONT_HERSHEY_PLAIN, 1.5, (255, 255, 255), 1)
198     cv2.imshow("Image", img)
199     cv2.waitKey(1)
```

Figura 37 – Muestra de velocidades en la pantalla

Los resultados obtenidos podrán observarse más adelante en el informe.

5.3. Desarrollo código Arduino

5.3.1. Arduino Mega

- **Librerías**

- **SPI.h**: Librería utilizada para realizar la comunicación SPI o Serial Peripheral Interface que es un protocolo de datos en serie síncrono que utilizan los microcontroladores para comunicarse rápidamente con uno o más dispositivos periféricos en distancias cortas. También se puede utilizar para la comunicación entre dos microcontroladores.[9]

- **RF24.h**: Es el driver de radio, biblioteca de capa 2 OSI para la comunicación de los módulos NRF24L01.

- **nRF24L01.h**: es una parte de la librería RF24 que incluye un grupo de macros constantes.

- **Desarrollo**

  Al comienzo se incluyen las librerías nombradas.

```
#include <SPI.h>
#include <RF24.h>
#include <nRF24L01.h>
```

Figura 38 – Librerías de Arduino

A continuación, se definen los pines CE y CSN, utilizados para la comunicación por radiofrecuencia. Se debe crear un array de bytes que representará la dirección, o el llamado pipe a través del cual se comunicarán los dos módulos nRF24l01, definido en la variable “direccion”. Finalmente se crea el objeto radio, indicando los pines a utilizar.

También las variables “pos”, “cad”, “cad1” y “cad2” destinadas a la recepción de datos que llegarán por el puerto serie.

```
#define CE_PIN 7
#define CSN_PIN 8
String cad,cad1,cad2;
int pos;
byte direccion[5] ={'c','a','n','a','l'};
RF24 radio(CE_PIN, CSN_PIN); // Create a Radio
int velocidad[3];
```

Figura 39 – Definición de variables

Se habilitan las transmisiones y se define el baudrate de la comunicación serial y también el de la comunicación por radiofrecuencia.

```
void setup() {

  Serial.begin(115200);
  radio.begin();
  radio.setDataRate(RF24_250KBPS);
  //radio.setRetries(3,5); // delay, count
  radio.openWritingPipe(direccion);
}
```

Figura 40 – Setup() Arduino

Para la recepción de datos desde Python se revisa primero si hay algo disponible por puerto serial, de ser así se recibe como String. El dato enviado siempre será la velocidad izquierda separada por una coma de la velocidad derecha. Se busca la posición de la coma en la cadena, se separan para luego ser guardados como valores enteros y ser enviados por el módulo Nrf24l01.

```
void loop() {

  if (Serial.available() > 0) {
    cad = Serial.readString();
    pos = cad.indexOf(',');
    cad1= cad.substring(0,pos);
    cad2= cad.substring(pos+1);
    if(velocidad[0] != cad1.toInt()){
      velocidad[0] = cad1.toInt();
    }
    if(velocidad[1] != cad2.toInt()){
      velocidad[1] = cad2.toInt();
      }
      }
  bool ok = radio.write(velocidad, sizeof(velocidad));

}
```

Figura 41 –Bucle loop() Arduino

5.3.2.Arduino UNO

- **Librerías**

- **SPI.h**: Librería utilizada para realizar la comunicación SPI o Serial Peripheral Interface que es un protocolo de datos en serie síncrono que utilizan los microcontroladores para comunicarse rápidamente con uno o más dispositivos periféricos en distancias cortas. También se puede utilizar para la comunicación entre dos microcontroladores.[9]

- **RF24.h**: Es el driver de radio, biblioteca de capa 2 OSI para la comunicación de los módulos NRF24L01.

- **nRF24L01.h**: es una parte de la librería RF24 que incluye un grupo de macros constantes.

- **Desarrollo**

  Al comienzo se incluyen las librerías nombradas.

```
#include <SPI.h>
#include <RF24.h>
#include <nRF24L01.h>
#define CE_PIN 7
#define CSN PIN 8
```

Figura 42 – Librerías de Arduino

Se declaran las variables necesarias tanto para el control de los motores como para la comunicación nRF24l01.

```
RF24 radio(CE_PIN, CSN_PIN);

byte direccion[5] ={'c','a','n','a','l'};

int velocidad[4];

//RUEDA DERECHA
int ENB_IZQ = 6;
int IN3 = 9;
int IN4 = 10;

//RUEDA IZQUIERDA
int ENA_DER = 3;
int IN1 = 5;
int IN2 = 4;

int v_der = 0;
int v_izq = 0;


int speed = 0;

int pinMotorA[3] = { ENA_DER, IN1, IN2 };
int pinMotorB[3] = { ENB_IZQ, IN3, IN4 };
```

Figura 43 – Definición de variables y de pines a utilizar

Se inicializa la comunicación serial, la comunicación por radio y se inicializan los pines de salida los cuales estarán conectados al puente H y harán el control de los motores.

```
void setup()
{

  Serial.begin(115200);
  radio.begin();
  radio.openReadingPipe(1, direccion);
  //radio.setPALevel(RF24_PA_MIN);
  radio.setDataRate(RF24_250KBPS);
  radio.startListening();

  //HABILITA DERECHA
  pinMode(ENA_DER, OUTPUT);  // ENA
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);

  //HABILITA DERECHA
  pinMode(ENB_IZQ, OUTPUT);  // ENB
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);


}
```

Figura 44 – Inicialización de comunicaciones y habilitación de pines

Se definen funciones que darán las señales de movimiento de forma más fácil. Las funciones son para avanzar, retroceder y detenerse. A las cuales es necesario indicarle el motor y la consigna de velocidad para el motor.

```
void move_adelante(const int pinMotor[3], int speed)
{
   digitalWrite(pinMotor[1], HIGH);
   digitalWrite(pinMotor[2], LOW);

   analogWrite(pinMotor[0], speed);
}

void move_atras(const int pinMotor[3], int speed)
{
   digitalWrite(pinMotor[1], LOW);
   digitalWrite(pinMotor[2], HIGH);

   analogWrite(pinMotor[0], speed);
}

void stop(const int pinMotor[3])
{
   digitalWrite(pinMotor[1], LOW);
   digitalWrite(pinMotor[2], LOW);

   analogWrite(pinMotor[0], 0);
}
```

Figura 45 – Definición de funciones

El código principal se encarga de revisar si hay algún dato que esté ingresando por la comunicación SPI del módulo nRF24l01. Luego son asignados a los motores derecho e izquierdo.

```
void loop(){

  if (radio.available()){
     radio.read(velocidad, sizeof(velocidad)); //Leemos los datos y los guardamos en la variable velocidad
  }

  v_der = velocidad[1];
  v_izq = velocidad[0];
```

Figura 46 – Recepción mediante módulo nRF24l01

Se revisa las velocidades recibidas si se encuentran entre el rango de 0 a 100 o de 100 a 200 para reconocer la dirección del movimiento en caso de ser consigna de retroceso se le dará signo negativo. A su vez dado que las consignas analógicas a los motores son del rango 55 a 250 se escalan los valores.

```
if (v_der > 100 && v_der<=200){
  v_der =(v_der - 100)*2 + 55;

}else if (v_der > 0 && v_der<100) {
  v_der = (v_der - 100)*2 - 55;
}
else {
  v_der=0;
  }

if (v_izq > 100 && v_izq<=200){
  v_izq = (v_izq - 100)*2 + 55;
}
else if (v_izq > 0 && v_izq<100) {
  v_izq = (v_izq - 100)*2 - 55;
}
else{v_izq=0;}
```

Figura 47 – Procesamiento de los valores recibidos

Por último, dependiendo del signo de la consigna se procede a llamar a la función que corresponda para cada motor y aplicar la velocidad deseada.

```
if(v_der > 55 && v_der <= 255){
 move_adelante(pinMotorA, v_der);

}else if(v_der < -55 && v_der >= -255){
  move_atras(pinMotorA, v_der*(-1));

}else if(v_der==0){
  stop(pinMotorA);
}

 if(v_izq > 55 && v_izq <= 255){
  move_adelante(pinMotorB, v_izq);

}else if(v_izq < -55 && v_izq >= -255){
  move_atras(pinMotorB, v_izq*(-1));

}else if (v_izq==0){
  stop(pinMotorB);
}
```

Figura 48 – Implementación de las funciones

6. RESULTADOS LOGRADOS

Después de un largo proceso de realización del proyecto se lograron realizar los objetivos planteados inicialmente. Se describirán uno a uno y se mostrarán los resultados logrados.

- Lograr la captura y procesamiento de imágenes de los gestos realizados por las manos del usuario, transformando estos en datos de velocidad.

Gracias a la ayuda de Python y la librería de OpenCV se pudo cumplir con este objetivo. A partir de la posición angular de ambas manos se definieron las velocidades de cada rueda, creando una interfaz cómoda e instintiva para el usuario. Se logró el reconocimiento de cada mano y su distinción entre derecha e izquierda y se estableció la necesidad de la presencia de ambas manos para poder conducir el vehículo, creando así condiciones prefijadas para el correcto funcionamiento.

- Añadir gestos para comenzar o pausar el proceso.

Al poder reconocer cuál mano es la derecha y cuál la izquierda, se pudo implementar el uso de un gesto de comienzo y uno de pausa. Al cerrar la mano derecha, el programa comenzará a transformar la posición angular en velocidad y luego al cerrar la mano izquierda, se pasará a modo pausa.

En la figura 1 comienza en modo pausa, por más que el usuario ponga las manos delante de la pantalla, el control no comenzará. Para poder darle inicio utilizamos el gesto de cerrar la mano derecha como en la figura 3. Las siguientes tres figuras muestran valores máximos y mínimos de cada rueda y para finalizar o poner en pausa el control, se cierra la mano izquierda.

Figura 49 – Modo pausado

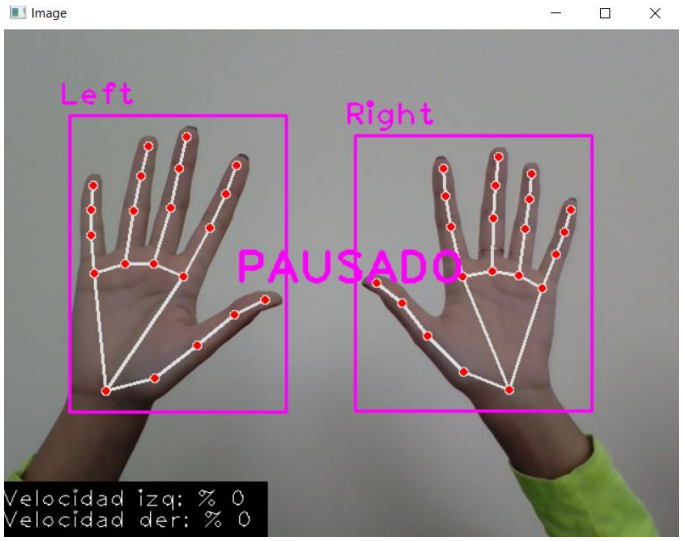

Figura 50 – Modo pausa con manos presentes

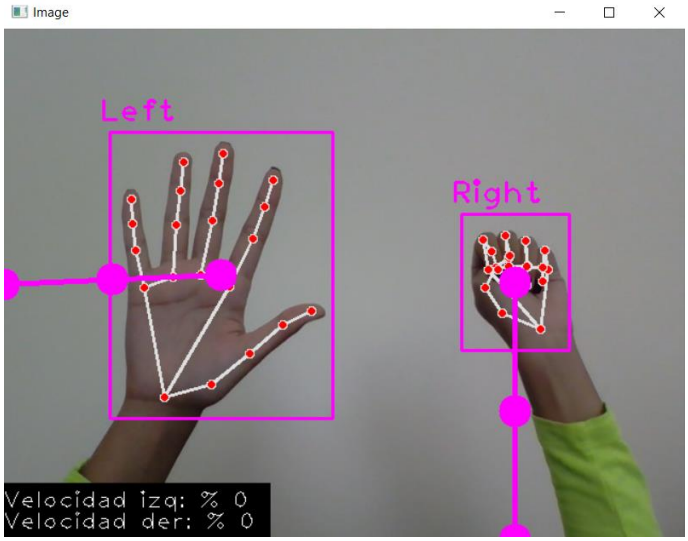

Figura 51 – Gesto de activación

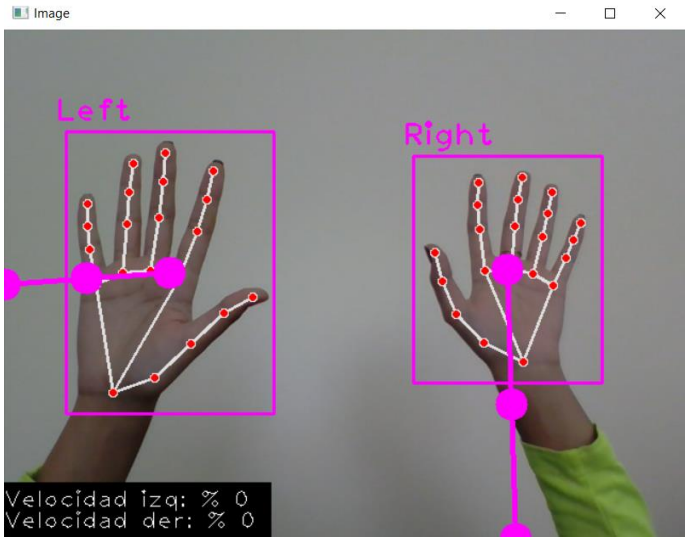

Figura 52 – En funcionamiento, posiciones angulares nulas

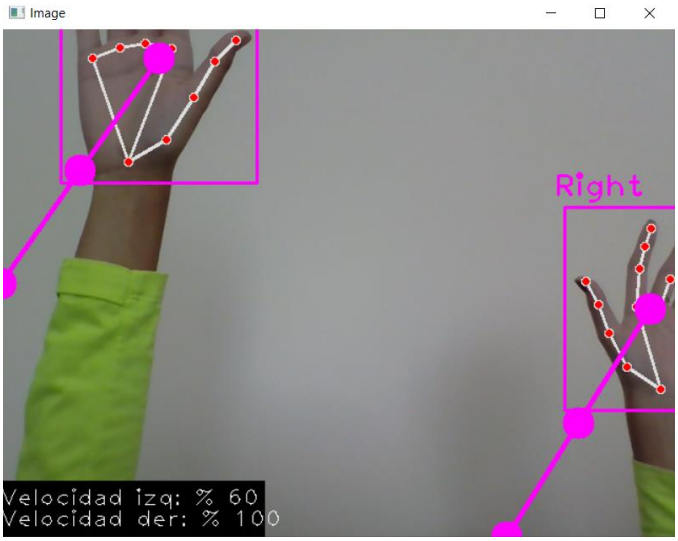

Figura 53 – En funcionamiento, posiciones angulares máximas

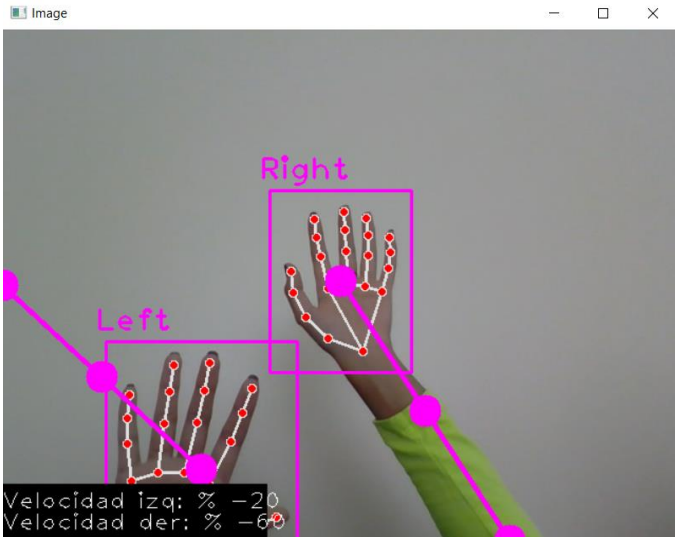

Figura 54 – En funcionamiento, posiciones angulares mínimas

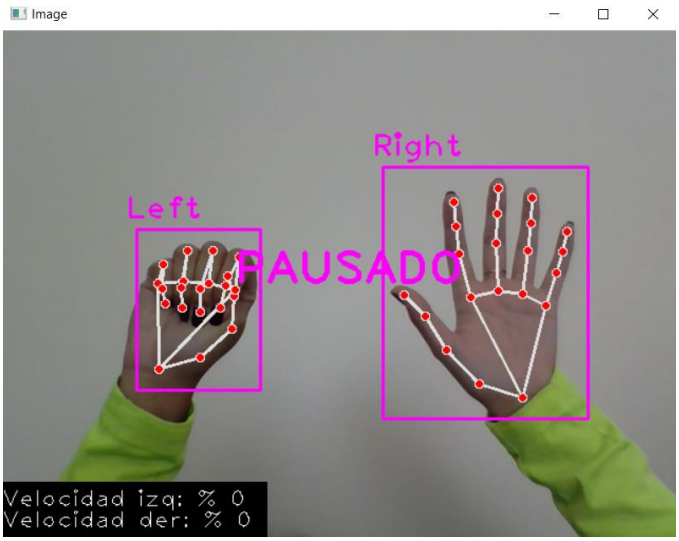

Figura 55 – Gesto de activación del modo pausado

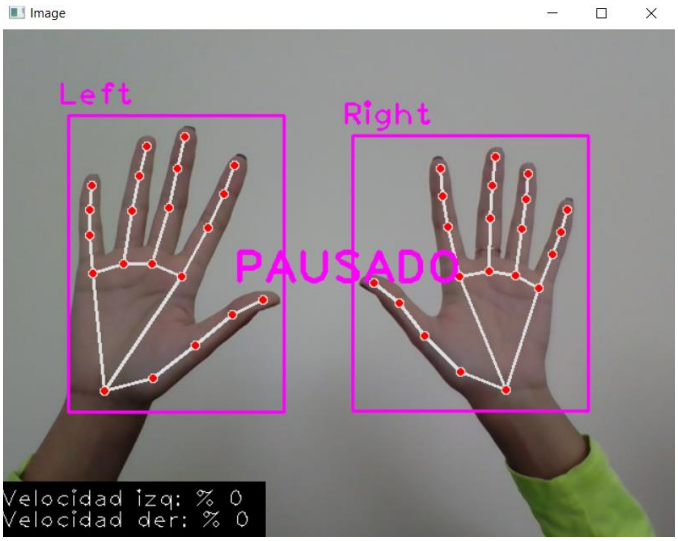

Figura 56 – Modo pausa con manos presentes

- Utilizar ArUco para realizar funciones determinadas.

Se implementó el uso de tres ArUco destinados a distintas funciones. Al presentarlos frente a la cámara, el automóvil realiza efectivamente las órdenes preestablecidas correspondiente a cada uno de ellos.

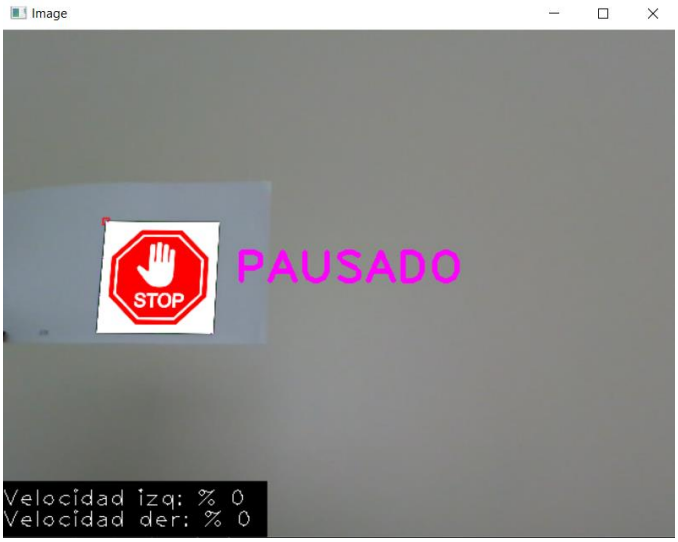

Figura 57 – ArUco IDE 1, pausado

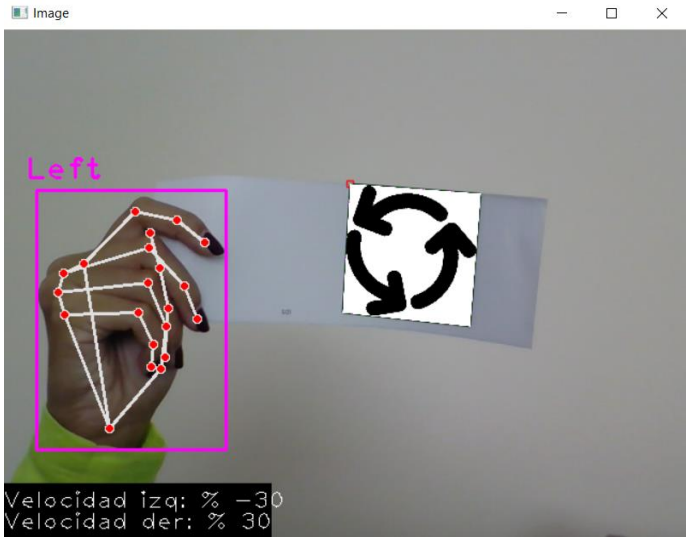

Figura 58 – ArUco IDE 2, giro en el lugar

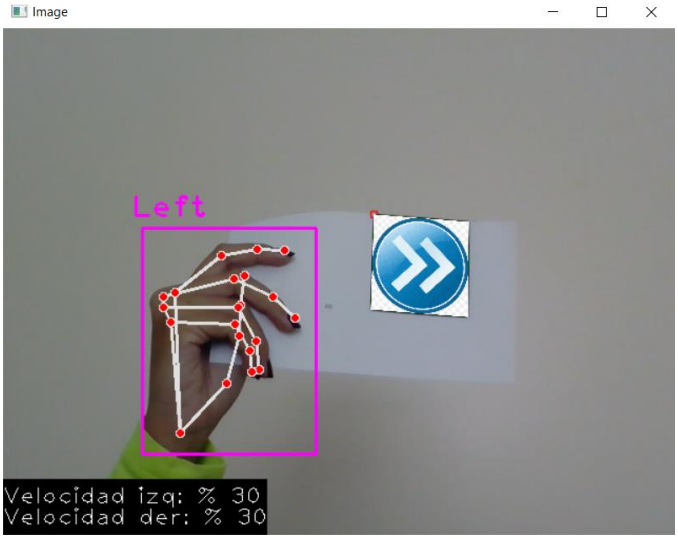

Figura 59 – ArUco IDE 3, avanzar

- Construir un automóvil de dos ruedas motorizadas.

Se logró la impresión del automóvil 3D y también la implementación de su parte electrónica sin ningún problema. Cabe destacar que los dispositivos electrónicos utilizados fueron 100% recicladas.

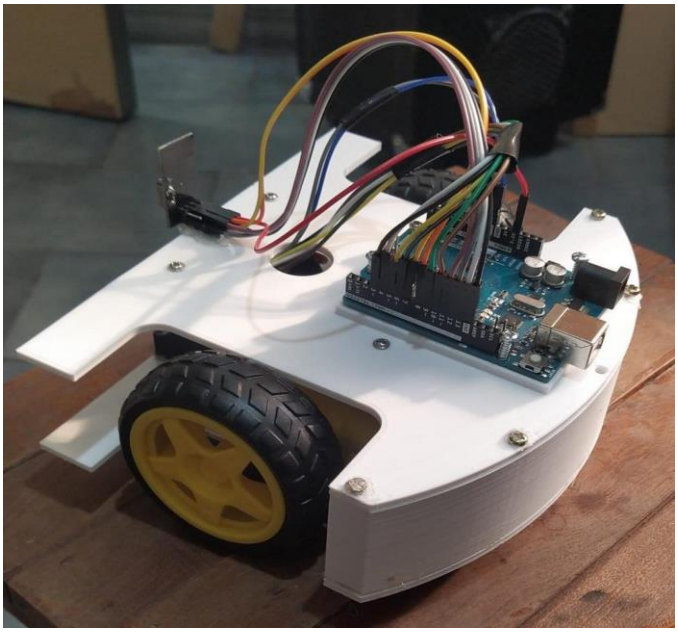

Figura 60 – Automóvil de dos ruedas motorizadas

- Lograr la comunicación serial entre la PC y un Arduino Mega.

Si bien la comunicación fue posible y el envío y recepción de datos se realizó correctamente, en esta transmisión se introduce una latencia de un segundo, lo cual es demasiado y hace que la conducción no sea verdaderamente en tiempo real. Dentro de todo el objetivo fue cumplido, pero puede ser mejorado.

- Establecer la comunicación entre el Arduino Mega y el Arduino Uno ubicado en el automóvil mediante los módulos nRF24l01.

Esta comunicación fue un éxito, no incluyó ningún tipo de latencia y los datos son enviados y recibidos correctamente. Se puede decir que la implementación de los módulos nRF24l01 fue efectiva y además fue sencilla a la hora de su programación. Cabe destacar que las distancias que maneja no son muy grandes, pero si están dentro de los límites requeridos para este proyecto.

- Accionar los motores según las órdenes recibidas.

Luego de pasar por las diferentes transmisiones, los datos llegan al fin al Arduino UNO, donde son procesados por última vez y finalmente se envía la orden a los motores, haciendo mover el automóvil en la dirección deseada. Este objetivo fue efectivamente logrado una vez recibidos los datos.

7. CONCLUSIONES

Para concluir con este proyecto se debe destacar que todos los objetivos planteados al principio fueron logrados. Si bien se presentaron varias dificultades, se supieron sobrellevar de la mejor manera que se tenía al alcance.

La implementación del reconocimiento de gestos se llevó a cabo sin dificultades y se realizaron numerosas pruebas de distintas posibilidades y gestos a reconocer, hasta que se llegó a la decisión de las posiciones angulares como el método más óptimo. Su implementación fue efectiva y sin fallos, junto con los comandos de start/stop hicieron que la conducción del automóvil sea cómoda e intuitiva para el usuario.

La comunicación fue la parte más compleja y aquella que presentó más problemas a la hora de ser desarrollada. Si bien los módulos nRF24l01 funcionaron sin problema, la comunicación serie entre Python y el Arduino Mega llevó un poco más de tiempo de lo que se esperaba. No se pudo evitar la inclusión de latencia al procesamiento con esta transmisión, siendo el mayor inconveniente de este proyecto y quizá se pueda decir que el manejo no es en tiempo real. Esto podría haber sido solucionado con la implementación de una Rapsberry, pero esto escapaba del presupuesto disponible, así que de deja planteado como una posible mejora a futuro de este proyecto.

Finalmente, la construcción y el ensamblaje del automóvil fueron llevado a cabo con piezas 100% recicladas. Su programación también fue efectiva, lo que llevó a un logro completo de los objetivos planteados. Los datos son recibidos sin inconvenientes y los motores responden favorablemente a las órdenes dadas por el usuario.

8. REFERENCIAS

[1] https://opencv.org/about/

[2] https://www.arduino.cc/en/software

[3] https://docs.arduino.cc/hardware/mega-2560

[4] https://docs.arduino.cc/hardware/uno-rev3

[5] https://descubrearduino.com/nrf24l01/

[6] http://robots-argentina.com.ar/didactica/control-de-motores-de-corriente-continua-con-puente-h/

[7] https://www.bolanosdj.com.ar/MOVIL/ARDUINO2/MotoresConRuedasArd.pdf

[8] https://www.thingiverse.com/thing:2450012

[9] https://www.arduino.cc/en/reference/SPI