



Universidad Nacional de San Luis

FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS Y NATURALES
DEPARTAMENTO DE ELECTRÓNICA

INGENIERÍA ELECTRÓNICA CON ORIENTACIÓN EN SISTEMAS DIGITALES

VISIÓN ARTIFICIAL EN LA ROBÓTICA

Proyecto final de materia

Materia:
Procesamiento Digital de Señales II

Docentes:
Mg. Ing. Ricardo Petrino
Ing. Jesús García

Autor:
Pablo Ezequiel Córdoba

2.^o cuatrimestre de 2022

Índice

1. Introducción	2
2. Desarrollo	2
2.1. Componentes	2
2.1.1. Raspberry Pi 3B	2
2.1.2. Raspberry Pi Camera V2	3
2.1.3. Driver de motores	3
2.1.4. Fuente reguladora de tensión	4
2.1.5. Fuentes de energía	4
2.1.6. Motores y carcasa	5
2.2. Robot completo	6
2.3. Programación	7
2.3.1. captura()	7
2.3.2. curvas_90()	9
2.3.3. seguidor_lineas()	9
2.3.4. motores(izq,der)	11
2.3.5. interseccion()	12
2.4. Funcionamiento	15
3. Conclusión	15
A. Anexo	17
A.1. Código seguidor de línea y rectas de 90 grados	17
A.2. Detector de intersecciones	22

1. Introducción

Se tiene como objetivo de este proyecto poder volcar los conocimientos adquiridos a lo largo de la cursada de la materia para resolver un problema, en particular se decidió aplicar la visión artificial para el desarrollo de un robot seguidor de líneas, el cual está basado en el reglamento oficial de la competencia mundial de robótica Robocup. [1]

2. Desarrollo

2.1. Componentes

Para el desarrollo y el armado del robot, se utilizó el siguiente hardware.

2.1.1. Raspberry Pi 3B

Es el ordenador encargado de realizar todo el procesamiento, desde el manejo de motores a través de sus pines GPIO, hasta el procesamiento de imágenes. En la Figura 1 se muestra la misma con su correspondiente pinout. [2]

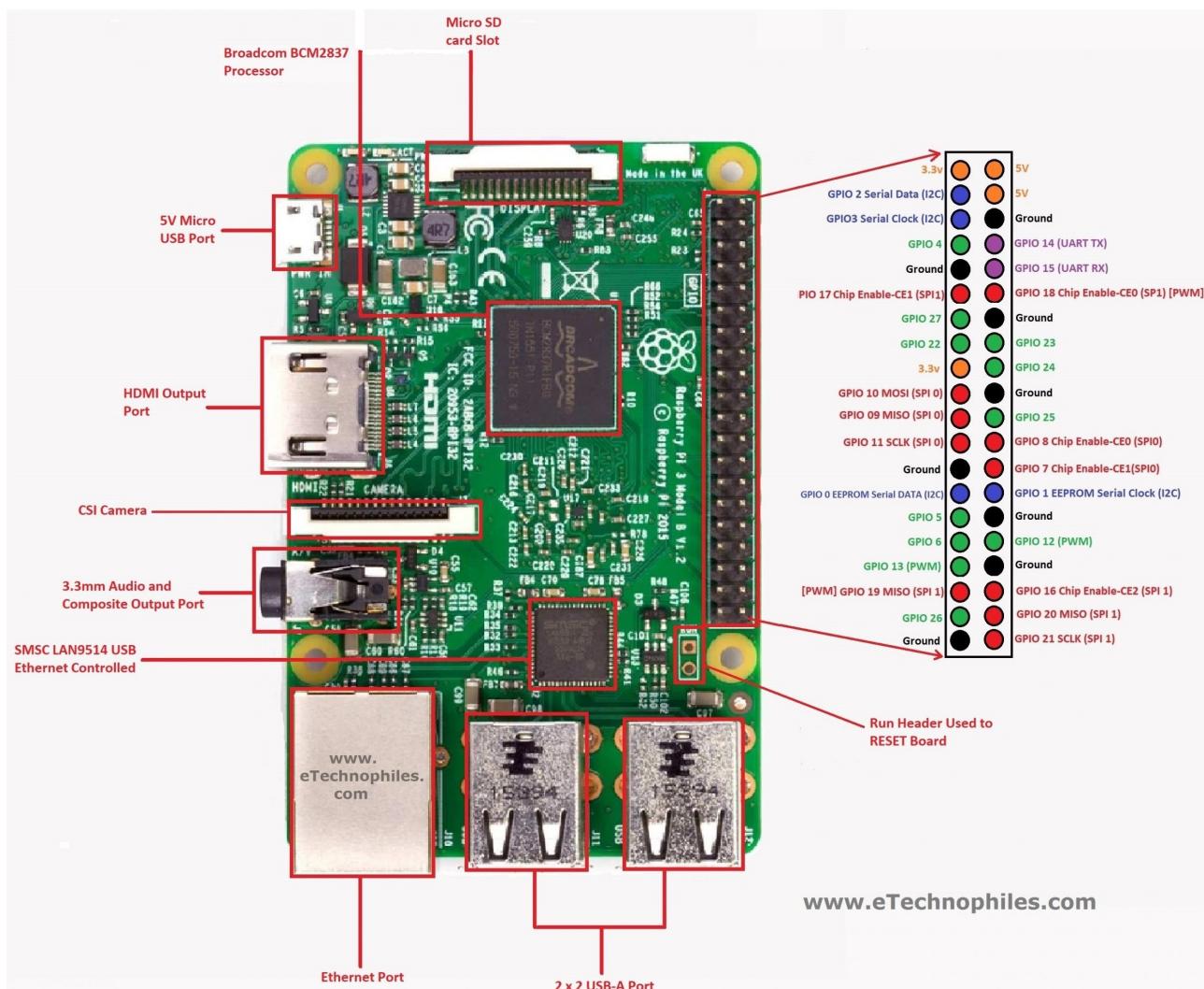


Figura 1 – Raspberry Pi 3B.

2.1.2. Raspberry Pi Camera V2

Es la cámara utilizada para la captura de imágenes, la misma posee una resolución de 8 Mpx con distintas resoluciones como FPS para realizar videos o sacar fotos [3]. La misma se conecta en el puerto CSI de la Raspberry Pi 3B, a continuación en la Figura 2 se ilustra el módulo mencionado previamente.



Figura 2 – Raspberry Pi Camera V2

2.1.3. Driver de motores

El controlador para los motores utilizado es el TB6612FNG, permitiendo manejar dos motores por módulo, según las señales de dirección y PWM según corresponda [4]. El pinout del mismo se muestra en la Figura 3. Vale aclarar que el mismo requiere de dos alimentaciones, una para el manejo de motores (VM) y otra para la lógica (VCC) siendo esta última de 5 V.

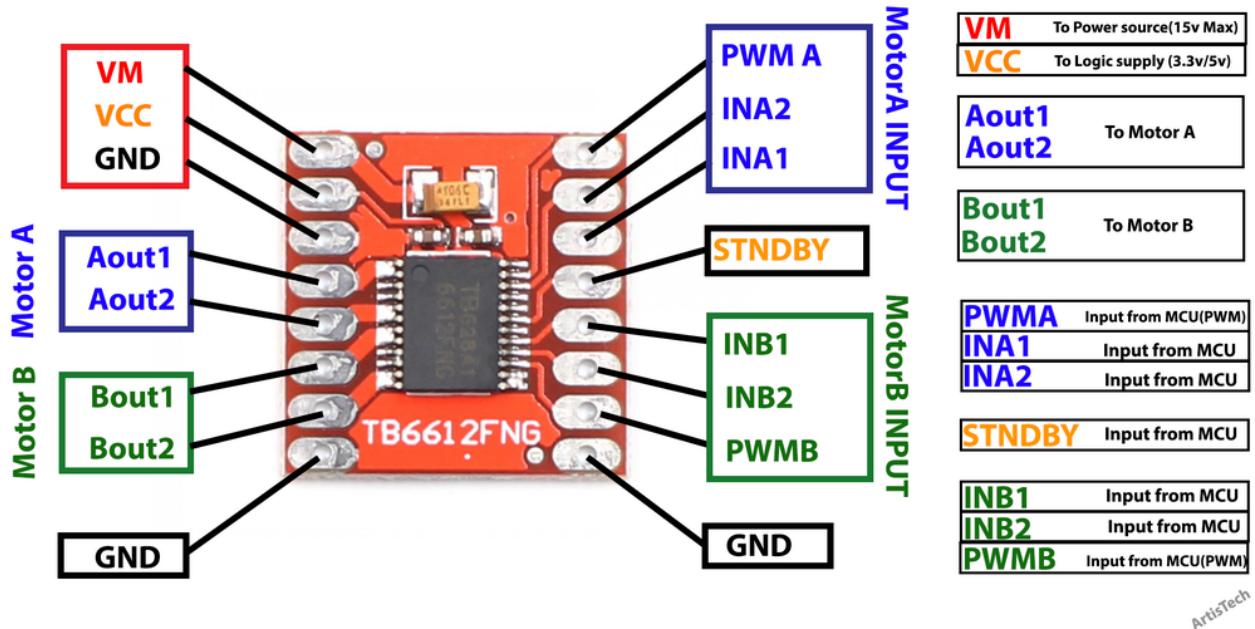


Figura 3 – Driver TB6612FNG.

2.1.4. Fuente reguladora de tensión

Para alimentar la lógica del controlador de motores mencionado previamente (5 V), se requiere de una fuente reguladora de tensión, particularmente step-down (reductora de tensión), ya que como se verá más adelante, los motores utilizados y por lo tanto, la fuente que los alimenta, es de 12 V. Por lo antedicho, la fuente reguladora step-down implementada es LM2596 [5]. Luego, en la Figura 4 se ilustra la misma.

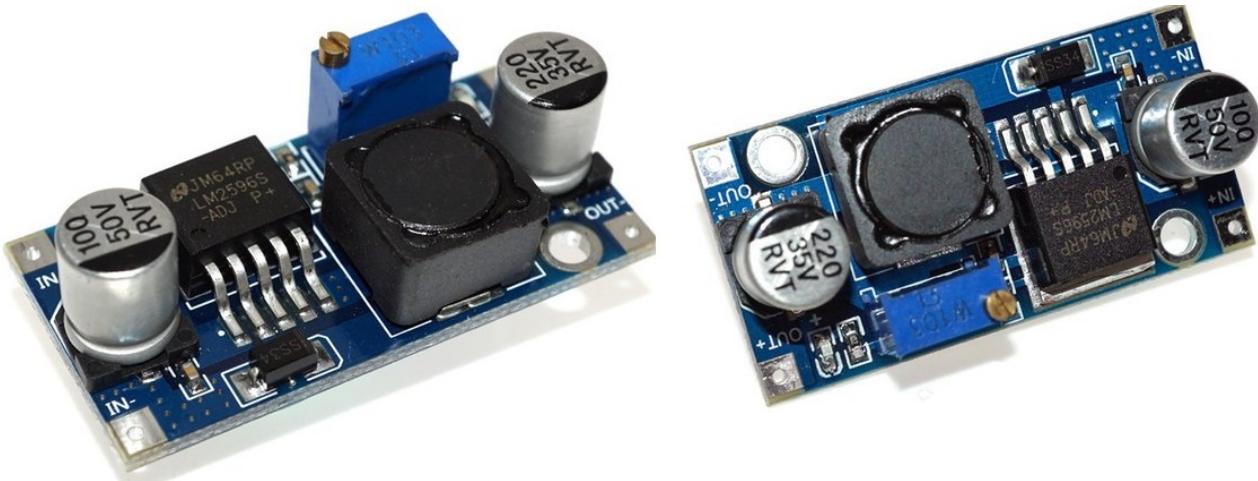


Figura 4 – Fuente Step Down

2.1.5. Fuentes de energía

Como se mencionó en la sección anterior, la fuente que provee energía a los motores y a la lógica del controlador de los mismos, es de 12 V (11,1 V nominal). La misma se trata de una batería LiPo, es decir compuesta de Litio y Polímero, particularmente el modelo utilizado para este proyecto, posee 3 celdas y una capacidad de 2250 mAh [6], como se puede observar en la Figura 5.



Figura 5 – Batería LiPo

Luego, para alimentar a la placa Raspberry Pi 3B, se optó por la utilización de una batería externa Samsung EEB-E11CWE, la cual se muestra en la Figura 6, las características que posee son:

- Entrada y salida: 5 V - 1.8 A (corriente continua).
- Capacidad: 9000 mAh.

Vale destacar que dicha batería ya no se encuentra en el mercado.



(a) Vista frontal.

(b) Vista trasera.

(c) Vista superior.

Figura 6 – Batería externa.

2.1.6. Motores y carcasa

Por último, los motores utilizados son unos de corriente continua de 1,5 a 12 V y de 400 mA con el motor detenido, estos se utilizan para tener una tracción diferencial, por medio de una caja reductora con engranajes de alta resistencia y bujes metálicos. La carcasa es la de un robot modelo N6, en la Figura 7 se ilustra un ejemplo, estos componentes mencionados son los que fabricaba la empresa argentina de robótica Robotgroup, la cual su página web ya no se encuentra disponible, pero para mayor información se puede visitar su página de Facebook. [7]



Figura 7 – Carcasa N6 implementada.

Luego, en la Figura 8 se muestra uno de los motores utilizados.

Cabe destacar que para este proyecto se puede utilizar cualquier motor de corriente continua, como así la carcasa del mismo.



(a) Vista superior.

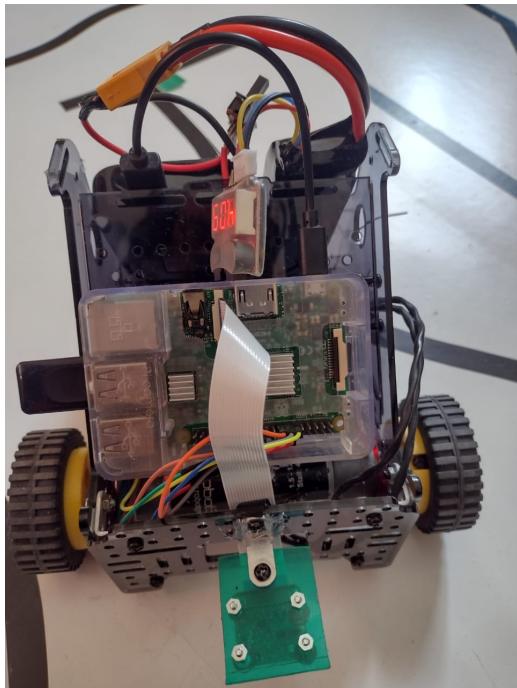


(b) Vista trasera.

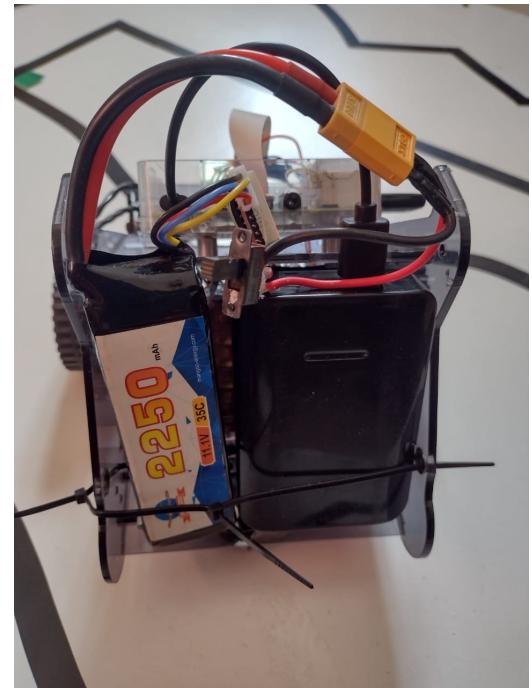
Figura 8 – Motores.

2.2. Robot completo

Con todo el hardware mencionado previamente, se procede a su armado y cableado, el cual da como resultado el robot que se muestra en la Figura 9.

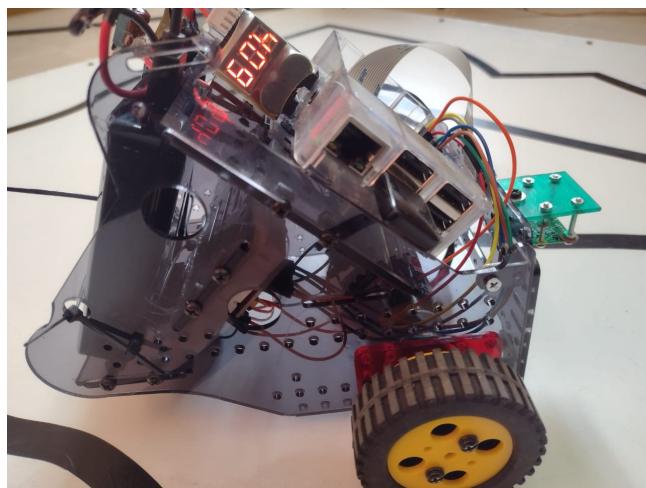


(a) Vista de frente.



(b) Vista trasera.

Figura 9 – Robot armado.



(c) Vista lateral.

Figura 9 – Robot armado. (Continuación)

2.3. Programación

Para lograr la detección de líneas, se requieren de las siguientes librerías:

- **cv2** [8]: Librería de OpenCV, la cual es la encargada de realizar todo tipo de operaciones sobre las imágenes.
- **RPi.GPIO** [9]: Manejo de los GPIO de la Raspberry Pi 3B, la misma ya viene instalada con el sistema operativo.
- **RC_lib** [10]: Pero precisamente la función **encontrar-bordes v0.0.13**, permite detectar bordes de objetos en las imágenes, graficar el contorno con cierto grado de personalización. Esta librería fue creada por los estudiantes Paul Andrés Romero Coronado y Pablo Ezequiel Córdoba para un trabajo de investigación de la asignatura Procesamiento Digital de Señales II.
- **Numpy**(opcional) [11]: Permite trabajar con funciones matemáticas, operar con matrices, vectores, etc.

El código adjunto en el apéndice, se puede observar que está compuesto por las configuraciones y variables necesarias previo a una función **while()**, la cual será verdadera si la captura de imágenes se encuentra abierta y la forma de cerrar la ejecución es cuando en el teclado se presiona la tecla "q", además de poseer una pausa con su respectiva reanudación si la tecla "p", es presionada.

Luego, dentro de dicho **while()** se observa que está compuesto de tres funciones, las cuales son: **captura()**,**curvas_90()** y **seguidor_lineas()**.

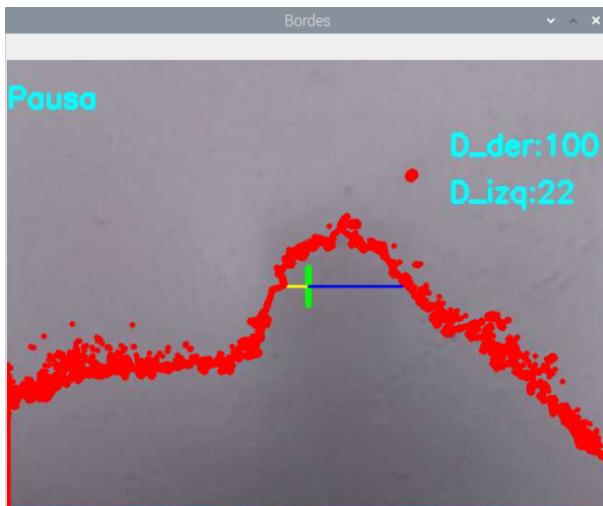
2.3.1. **captura()**

Esta función como bien indica su nombre, se encarga de realizar la captura de imagen, para luego obtener una imagen nueva con los bordes de la línea marcados invocando a la función **encontrar_bordes()**, se puede observar que esta última con respecto a la versión 0.0.11 (versión estable) recibe y devuelve un parámetro más, y este es un valor que está relacionado con la umbralización de la imagen.

El parámetro de salida indica con qué valor de umbralización se binarizó la imagen, mientras que la de entrada establece un límite máximo para que se considere la umbralización, esto sirve como una medida de calibración.

En la siguiente imagen se muestra un ejemplo de lo mencionado anteriormente, en la Figura 10a, se observa el resultado de colocar un límite de umbralización muy alto (255), mientras que en la Figura 10b se visualiza el resultado de un umbral menor, como se ve, esto permite obtener la imagen

original sin marcar bordes, ya que como se verá más adelante, será una molestia para el correcto funcionamiento del autómata.



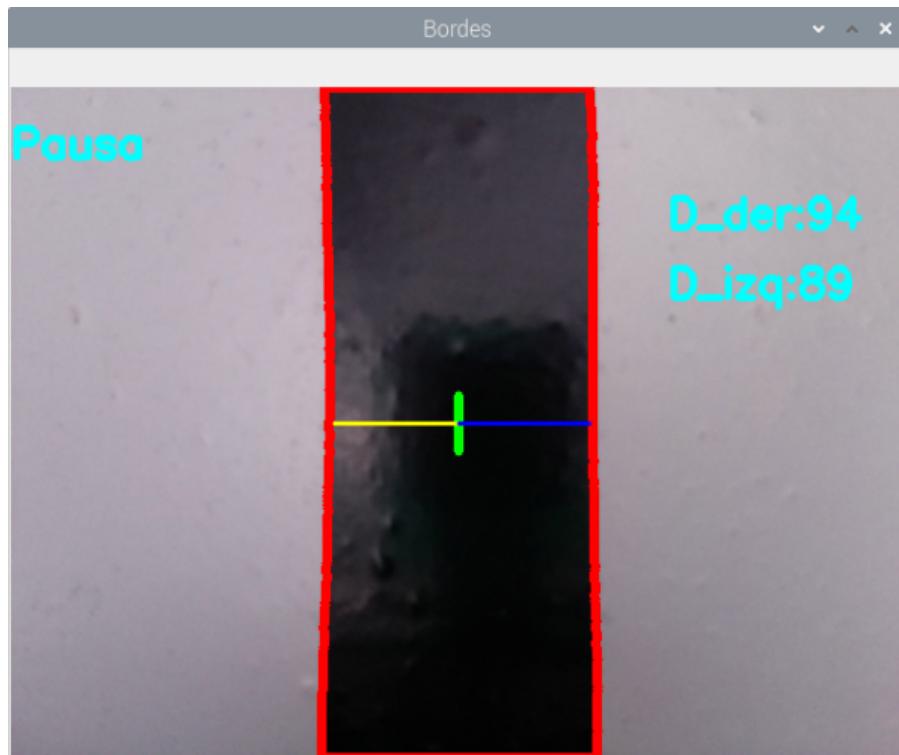
(a) Umbralización excesiva (255).



(b) Umbralización correcta (115).

Figura 10 – Diferencia con respecto a elegir distintos valores de umbralización.

Luego de obtener la imagen con los bordes marcados (o no si tal es el caso), se procede a calcular la distancia que hay de los bordes tanto derecho como izquierdo con respecto al centro de la imagen (marcado en verde), y la distancia que hay hacia el borde superior, esto se puede ver en la Figura 11. A su vez en la parte derecha de la misma, se indica con **D_der** y **D_izq** las distancias que hay con respecto a los bordes, los cuales se marcan con una traza amarilla para el borde izquierdo y con un trazo azul para el borde derecho.

**Figura 11** – Distancia a los bordes.

2.3.2. curvas_90()

En la misma se realiza la detección de curvas de 90 grados. Esto lo hace en base a las distancias de sus lados:

- Detección a la izquierda: Cuando la distancia al borde izquierdo supera el ancho aproximado de la línea y la distancia al borde derecho es menor al ancho aproximado de la línea.
- Detección a la derecha: En este caso es lo contrario, cuando la distancia al borde derecho supera el ancho y la distancia al borde izquierdo sea menor al ancho aproximado de la línea.

Luego, el caso que se haya detectado, el robot gira en su propio eje para el lado que se ha detectado una curva de 90 grados, hasta que detecte que ya se encuentra de nuevo dentro de la línea.

En la Figura 12 se muestran los casos en que se realiza la detección de curvas de 90 grados.

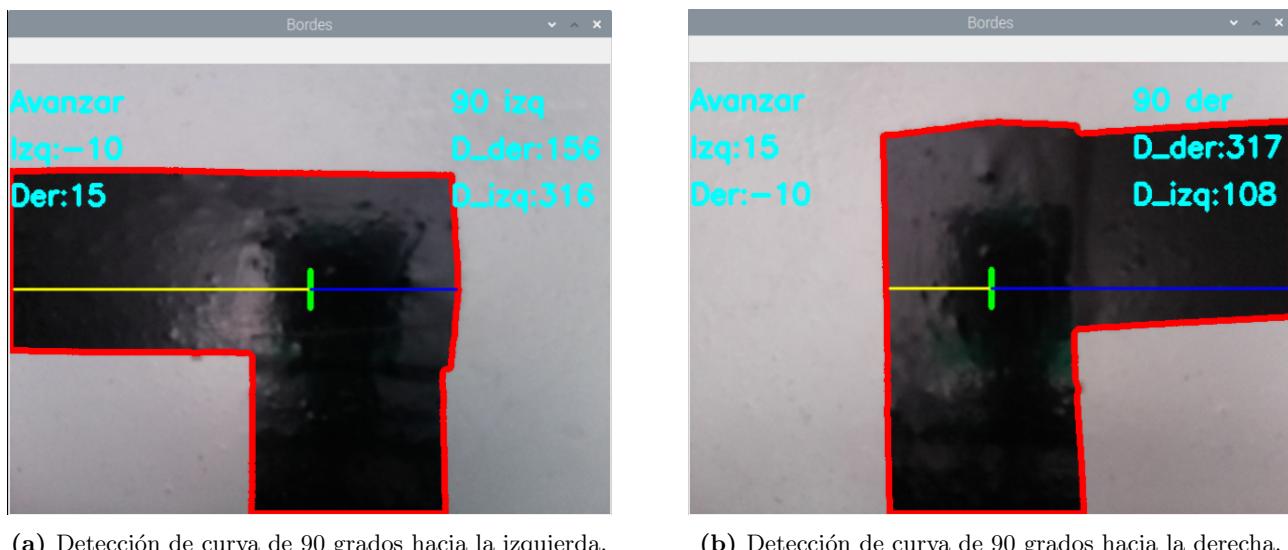
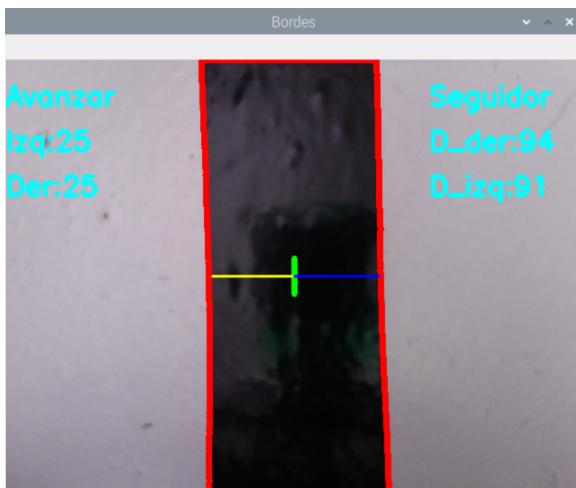


Figura 12 – Diferencia con respecto a elegir distintos valores de umbralización.

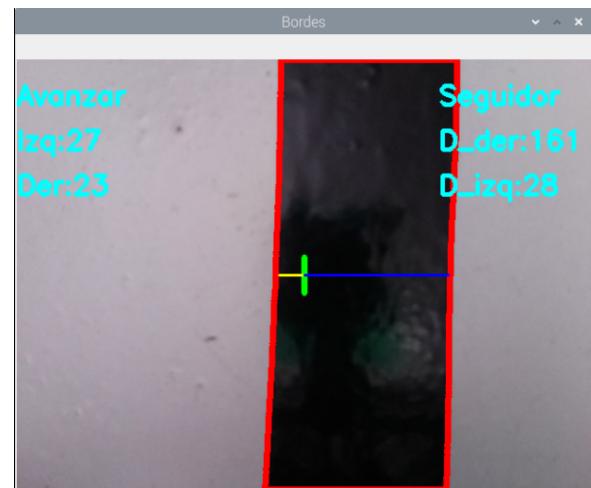
2.3.3. seguidor_lineas()

Esta función se encarga de acomodar el robot para que sea capaz de seguir la línea como su nombre lo indica, para entender el funcionamiento de esta, es necesario dividir el análisis en distintos tramos.

- **Dentro de la línea:** Cuando sucede dicho caso, las velocidades de ambos motores se acomodian según las distancias a los bordes laterales, es decir, que cuando se encuentra en el centro de la línea, ambos motores tendrán la misma velocidad, mientras que si se acerca al borde izquierdo, el motor de dicho lado tendrá mayor velocidad que el derecho; en cambio si ocurre lo opuesto, se acerca al borde derecho, el motor de este lado tendrá mayor velocidad que el izquierdo. Todos estos cambios ocurren para cada frame analizado, por lo tanto hay una corrección leve pero constante. En la Figura 13 se ilustra lo mencionado. Donde en la parte izquierda de las figuras se muestran las velocidades que tienen los motores.



(a) Corrección dentro de la línea. Centro.



(b) Corrección dentro de la línea. Lado izquierdo.



(c) Corrección dentro de la línea. Lado derecho.

Figura 13 – Corrección de velocidades de motores para distintos casos dentro de la línea.

- **Pasado del borde izquierdo:** En este caso, lo que sucede es que de forma brusca se cambia la velocidad de los motores para que gire lentamente hacia dentro de la línea, cuanto más alejado esté, mayor serán las velocidades para volver a ingresar, ya que se utiliza como referencia la distancia al borde derecho. Esto se ilustra en la Figura 14.



(a) Sobrepujo del borde izquierdo.



(b) Sobrepujo mucho más del borde izquierdo.

Figura 14 – Sobrepujo del borde izquierdo.

- **Pasado del borde derecho:** Es similar al descrito anteriormente, pero para el borde derecho. Como se ve en la Figura 15.

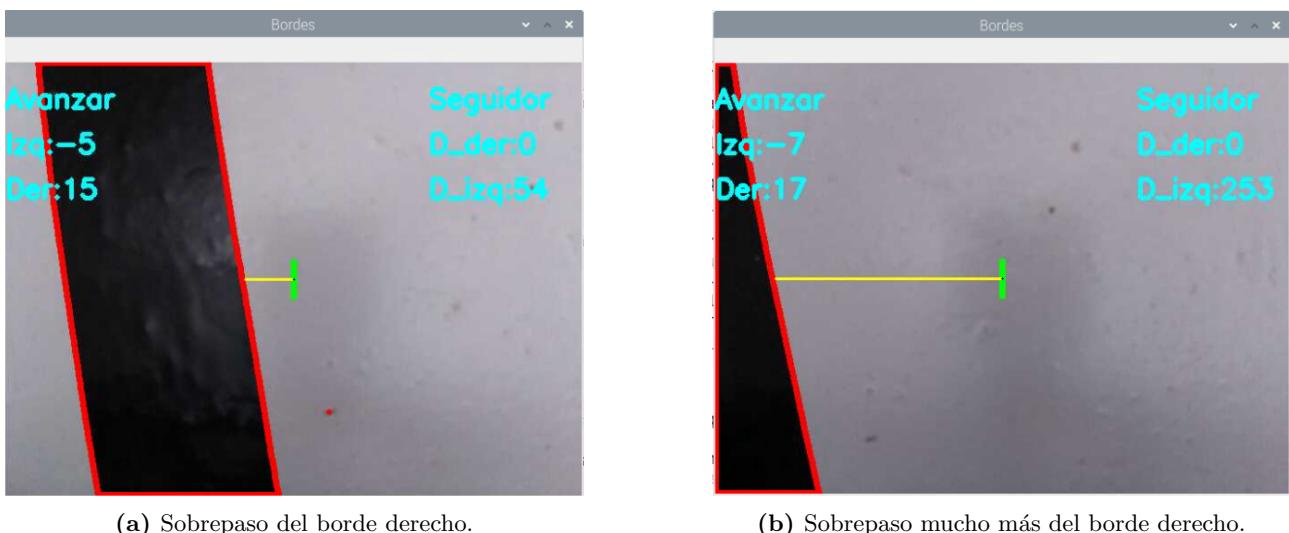


Figura 15 – Sobrepasso del borde derecho.

- **Volviendo a la línea luego de doblar brusco:** Es otra medida para que el robot quede lo más centrado posible al centro de la línea, esto ocurre cuando se ha detectado que se ha pasado de alguno de los bordes, entonces lo que se realiza es seguir disminuyendo la velocidad, hasta que aproximadamente se encuentre en el centro de la línea. Esto se observa en la Figura 16.

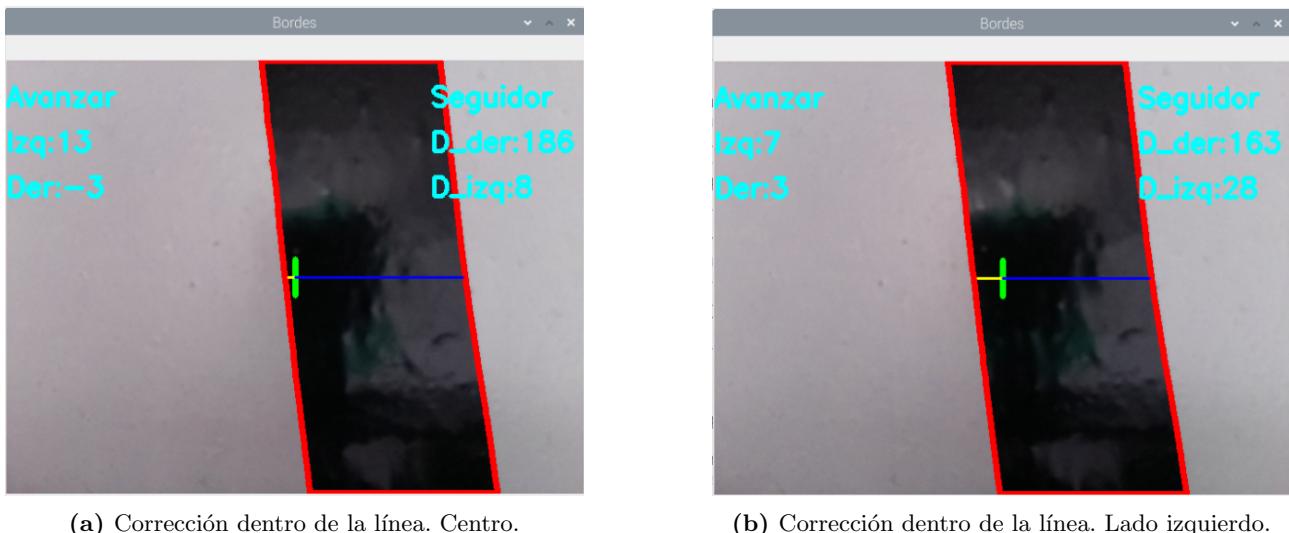


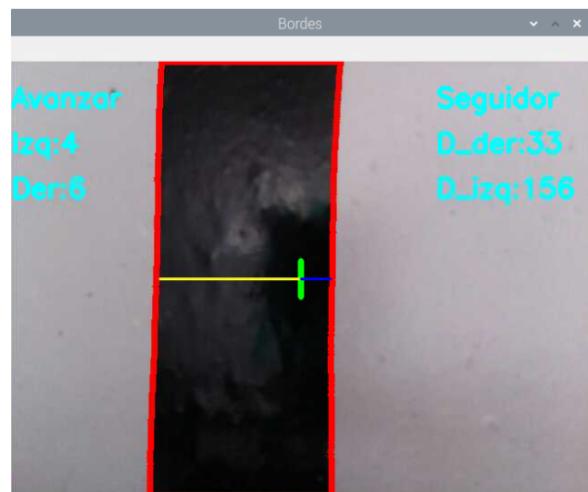
Figura 16 – Corrección de velocidades de motores para distintos casos dentro de la línea cuando ha sobrepasado uno de los bordes.

2.3.4. motores(izq,der)

Como se deduce con su nombre, esta función se encarga del manejo de los motores. Según las velocidades recibidas por parámetros, son los valores que se le asignan a cada una de las señales para el controlador de los motores. Cabe destacar que las velocidades que se reciben por parámetro van desde 0 a ± 100 . Es decir, frenado o a toda velocidad ya sea para un sentido u otro.



(c) Corrección dentro de la línea. Lado derecho.



(d) Corrección dentro de la línea. Lado derecho.

Figura 16 – Corrección de velocidades de motores para distintos casos dentro de la línea cuando ha sobrepasado uno de los bordes. (Continuación)

2.3.5. interseccion()

Esta no es una función que finalmente se implementó, sino que se ha planteado el problema para detectar intersecciones, según el reglamento mencionado, establece los casos que se ilustran en la Figura 17. *Nota: los cuadrados de color verde poseen un tamaño de 25 mm x 25 mm.*

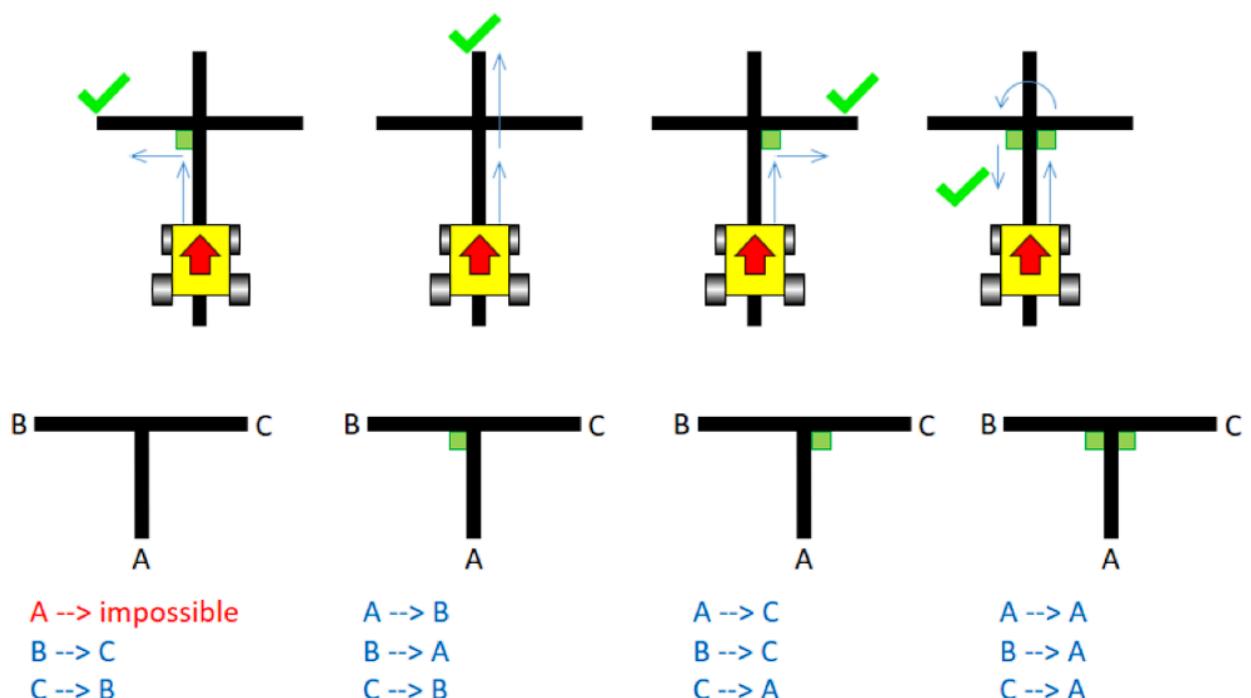


Figura 17 – Posibles casos de intersecciones.

Por lo que, para detectar la presencia de una intersección, se optó por el reconocimiento de colores usando el modelo HSV. Cuya representación se encuentra en la Figura 18.

- **H (Hue):** Es la representación de grado de ángulos cuyos valores posibles son entre 0 y 360°.
- **S (Saturation):** Se representa como la distancia al eje de brillo negro-blanco. Cuanto menor sea la saturación de un color, mayor tonalidad grisácea habrá y más decolorado estará.

- **V (Value)**: Representa la altura en el eje blanco-negro.

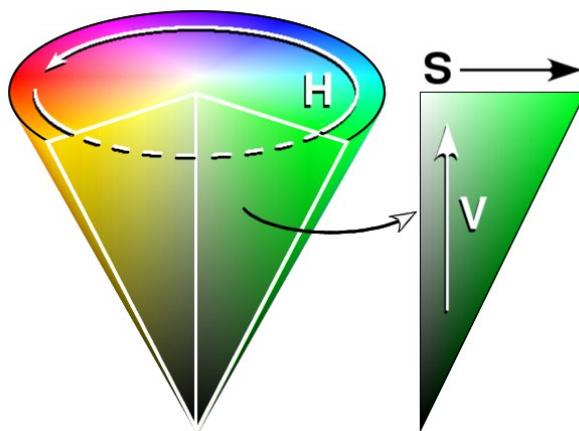


Figura 18 – Cono de colores del modelo HSV.

Como se observa en el anexo, lo que se realiza es analizar por frame. Primero se calibra o más bien se elige un rango de valores dentro del modelo HSV que se interpretan como el valor mínimo y máximo de verde deseado a detectar, esto resultará en una imagen que en el caso óptimo solamente se verá el cuadrado en cuestión. Luego, se aplican dos máscaras, partiendo la imagen en dos, en la que a cada una de ellas se le obtendrán los contornos y calcularán sus áreas. Luego, se deberá elegir un umbral para saber si el área medida se corresponde a un posible cuadrado verde para así tomar la decisión si esta se encuentra en el lado izquierdo o derecho de la línea, y por lo tanto doblar para el camino marcado.

En la Figura 19 se muestra el caso en que no se ha detectado un cuadrado verde.

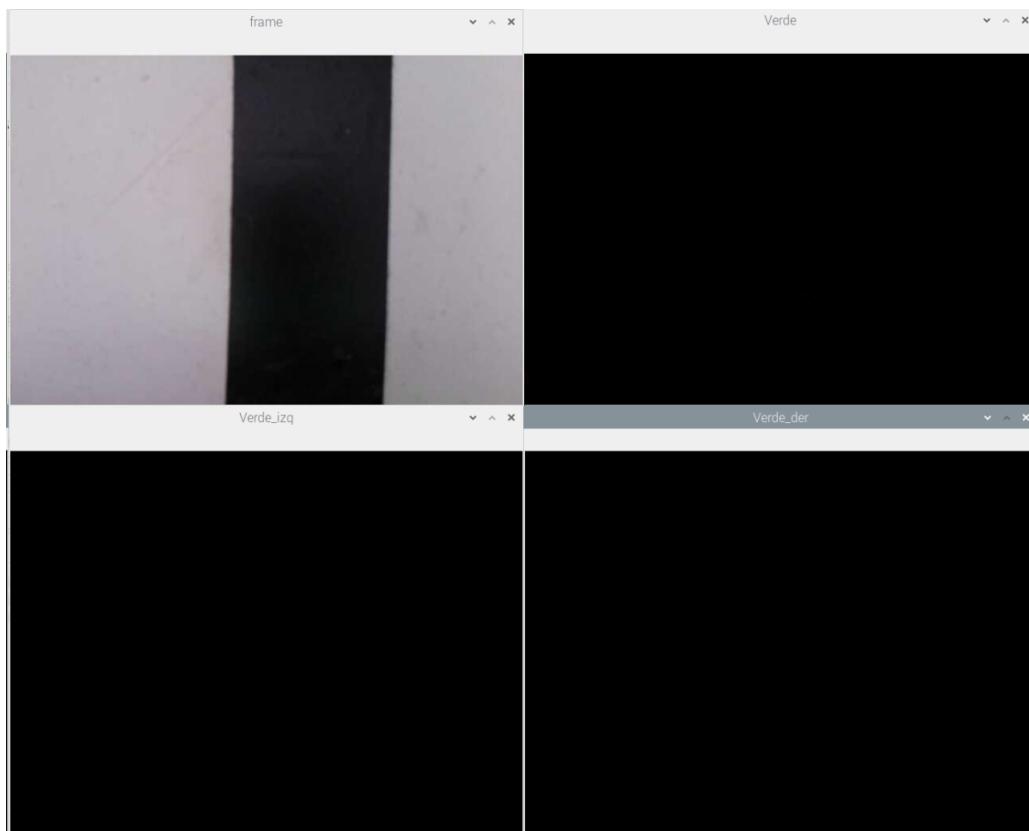


Figura 19 – Sin intersección.

Luego, en la Figura 20 se observa cuando se debe doblar hacia la izquierda.

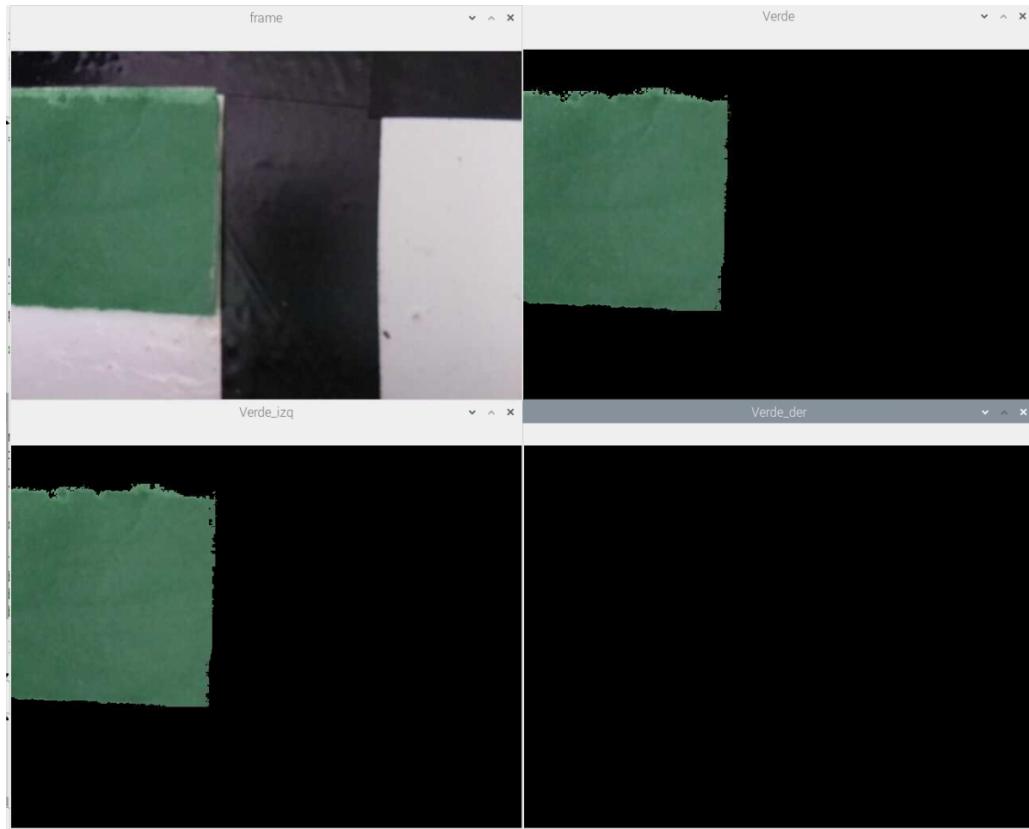


Figura 20 – Intersección. Doblar izquierda.

A continuación, en la Figura 21 se presenta el caso que se debe doblar hacia la derecha.

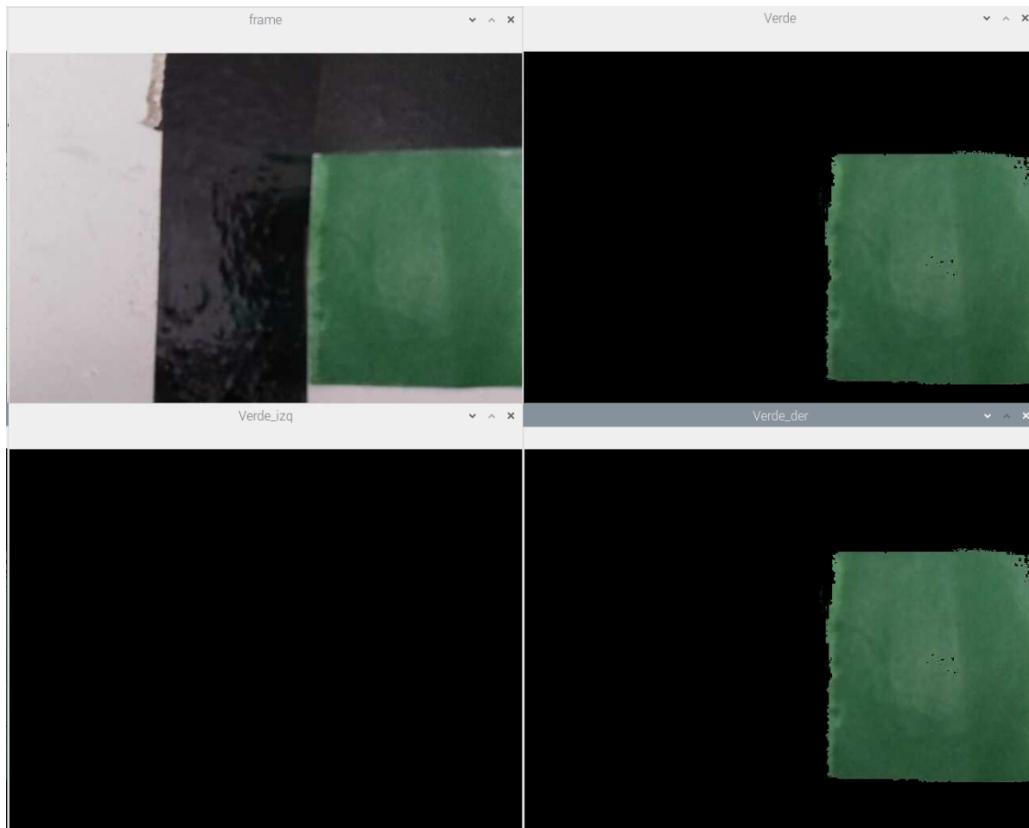


Figura 21 – Intersección. Doblar derecha.

Finalmente, en la Figura 22 se ilustra cuando debe volver por el camino en el que vino.

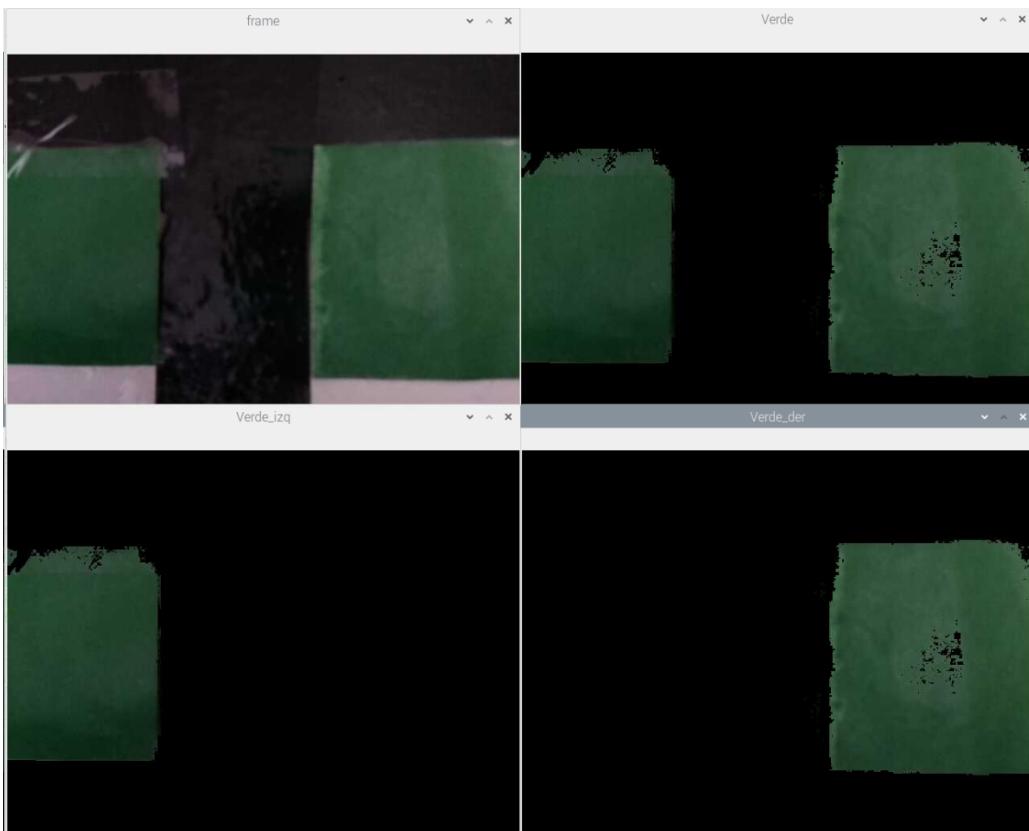


Figura 22 – Intersección. Volver por el mismo camino.

2.4. Funcionamiento

Para ver el funcionamiento del robot, se puede referir al [siguiente link](#), donde se observa como responde según lo que esté ocurriendo.

3. Conclusión

Con la realización del proyecto explicado en el presente informe, se pudo afianzar y reformar los conocimientos que se adquirieron a lo largo de la cursada de la correspondiente asignatura, ya que al realizar un sistema que responda en tiempo real, se tornó complicado debido al algoritmo que se debe plantear para resolver el problema, ya que el mismo no debe bloquear el funcionamiento de las demás partes que componen al sistema, como así elegir funciones que no requieran de tanto tiempo de procesamiento.

Uno de los primeros modelos del robot, fue la utilización de una web cam usb cualquiera, el problema que presentaba la misma era la reacción o el tiempo que se demoraba en poder actualizar un frame, y esto como consecuencia llevaba a que se sature el buffer, provocando una lentitud del procesamiento. Además, se implementaba una comunicación serie entre la Raspberry Pi y una placa de la empresa RobotGroup conocida como DuinoBot, la cual está basada en Arduino, esta comunicación se hacía para que la Raspberry Pi se independizara del manejo de motores. Pero no resultó ya que esta segunda placa, se alimentaba con pilas AA recargables, las cuales no poseen demasiada corriente para ser capaces de controlar los motores a bajas velocidades. Por lo que todo este conjunto de hechos conllevaba a un funcionamiento lento, tosco y fuera de los límites esperados.

Otro problema encontrado fue, ¿cómo hacer para detectar los bordes de la línea?, una de las primeras pruebas fue realizando la segunda derivada aplicada a imágenes, utilizando el Laplaciano, o Canny, pero como la misma webcam poseía ruido, este era relevante a pesar de realizar una etapa de filtrado.

Aunque tampoco se descarta la posibilidad de implementar otro algoritmo o una mejor implementación de las funciones que provee OpenCv, NumPy, etc.

A. Anexo

Los códigos mostrados en los siguientes anexos se encontrarán además para su descarga en el repositorio de GitHub [12].

A.1. Código seguidor de línea y rectas de 90 grados

```
1 import cv2
2 import numpy as np
3 from RC_lib import encontrar_bordes
4 import serial
5 import RPi.GPIO as GPIO
6 import time
7
8 cap = cv2.VideoCapture(0)
9 cap.set(cv2.CAP_PROP_BUFFERSIZE, 1)
10 cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
11 cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
12 #fourcc = cv2.VideoWriter_fourcc(*'DIVX')
13 #out = cv2.VideoWriter('Seguidor_90.avi',fourcc, 30, (640,480))
14 #----- configuracion texto en imagen
15 font = cv2.FONT_HERSHEY_SIMPLEX
16 position = (0,100)
17 position2 = (0,150)
18 position3 = (0,50)
19 position4 = (470,50)
20 position5 = (470,100)
21 position6 = (470,150)
22 fontScale = 1
23 fontColor = (255,255,0)
24 thick = 3
25 #
26 MOT_A1 = 17 #GPIO17 dir
27 MOT_A2 = 27 #GPIO027 dir
28 MOT_AE = 22 #GPIO022 enable
29
30 MOT_B1 = 25 #GPIO025 dir
31 MOT_B2 = 8 #GPIO08 dir
32 MOT_BE = 7 #GPIO07 enable
33
34 GPIO.setmode(GPIO.BCM) #le estamos indicando que el pin x que pondremos, sera el que
#esta etiquetado como GPIOx
35 GPIO.setup(MOT_A1,GPIO.OUT) #configurar de salida
36 GPIO.setup(MOT_A2,GPIO.OUT) #configurar de salida
37 GPIO.setup(MOT_AE,GPIO.OUT) #configurar de salida
38 GPIO.setup(MOT_B1,GPIO.OUT) #configurar de salida
39 GPIO.setup(MOT_B2,GPIO.OUT) #configurar de salida
40 GPIO.setup(MOT_BE,GPIO.OUT) #configurar de salida
41
42 #definir salidas pwm
43 pwm_a = GPIO.PWM(MOT_AE,100) # dividir en 100 trozos el segundo, configurando Enable
#para el PWM
44 pwm_b = GPIO.PWM(MOT_BE,100)
45
```

```

46 #inicializar los enable con duty cycle de 0
47 pwm_a.start(0)
48 pwm_b.start(0)
49
50 #flags y algunas variables
51 pausa = 1
52 vel_izq = 0
53 vel_der = 0
54 brusco_izq = 0
55 brusco_der = 0
56 curva_90_izq = 0
57 curva_90_der = 0
58 ancho_aprox = 200 #con valores medidos respecto a las distancias de los bordes,
      ancho de la linea
59
60 alto = 480
61 ancho = 640
62 def motores(izq,der):
63     #motor izquierda
64     if(izq >= 0): #velocidades positivas
65         GPIO.output(MOT_A1,True)
66         GPIO.output(MOT_A2,False)
67         pwm_a.ChangeDutyCycle(izq)
68
69     if(izq < 0): #velocidades negativas
70         GPIO.output(MOT_A1,False)
71         GPIO.output(MOT_A2,True)
72         pwm_a.ChangeDutyCycle(abs(izq))
73
74     #motor derecha
75     if(der >= 0): #velocidades positivas
76         GPIO.output(MOT_B1,True)
77         GPIO.output(MOT_B2,False)
78         pwm_b.ChangeDutyCycle(der)
79
80     if(der < 0): #velocidades negativas
81         GPIO.output(MOT_B1,False)
82         GPIO.output(MOT_B2,True)
83         pwm_b.ChangeDutyCycle(abs(der))
84
85 def seguidor_lineas():
86     global vel_izq,vel_der,brusco_izq,brusco_der,cont_izq,cont_der,dist_izq,dist_der,
          estado
87     vel_izq_ant = vel_izq
88     vel_der_ant = vel_der
89
90     if(not curva_90_izq and not curva_90_der):
91         estado = "Seguidor"
92         if(cont_izq == 1 and cont_der == 1 and not brusco_der and not brusco_izq): #si
              estan dentro de la linea negra
93             vel_izq = 25 - int((dist_izq-dist_der)*0.02)
94             vel_der = 25 - int((dist_der-dist_izq)*0.02)
95

```

```

96     elif(cont_izq == 0 and cont_der == 1):#si se paso del borde izquierdo
97         vel_izq = 15 + int(dist_der*0.01)
98         vel_der = -5 - int(dist_der*0.01)
99         brusco_izq = 1
100
101    elif(cont_izq == 1 and cont_der == 0):#si se paso del borde derecho
102        vel_izq = -5 - int(dist_izq*0.01)
103        vel_der = 15 + int(dist_izq*0.01)
104        brusco_der = 1
105
106    if(vel_der == 0 and vel_izq == 0): #para evitar los casos bordes, tomo el valor
107        anterior
108        vel_der = vel_der_ant
109        vel_izq = vel_izq_ant
110
111    if(brusco_izq and cont_izq and cont_der): #se centra en la linea cuando vuelve
112        de pasarse por el borde izquierdo
113        #print("Brusco izquierdo")
114        vel_izq = 15 - int((dist_izq)*0.2)
115        vel_der = -5 + int((dist_izq)*0.2)
116        if(dist_izq >= 50):
117            brusco_izq = 0
118
119    elif(brusco_der and cont_izq and cont_der): #se centra en la linea cuando vuelve
120        de pasarse por el borde derecho
121        #print("Brusco izquierdo")
122        vel_izq = -5 + int((dist_der)*0.2)
123        vel_der = 15 - int((dist_der)*0.2)
124        if(dist_der >= 50):
125            brusco_der = 0
126
127
128 def curvas_90():
129     global curva_90_izq,curva_90_der,vel_izq,vel_der,cont_izq,cont_der,dist_izq,
130             dist_der,dist_adelante,ancho_aprox,estado
131
132     if(dist_der >= ancho_aprox and dist_izq < ancho_aprox and dist_izq > 0 and not
133         curva_90_der): #detecta una curva a 90 grados a la derecha
134         #print("Se detecto una curva de 90 grados a la derecha")
135         curva_90_der = 1
136         motores(0,0)
137         estado = "90 der"
138         #time.sleep(3)
139
140     elif(dist_izq >= ancho_aprox and dist_der < ancho_aprox and dist_der > 0 and not
141         curva_90_izq): #detecta una curva a 90 grados a la izquierda
142         #print("Se detecto una curva de 90 grados a la izquierda")
143         curva_90_izq = 1
144         motores(0,0)
145         estado = "90 izq"
146         #time.sleep(3)
147
148     if(curva_90_der):
149

```

```

143     #print("girando derecha")
144     vel_izq = 15
145     vel_der = -10
146     #motores(20,-20)
147     if(cont_izq and cont_der and dist_adelante >= ancho_aprox):
148         curva_90_der = 0
149         brusco = 1
150
151 elif(curva_90_izq):
152     #print("girando izquierda")
153     vel_izq = -10
154     vel_der = 15
155     #motores(-20,20)
156     if(cont_izq and cont_der and dist_adelante >= ancho_aprox):
157         curva_90_izq = 0
158         brusco = 1
159
160 def captura():
161     global imagen_final,dist_izq,dist_der,dist_adelante,cont_izq,cont_der,
162         cont_adelante,thresh
163
164     imagen_final,bordes,thresh = encontrar_bordes(frame,"binv","ext","r",5,115) #
165         buscar bordes
166
167     dist_izq = 0 #distancia al borde izquierdo de la linea
168     dist_der = 0 #distancia al borde derecho de la linea
169     dist_adelante = 0 #distancia al borde superior
170     cont_izq = 0 #actualizar los flags
171     cont_der = 0
172     cont_adelante = 0
173
174     for i in range(0,int(ancho/2)-1):
175
176         if(imagen_final[int(alto/2)-1,int(ancho/2)-1-i , 2] == 255 and cont_izq == 0 and
177             imagen_final[int(alto/2)-1,int(ancho/2)-1-i, 0:1] == 0): #este contador es
178             para que detecte una sola linea
179             dist_izq = i
180             cont_izq += 1
181
182         if(imagen_final[int(alto/2)-1,int(ancho/2)-1+i , 2] == 255 and cont_der == 0 and
183             imagen_final[int(alto/2)-1,int(ancho/2)-1+i, 0:1] == 0):
184             dist_der = i
185             cont_der += 1
186
187         if(i < int(alto/2) and imagen_final[int(alto/2)-1-i,int(ancho/2)-1 , 2] == 255
188             and cont_adelante == 0 and imagen_final[int(alto/2)-1-i,int(ancho/2)-1, 0:1]
189             == 0):
190             dist_adelante = i
191             cont_adelante += 1
192
193
194     #con el for, recorro la mitad de la imagen en todo el ancho, y busco aquellos
195     #puntos que esten en rojo, esto significa que son los bordes, de ahí sacar la
196     #distancia
197     #similar para la distancia hacia adelante

```

```

187     cv2.line(imagen_final, (int(ancho/2),int(alto/2)-20),(int(ancho/2),int(alto/2)+20)
188         ,(0,255,0),thickness=5) #dibuja la linea verde centrada en el origen
189     cv2.line(imagen_final,(int(ancho/2),int(alto/2)),(int(ancho/2)-dist_izq,int(alto
190         /2)),(0,255,255),thickness=2) #dibuja una linea hasta el borde izquierdo
191     cv2.line(imagen_final,(int(ancho/2),int(alto/2)),(int(ancho/2)+dist_der,int(alto
192         /2)),(255,0,0),thickness=2) #dibuja una linea hasta el borde derecho
193
194
195 while(cap.isOpened()):
196     ret, frame = cap.read()
197     if ret==True:
198
199         captura()
200         curvas_90()
201         seguidor_lineas()
202
203         #print("dist_izq:",dist_izq)
204         #print("dist_der:",dist_der)
205         #print("dist_adelante:",dist_adelante)
206         #print("vel_izq:",vel_izq)
207         #print("vel_der:",vel_der)
208         #print("thresh:")
209         #print("\n")
210         tecla = cv2.waitKey(1)
211
212         if tecla == ord('q'): #terminar de ejecutar
213             motores(0,0)
214             break
215
216         elif tecla == ord('p'): #si se presiona la letra 'p' en el teclado, se pone en
217             pausa o reanuda
218             pausa = not pausa
219             print("pausa:",pausa)
220
221         elif(pausa):
222             motores(0,0)
223             cv2.putText(imagen_final,'Pausa',position3,font,fontSize,fontColor,thick)
224             cv2.putText(imagen_final,'D_der:' +str(dist_der),position5,font,fontSize,
225                 fontColor,thick)
226             cv2.putText(imagen_final,'D_izq:' +str(dist_izq),position6,font,fontSize,
227                 fontColor,thick)
228             else:
229                 cv2.putText(imagen_final,'Avanzar',position3,font,fontSize,fontColor,thick)
230                 cv2.putText(imagen_final,'Der:' +str(vel_der),position2,font,fontSize,
231                     fontColor,thick)
232                 cv2.putText(imagen_final,'Izq:' +str(vel_izq),position,font,fontSize,fontColor
233                     ,thick)
234                 cv2.putText(imagen_final,estado,position4,font,fontSize,fontColor,thick)
235                 cv2.putText(imagen_final,'D_der:' +str(dist_der),position5,font,fontSize,
236                     fontColor,thick)
237                 cv2.putText(imagen_final,'D_izq:' +str(dist_izq),position6,font,fontSize,
238                     fontColor,thick)
239                 motores(vel_izq,vel_der)
240                 #out.write(imagen_final)

```

```

230     #cv2.imshow('Bordes y perspectiva', cv2.hconcat([imagen_final, img_perspectiva
231         ])) #muestra las imagenes juntas
232     cv2.imshow('Bordes', imagen_final) #muestra las imagenes juntas
233 else:
234     break
235
236 #Libera todo si la tarea ha terminado
237 cap.release()
238 #out.release()
239 cv2.destroyAllWindows()
240 pwm_a.stop()
241 pwm_b.stop()
242 GPIO.cleanup()

```

A.2. Detector de intersecciones

```

1 import cv2
2 import numpy as np
3
4 cap = cv2.VideoCapture(0)
5 cap.set(cv2.CAP_PROP_BUFFERSIZE, 1)
6 cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
7 cap.set(cv2.CAP_PROP_FRAME_HEIGHT,480)
8
9 upper_green = np.array([80,255,255],np.uint8)
10 lower_green = np.array([45,80,5],np.uint8)
11
12 mask_izq = np.zeros((480,640), np.uint8)
13 mask_der = np.zeros((480,640), np.uint8)
14 mask_izq[:,0:319] = 255
15 mask_der[:,320:639] = 255
16
17 while(cap.isOpened()):
18
19     # Tomar de a un frame
20     ret, frame = cap.read()
21
22
23     # Convertir BGR en HSV
24     hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
25
26     # "umbralizar" la imagen en hsv para que quede solamente el verde
27     mask2 = cv2.inRange(hsv,lower_green,upper_green)
28
29     # operacion AND bit a bit con la mascara y el frame capturado
30     green = cv2.bitwise_and(frame,frame,mask = mask2)
31
32     #separar la imagen en dos
33     verde_izq = cv2.bitwise_and(green,green,mask = mask_izq)
34     verde_der = cv2.bitwise_and(green,green,mask = mask_der)
35
36     #convertir ambas mitades a escala de gris
37     verde_izq_gray = cv2.cvtColor(verde_izq, cv2.COLOR_BGR2GRAY)

```

```

38     verde_der_gray = cv2.cvtColor(verde_der, cv2.COLOR_BGR2GRAY)
39
40     #Para la imagen izquierda, se obtienen los contornos y luego se extrae el area mas
41     #grande
41     contornos_izq, _ = cv2.findContours(verde_izq_gray, cv2.RETR_EXTERNAL, cv2.
42                                         CHAIN_APPROX_SIMPLE)
42
43     area_izq = 0
44     area_der = 0
45     for c in contornos_izq:
46         area = cv2.contourArea(c)
47         if(area > area_izq): #sobreescribe el area con el mayor valor que va encontrando
48             area_izq = area
49
50     print("area mayor_izq",area_izq)
51
52     #Para la imagen derecha, se obtienen los contornos y luego se extrae el area mas
53     #grande
53     contornos_der, _ = cv2.findContours(verde_der_gray, cv2.RETR_EXTERNAL, cv2.
54                                         CHAIN_APPROX_SIMPLE)
54
55     for c in contornos_der:
56         area = cv2.contourArea(c)
57         if(area > area_der): #sobreescribe el area con el mayor valor que va encontrando
58             area_der = area
59
60     print("area mayor_der",area_der)
61
62     cv2.imshow('frame',frame)
63     cv2.imshow('Verde',green)
64     cv2.imshow('Verde_izq',verde_izq)
65     cv2.imshow('Verde_der',verde_der)
66
67     k = cv2.waitKey(1)
68     # si pulsa q se rompe el ciclo
69     if k == ord("q"):
70         break
71
72     #libera todo
73     cap.release()
74     cv2.destroyAllWindows()

```

Referencias

- [1] RoboCupJunior Rescue Line - Rules 2022
https://junior.robocup.org/wp-content/uploads/2022Rules/2022_RescueLine_Rules_draft01.pdf
- [2] Raspberry Pi 3B
<https://raspberrypi.cl/raspberry-pi-3b/>
- [3] Raspberry Pi Camera V2
<https://export.farnell.com/raspberry-pi/rpi-8mp-camera-board/raspberry-pi-camera-board-v2/dp/2510728?CMP=e14c-noscript&COM=e14c-noscript>
- [4] Driver de motor TB6612FNG
<https://www.luisllamas.es/arduino-motor-dc-tb6612fng/>
<https://www.sparkfun.com/datasheets/Robotics/TB6612FNG.pdf>
- [5] Fuente Step-Down LM2596 DC-DC 1,25-30V 3A
<https://www.unibot.com.ar/productos/fuente-step-down-lm2596-dc-dc-125-30v-3a/>
- [6] Batería Lipo
https://www.rc-flash.com.ar/MLA-623310634-bateria-litio-polimero-lipo-111v-2250mah-35c-helicoptero-kds-450_-JM
- [7] Robotgroup Argentina - página de Facebook
<https://www.facebook.com/robotgroup>
- [8] OpenCV
<https://opencv.org/>
- [9] Raspberry Pi y el PWM
<https://www.prometec.net/raspberry-y-pwm/>
- [10] encontrar-bordes 0.0.13
<https://pypi.org/project/encontrar-bordes/>
- [11] NumPy
<https://numpy.org/>
- [12] Proyecto en GitHub
https://github.com/Ezequiel2003/DSPII-TrabajoFinal_materia