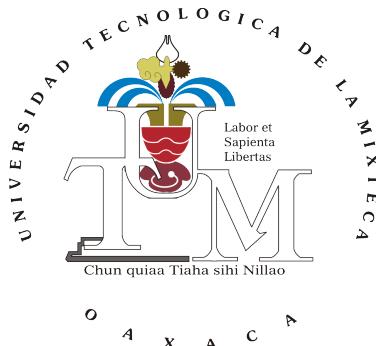


UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA
DIVISIÓN DE ESTUDIOS DE POSGRADO
MAESTRÍA EN ROBÓTICA



“DISEÑO Y CONSTRUCCIÓN DE UNA PLATAFORMA MÓVIL”

REPORTE

QUE SE PRESENTA PARA LA MATERIA DE ROBÓTICA II

PRESENTAN:

**BERITH ATEMOZTLI DE LA CRUZ SÁNCHEZ
BELÉN AIDEE SANTIAGO MARCIAL
YUBESNY IRIANI VENTURA SIGÜENZA**

PROFESOR:

DR. ALBERTO ELÍAS PETRILLI BARCELÓ

HUAJUAPAN DE LEÓN, OAXACA, MÉXICO, 04 DE JULIO DEL 2018

©Derechos Reservados: De La Cruz Sánchez, Santiago Marcial, Ventura Siguenza, 2017

Índice

| | |
|---|-----------|
| 1. Materiales y componentes | 6 |
| 2. Manufactura y ensamble | 8 |
| 2.1. Diseño CAD | 8 |
| 2.2. Manufactura de las Bases de Acrílico | 10 |
| 2.3. Impresión 3D de componentes | 11 |
| 3. Sistema electrónico | 14 |
| 4. Segmentación | 16 |
| 4.1. <i>PythonTM</i> | 16 |
| 4.2. Descripción de cada una de las etapas | 17 |
| 4.2.1. Función main | 17 |
| 4.2.2. Función tomar_foto | 18 |
| 4.2.3. Función scroll_bar | 19 |
| 4.2.4. Función seg_foto | 19 |
| 4.2.5. Función centro | 19 |
| 4.2.6. Función etiquetado | 20 |
| 5. Navegación por campos potenciales | 20 |
| 6. Desarrollo de software para la comunicación. | 26 |
| 7. Resultados | 30 |
| 8. Conclusiones | 40 |
| A. Anexos | 41 |
| A.1. Programa para segmentación y navegación usando campos potenciales: | 41 |

Índice de figuras

| | | |
|-----|--|----|
| 1. | Primer nivel de la plataforma móvil. | 8 |
| 2. | Segundo nivel de la plataforma móvil. | 9 |
| 3. | Tercer nivel de la plataforma móvil. | 9 |
| 4. | Vista isométrica de la plataforma móvil. | 10 |
| 5. | Vista isométrica de la plataforma frontal. | 10 |
| 6. | Corte del acrílico. | 11 |
| 7. | Impresora 3D marca TEVO, modelo Tarantula. | 12 |
| 8. | Ventana principal del software Repetier. | 12 |
| 9. | Proceso de impresión 3D de una pieza. | 13 |
| 10. | Ensamble de la plataforma móvil. | 14 |
| 11. | Conexión de la terminal Bluetooth y la tarjeta arduino. | 15 |
| 12. | Diagrama esquemático de las conexiones realizadas entre la tarjeta Arduino, controlador y encoder del motor. | 15 |
| 13. | Código en Python para la generación del campo potencial concerniente a la meta. | 21 |
| 14. | Código en Python para la generación del campo potencial concerniente a los obstáculos. | 22 |
| 15. | Campo potencial de la meta | 22 |
| 16. | Campo potencial de la meta | 23 |
| 17. | Campo potencial de la meta | 24 |
| 18. | Campo potencial de la meta | 25 |
| 19. | Ensamble de la plataforma móvil. | 26 |
| 20. | Ensamble de la plataforma móvil. | 26 |
| 21. | Ensamble de la plataforma móvil. | 27 |
| 22. | Ensamble de la plataforma móvil. | 28 |
| 23. | Imagen de la pista sin segmentar, con el robot en la posición inicial. | 30 |
| 24. | Barra para seleccionar el color RGB a segmentar. | 31 |
| 25. | Barra con el color verde en RGB seleccionado para los obstáculos. | 31 |
| 26. | Imagen con los obstáculos segmentados en el espacio HVS. | 32 |
| 27. | Imagen con los obstáculos segmentados. | 32 |

| | | |
|-----|---|----|
| 28. | Imagen con las etiquetas de los obstáculos. | 33 |
| 29. | Imagen con los centros de los obstáculos. | 33 |
| 30. | Barra con el color rojo en RGB seleccionado para la meta. | 34 |
| 31. | Imagen con la meta segmentada en el espacio HVS. | 34 |
| 32. | Imagen con la meta segmentada. | 35 |
| 33. | Imagen con el centro de la meta. | 35 |
| 34. | Barra con el color azul en RGB seleccionado para el robot. | 36 |
| 35. | Imagen con el robot segmentado en el espacio HVS. | 36 |
| 36. | Imagen con el centro del robot. | 37 |
| 37. | Barra con el color rosa en RGB seleccionado para la marca de orientación. | 37 |
| 38. | Imagen con la marca de orientación en el espacio HVS. | 38 |
| 39. | Imagen con el centro de la marca de orientación. | 38 |
| 40. | Imagen de la pista cuando el robot ha llegado ala meta. | 39 |

Índice de cuadros

| | | |
|----|--|----|
| 1. | Características de la impresora. | 11 |
|----|--|----|

1. Materiales y componentes

Los materiales que se utilizaron para la construcción de la plataforma móvil, son lo que se presentan a continuación:

- Placa de acrílico de 450x150X3 mm.
- 4 Tornillos de 3/16 in 14 cm.
- 22 Tuercas de 3/16 de in.
- 8 Tornillos de 3/32 in 1 cm.
- 11 Tornillos de 3/32 in 2 cm.
- 19 Tuercas de 3/32 in.
- 2 Ruedas locas de metal 1.5 cm.
- 4 Tornillos de 1/8 in.
- 4 Tuercas de 1/8 in.
- 2 Bases para Micro motorreductores tipo pololu.
- 1 Base para arduino UNO.
- 1 Base para regulador de voltage.
- 2 Ruedas de 5 cm de diámetro.
- 2 Ruedas omnidireccionales.

Los componentes electrónicos que se utilizaron son los siguientes:

- 1 Tira de pines macho.
- 2 Tiras de pines hembra.
- 1 Placa fenólica de 6x4 cm.
- 1 Placa de arduino UNO.
- 1 Módulo regulador de voltaje JE LM2596.
- 2 Micro motorreductores tipo pololu con eje extendido.
- 1 Driver TB6612FNG para micro motorreductores.

- 1 m de cable UTP.
- 1 Par de cables con caimanes.
- 18 cables tipo dupont.
- 1 Pila KDLIPO de 11.1V a 1500 mAh. ll

2. Manufactura y ensamble

2.1. Diseño CAD

Para comenzar con el desarrollo del proyecto fue necesario realizar el diseño en CAD para la manufactura de algunos de los componentes, además de facilitar la colocación de cada una de las piezas y componentes de la plataforma móvil en una ubicación que permita una conexión adecuada entre todos los componentes. El diseño del robot está constituido de tres niveles de áreas pequeñas, de tal manera que permita obtener un diseño estético y funcional. En el primer nivel de la plataforma como se muestra en la Figura 1 se colocan los motores con cada uno de sus sujetadores, así como las ruedas guías para la plataforma, por la parte superior se coloca el módulo bluetooth HC-06 de arduino y driver del motor dual del motor TB6612FNG. Esto permitirá que las conexiones con el encoder de los motores sean más cortas y pueda evitarse que los cables se enreden o generen confusión en las conexiones.

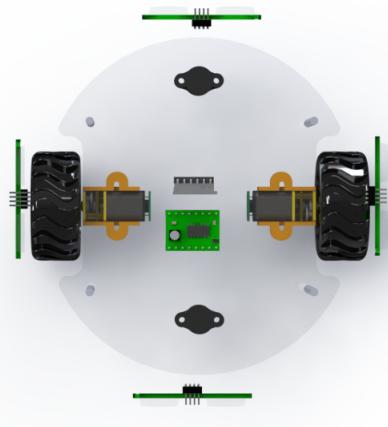


Figura 1: Primer nivel de la plataforma móvil.

En el nivel dos de la plataforma se colocó la tarjeta de desarrollo arduino y la tarjeta Raspberry 3, las cuales se encargan del procesamiento de las señales, así como de las señales de activación y generación de protocolos de comunicación para los distintos componentes, en la Figura 2 se muestra el segundo nivel de la plataforma simulada en Solidworks.

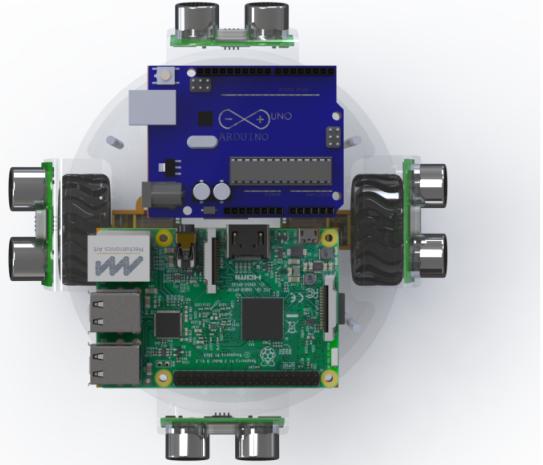


Figura 2: Segundo nivel de la plataforma móvil.

En la Figura 3 se muestra el tercer nivel de la plataforma, la cual se encarga de generar la energía para toda la plataforma, la cual consiste de una batería tipo Lipo y de un regulador de voltaje para la alimentación de los distintos componentes del sistema.

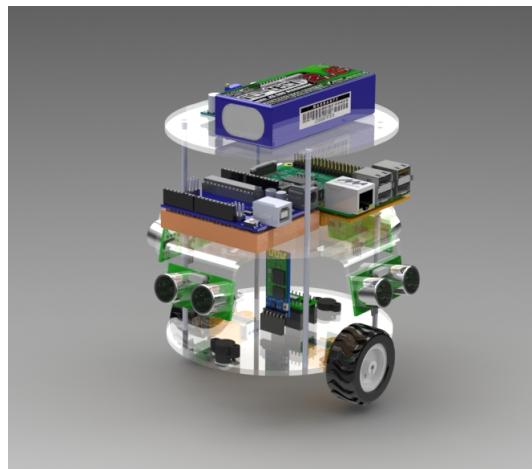


Figura 3: Tercer nivel de la plataforma móvil.

El diseño en CAD de la plataforma móvil con la incorporación de las tarjetas y componentes para el funcionamiento del sistema, se muestran una vista isométrica en la Figura 4, donde es posible apreciar los tres niveles de la plataforma, así como las tarjetas que forman parte del sistema. En la Figura 5 se muestra una vista frontal de la plataforma.

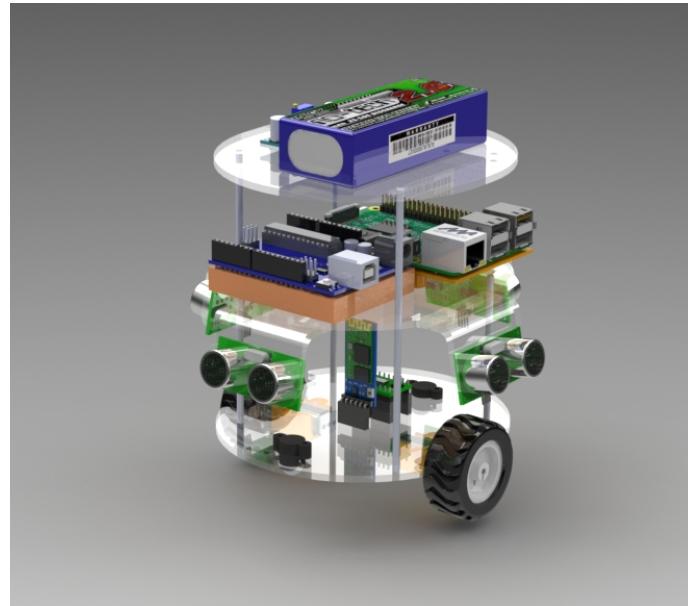


Figura 4: Vista isométrica de la plataforma móvil.

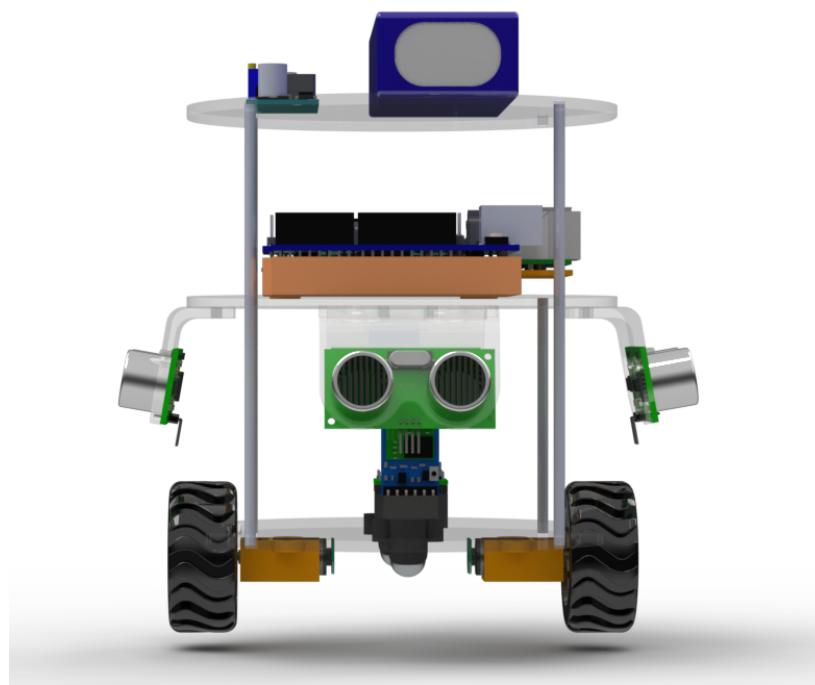


Figura 5: Vista frontal de la plataforma móvil.

2.2. Manufactura de las Bases de Acrílico

Después de realizar el diseño CAD de las bases principales de la plataforma móvil, se procedió a realizar el corte de ellas como se muestra en la figura 6 en el taller de Plásticos de la UTM.



Figura 6: Corte del acrílico.

2.3. Impresión 3D de componentes

Algunas de las piezas pertenecientes a las bases de algunos componentes, se manufacturaron con impresión 3D por deposición fundida, utilizando termoplástico PLA, en una impresora de la marca TEVO, modelo Tarantula, la cual se muestra en la Figura 7, las características de la impresora utilizada se muestran en la Tabla 1 y se encuentran disponibles en [1].

Cuadro 1: Características de la impresora.

| Parámetro | Valor |
|---------------------------|-----------------------|
| Tecnología de Impresión | Deposición fundida |
| Filamento | .75mm |
| Volumen de trabajo | 200mm X 200mm X 200mm |
| Resolución de impresión | 1.75mm |
| Velocidad de impresión | 150 mm/s |
| Resolución de capa | 50 micras |
| Temperatura de extrusor | 210°C-260°C |
| Materiales | PLA, ABS, PETG |
| Tipo de archivo soportado | STL,G-Code |

El procedimiento llevado a cabo para la impresión de las piezas es el siguiente:

- Generación del modelo 3D.

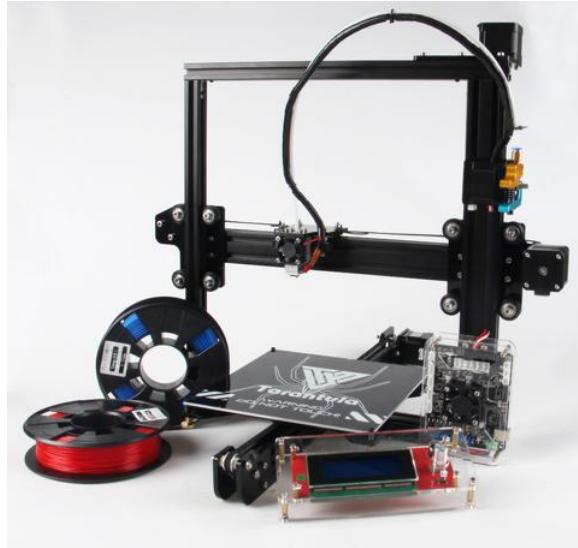


Figura 7: Impresora 3D marca TEVO, modelo Tarantula.

- Preparación de la cama de impresión.
- Preparación del archivo digital.
- Impresión de la pieza.

Para realizar la impresión 3D, el procedimiento consistió en realizar una copia de cada una de las piezas que conforman el exoesqueleto que se diseñaron en Solidworks, se convierte cada uno de los archivos formato STL (STereo Lithography), el cual es un formato de fichero que la gran mayoría de programas de control de impresoras 3D aceptan. Después de tener los archivos, se usa el programa Cura ®, que se encarga de controlar, calibrar la impresora 3D y transmitir los datos de un archivo GCode para ser finalmente impresos, en la Figura 8 se muestra la pantalla principal del software con una pieza a imprimir.

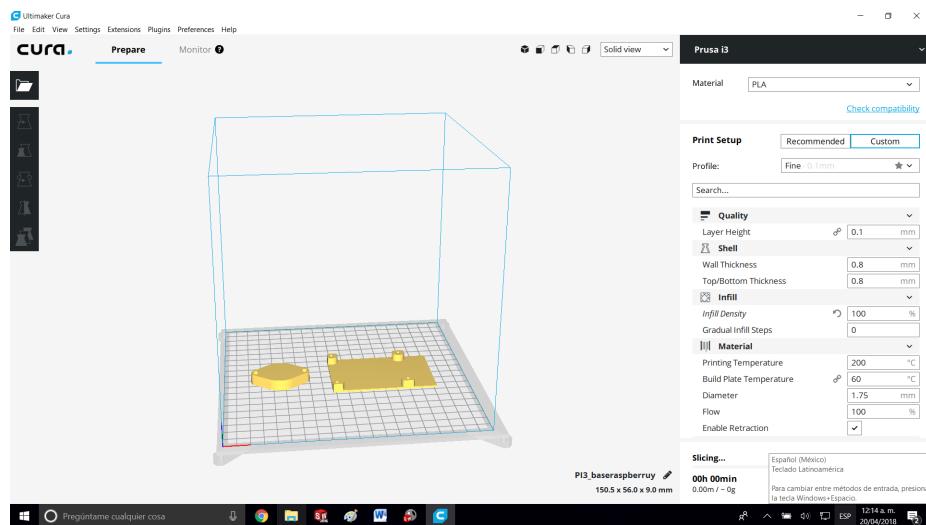


Figura 8: Ventana principal del software Cura.

Una vez generado el archivo en extensión STL, se envía al programa de impresión y comienza la construcción de la pieza, como se observa en la Figura 9.

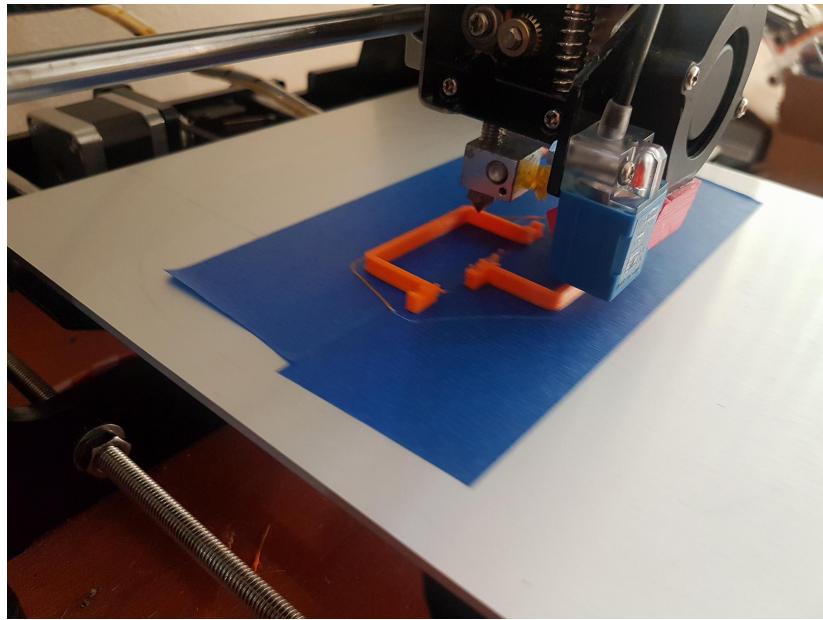


Figura 9: Proceso de impresión 3D de una pieza.

Aunque la impresión por FDM fue diseñada para producir piezas de alta calidad, es necesario hacer un proceso de post-impresión, ya que algunas capas pueden ser visibles, alterando su estética y funcionamiento. Ya que la pieza se encuentra impresa, se procede a lijarla para eliminar rebabas y los restos de material no deseado que puedan haberse depositado en la pieza, obteniendo acabados superficiales que no sean ásperos o rugosos.

Posteriormente de imprimir los componentes y de manufaturar las bases de los niveles de la plataforma, se ensambló el sistema como se muestra en la Figura 10, donde es posible observar el ensamble del segundo nivel de la plataforma, donde las conexiones con los cables entre los distintos componentes jugaron un papel muy importante.

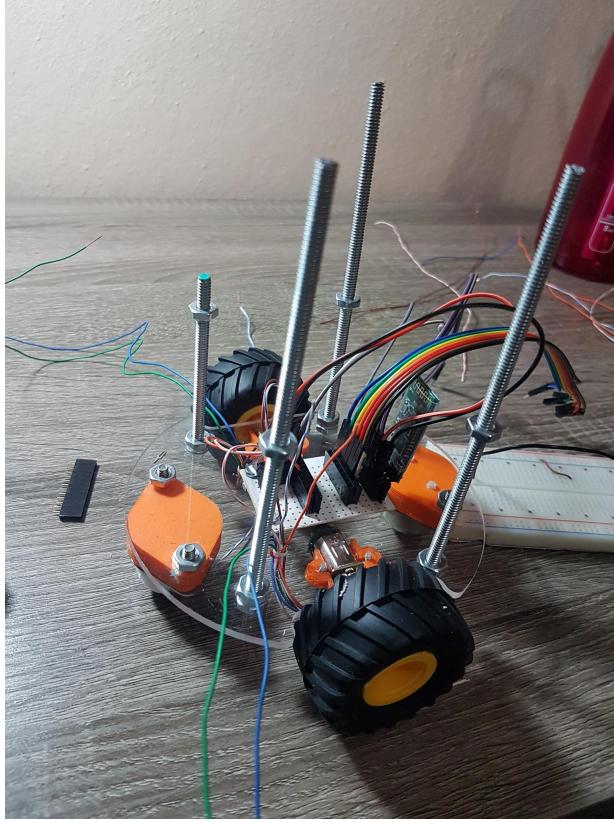


Figura 10: Ensamble de la plataforma móvil.

3. Sistema electrónico

Para el control remoto del robot móvil, se estableció la comunicación inalámbrica vía bluetooth, haciendo uso de la terminal Bluetooth HC-05.

En la imagen 11 se muestra la conexión establecida para la terminal bluetooth al módulo arduino, así como su respectiva alimentación, la cual proviene de un regulador de voltaje modelo LM2596 DC-DC, que entrega 5V de salida al tener una entrada de 11.1V proveniente de la batería de LiPo. El módulo cuenta con puertos TxD y RxD el cual permitirá realizar la comunicación inalámbrica a una distancia máxima de 10 metros, enviando los comandos al microcontrolador por un puerto serial.

Para cada uno de los motores del robot, es necesario un motorreductor de CD como sistema de actuación. Cada motor cuenta con un encoder magnético de la empresa Pololu y el cual se conecta de acuerdo con la hoja de datos reportada en [2], Para el control bidireccional de los motores se ocupará una tarjeta de la marca Polulu con un controlador Toshiba TB6612FNG, el cual permite el control de dos motores mediante la generación de una señal PWM (modulación de ancho de pulso, por sus siglas en inglés), las conexiones con el motor y el microcontrolador, se realizaron de acuerdo con las especificaciones de la hoja de datos [3]. El microcontrolador que se ocupó para la generación de las señales de PWM y para la adquisición de datos provenientes de las señales de los encoders, es una tarjeta de desarrollo Arduino UNO, la cual cuenta con pines especiales para generación de señales PWM, lectura de señales analógicas y para comunicación serial, SPI y TWI, [4].

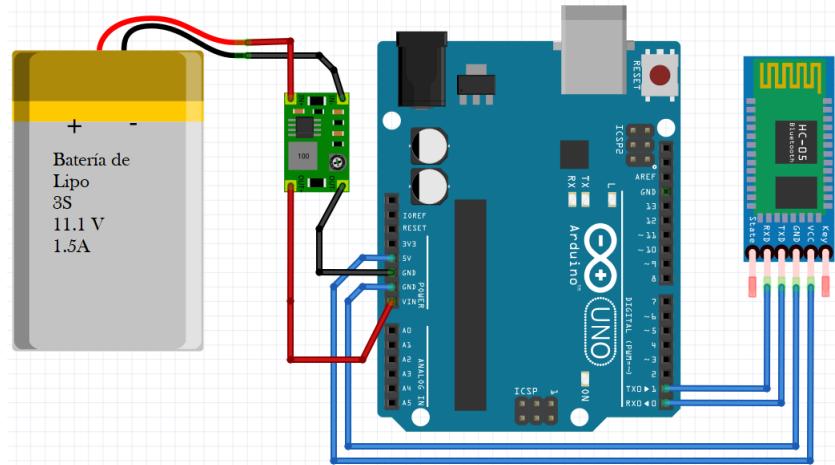


Figura 11: Conexión de la terminal Bluetooth y la tarjeta arduino.

En la Figura 12 se muestra el diagrama de conexiones de la tarjeta de desarrollo Arduino con el driver, encoder y los motores. La alimentación de los motores se realizó con un voltaje de 12V, para la alimentación del driver y del encoder se alimenta con una salida directa de la tarjeta Arduino, la cual genera 5V. Cada uno de los encoders cuenta con dos pines los cuales se conectan a pines directos del microcontrolador, con los cuales se puede obtener las mediciones necesarias. Para el driver es necesaria la generación de dos señales de PWM para obtener la señal de alimentación de cada uno de los motores.

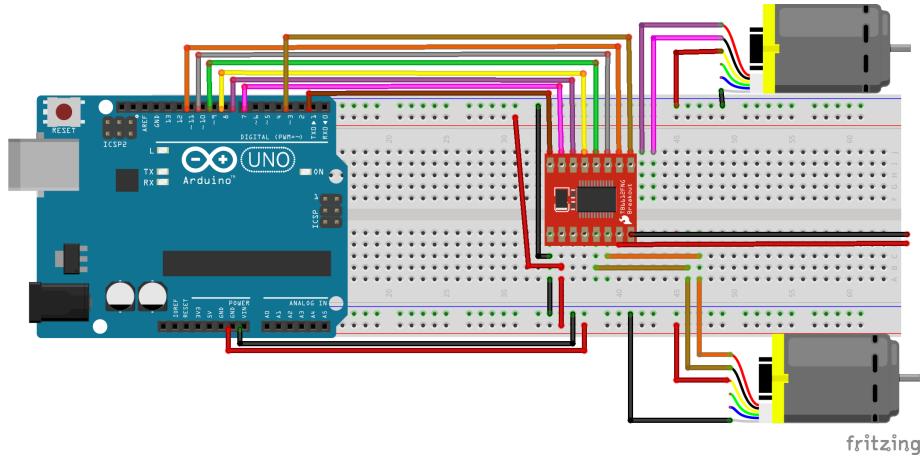


Figura 12: Diagrama esquemático de las conexiones realizadas entre la tarjeta Arduino, controlador y encoder del motor.

4. Segmentación

El programa para realizar la segmentación por color de los obstáculos, la meta, robot y la marca de orientación, se desarrolló en el lenguaje de programación orientado a objetos *PythonTM*. El programa principal consta de 1 función principal (main) y 6 funciones secundarias:

- tomar_foto
- scroll_bar
- seg_foto
- centro
- etiquetado
- keep_plots_open

La lógica del programa es la siguiente: toma una imagen (foto) de 640x480 píxeles con una cámara aérea con acceso remoto vía IP, en la cual se segmentan los obstáculos (verde) y la meta (rojo), los cuales son objetos estáticos, para luego obtener sus respectivos centros, para luego generas sus campos potenciales; repulsivo para los obstáculos y atractivo para la meta. Posteriormente en esa misma imagen se segmenta en color del robot (azul) y de la marca de orientación (rosa). Después en la imagen en tiempo real (video) de 640x480 píxeles con la misma cámara, se obtienen el centro del robot y de la marca de orientación, en cada instante de tiempo, y respecto al centro del robot se genera su campo potencial repulsivo. Y se envía vía bluetooth los datos de información para que el robot sepa hacia donde debe girar.

4.1. *PythonTM*

Python es un lenguaje de programación poderoso y fácil de aprender. Cuenta con estructuras de datos eficientes y de alto nivel y un enfoque simple pero efectivo a la programación orientada a objetos. La elegante sintaxis de Python y su tipado dinámico, junto con su naturaleza interpretada, hacen de éste un lenguaje ideal para scripting y desarrollo rápido de aplicaciones en diversas áreas y sobre la mayoría de las plataformas [5].

El intérprete de Python y la extensa biblioteca estándar están a libre disposición en forma binaria y de código fuente para las principales plataformas desde el sitio web de Python, <https://www.python.org/>, y puede distribuirse libremente. El mismo sitio contiene también distribuciones y enlaces de muchos módulos libres de Python de terceros, programas y herramientas, y documentación adicional [5].

4.2. Descripción de cada una de las etapas

En las primeras líneas del programa presentado en la sección de anexos, se agregan todas las librerías necesarias para realizar la segmentación de cada uno de los colores de la pista, comunicación bluetooth con el robot y la generación de los campos potenciales.

4.2.1. Función main

Se realiza la comunicación con el bluetooth del arduino, se define el puerto y la velocidad de transmisión de datos. En seguida se define el nombre de la imagen (foto) que tomará la cámara, luego se manda a llamar a la función "*tomar_foto*" a la cual se le manda el nombre de la imagen y regresa la imagen. Con la imagen que retorna la función *tomar_foto*, se obtienen los colores en el espacio HSV.

Se define el nombre "*Segmentacion – 1*" que llevará la imagen segmentada de los obstáculos y se manda a llamar a la función *scroll_bar*, y a la cual se le envía la imagen en el espacio de color HSV, la imagen original (foto) y el nombre que llevará la imagen segmentada, y recibe la imagen segmentada y los niveles HSV inferior y superior. A continuación se muestra la imagen segmentada con el nombre "*Obstculos*". Se define el nombre de la imagen para los centros de los obstáculos "*CentrosObstaculos*" y se manda a llamar a la función "*centro*", a la cual se le envía el nombre que llevará la imagen con los centros y la imagen segmentada y recibe dos vectores, uno con las coordenadas *x* y otro con las coordenadas *y* de los obstáculos. Después se define el nombre de la imagen para las etiquetas de los obstáculos "*EtiquetasObstaculos*" y se manda llamar a la función "*etiquetado*", la cual recibe el nombre de la imagen para las etiquetas de los obstáculos y la imagen segmentada, y regresa el ancho de los obstáculos. Luego se destruye la ventana de la imagen con el nombre *Segmentacion – 1*.

A continuación se crean los campos potenciales para los obstáculos, como se describe en la siguiente sección.

Se define el nombre "*Segmentacion – 2*" que llevará la imagen segmentada de la meta y se manda a llamar nuevamente a la función *scroll_bar*. A continuación se muestra la imagen segmentada con el nombre "*Meta*". Se define el nombre de la imagen para el centro de la meta "*CentrosMeta*" y se manda a llamar nuevamente a la función "*centro*" y recibe dos vectores, uno con la coordenada *x* y otro con la coordenada *y* de la meta. Después se define el nombre de la imagen para la etiqueta de la meta "*EtiquetaMeta*" y se manda llamar nuevamente a la función "*etiquetado*" y regresa el ancho de la meta. Luego se destruye la ventana de la imagen con el nombre *Segmentacion – 2*.

Se define el nombre "*Segmentacion – 3*" que llevará la imagen segmentada del robot y se manda a llamar nuevamente a la función *scroll_bar*. Para obtener los niveles de color inferior y superior en el espacio HSV y utilizarlos posteriormente para ubicar el centro del robot en la imagen en tiempo real (video). Luego se destruye la ventana de la imagen con el nombre *Segmentacion – 3*.

Se define el nombre "*Segmentacion – 4*" que llevará la imagen segmentada de la marca de orienta-

ción del robot y se manda a llamar nuevamente a la función *scroll_bar*. Para luego obtener los niveles de color inferior y superior en el espacio HSV y utilizarlos posteriormente para ubicar el centro de la marca en la imagen en tiempo real (video). Después se destruye la ventana de la imagen con el nombre *Segmentacion – 4*.

Luego se define la dirección IP para el video, se declaran los vectores con los niveles de color superior e inferior en el espacio HSV del robot y de la marca de orientación.

Se definen el nombre de la imagen para el centro del robot como "*Centro del Robot*" y de la imagen, para el centro de la marca de orientación como "*Centro Marca*". Y se normaliza la imagen a coordenadas del plano $x - y$, de tal forma que a todas las coordenadas y utilizadas para la ubicación de los campos potenciales quedan como $y = 480 - y$, pues 480 es el ancho de la imagen.

A continuación se crea el campo potencial para la meta, como se describe en la siguiente sección. Y se inicializa un contador, el cual sirve para dentro del ciclo *while* infinito que se crea , cada que llegue a 100 ciclos o más, tome las muestras del centro del robot y de la marca siempre y cuando los vectores que devuelva la función centro para estos dos objetos sea de tamaño 1, tanto para la coordenada x e y .

Dentro de ese mismo ciclo se toma la imagen del video y se obtienen sus colores en el espacio HSV. Se crea la máscara de color con los niveles bajos y altos tanto para el robot, como para la marca, se filtra el ruido aplicando una operación morfológica OPEN-CLOSE (erosión-dilatación) y se aplica una operación AND con la imagen original, para así obtener la imagen solo con el color entre los niveles HSV, para el robot y la marca respectivamente. Y se manda a llamar por tercera vez a la función "*centro*" enviándole el nombre para la imagen del centro del robot y recibe dos vectores, uno con la coordenada x y otro con la coordenada y del robot. Enseguida se manda a llamar por cuarta vez a la función "*centro*" enviándole el nombre para la imagen del centro de la marca y recibe dos vectores, uno con la coordenada x y otro con la coordenada y de la marca. Se crean sus respectivos campos potenciales después de las condiciones antes mencionadas, para que el robot tenga tiempo de llegar a la posición indicada. Cuando se cumplen las condiciones se vuelve a inicializar el contador.

Los ángulos de movimiento deseado son enviados, y si las coordenadas del centro del robot se encuentran dentro del área de la meta, se envía un número '0', para que se detenga el robot. En la figura 40 se muestra la imagen de la pista una vez que el robot ha llegado a la meta.

4.2.2. Función tomar_foto

Se recibe el nombre de la imagen (foto), se define la dirección IP de la cámara, se toma una imagen de la pista y se guarda en la computadora, para después leer y mostrar la imagen como "*FotoPista*" y retornar la imagen a la función principal, como se muestra en la figura .

4.2.3. Función scroll_bar

Recibe la imagen en el espacio de color HSV, la imagen original (foto) y el nombre que llevará la imagen segmentada.

- **Segmentacion-1:** para los obstáculos.
- **Segmentacion-2:** para la meta.
- **Segmentacion-3:** para el robot.
- **Segmentacion-4:** para la marca de orientación.

Se crea la barra de colores con deslizantes en RGB (Rojo, Verde y Azul), para así poner los niveles de colores para cada uno de los objetos, como se muestra en la figura 24. Y una vez establecido el nivel R (Red-Rojo), el nivel G (Green-Verde) y B (Blue-Azul). Luego se cierra la barra de colores y se obtienen los niveles de color H, S y V, tanto superior (hL, sL y vL), como inferior (hU, sU y vU). En las imágenes 25, 30, 34 y 37, se muestran las barras de colores con los niveles correspondientes de RGB, para cada color de los objetos; obstáculos, meta, robot y marca de orientación, respectivamente.

Enseguida de manda a llamar a la función **seg_foto**, a la cual se le envía el espacio de color HSV de la imagen (foto) original, los niveles HSV y la imagen (foto) original, y recibe la imagen segmentada, la cual se muestra con el nombre recibido en la llamada desde la función principal. Como se puede ver en las imágenes 26, 31, 35 y 38.

4.2.4. Función seg_foto

Recibe el espacio de color HSV de la imagen (foto) original, los niveles HSV y la imagen (foto) original, se definen los vectores con los niveles colores a detectar, se crea la mascara de color con los niveles bajos y altos, se filtra el ruido aplicando una operación morfológica OPEN-CLOSE (erosión-dilatación) y se aplica una operación AND con la imagen original, para así obtener la imagen solo con el color entre los los niveles HSV recibido y por último retornar esa imagen.

4.2.5. Función centro

Recibe el nombre que llevará la imagen con los centros y la imagen sementada, de acuerdo a los valores enviados en llamada desde la función principal:

- Centros Obstáculos
- Centros Meta
- Centro del Robot

- Centros de la Marca

Se encuentran los contornos de los objetos en cada imagen recibida, con ayuda de los momentos se obtienen los centros de cada objeto y se dibujan en la imagen, estos coordenadas se guardan en dos vectores, recibe dos vectores, uno con la coordenada x y otro con la coordenada y , muestra la imagen con los centros como se observa en las figuras 29, 33, 36 y 39; centro de los obstáculos, centro de la meta, centro del robot y centro de la marca de orientación, respectivamente. Por último retorna los vecotres a la función principal.

4.2.6. Función etiquetado

Recibe el nombre que llevará la imagen para las etiquetas y la imagen segmentada, se inicializa el valor de la variable w para el ancho, se obtiene el numero de objetos y se etiqueta cada uno de ellos, con el vector obtenido de las etiquetas, se realiza un ciclo for para obtener el ancho, largo, y sus coordenadas en x e y . Luego muestra la imagen con las etiquetas:

- Etiquetas , como se muestra en la figura 28
- Etiqueta Meta

Y por último regresa el ancho del recuadro de la etiqueta, para así crear el campo potencial de acuerdo a este valor.

5. Navegación por campos potenciales

Para el desplazamiento del robot se utilizó el método de campos potenciales artificiales, ya que resulta ser una buena opción para la generación de trayectorias de robots móviles debido a su baja complejidad y su fundamento físico que lo respalda, sin embargo, puede presentar problemas por la presencia de mínimos locales como también la generación de trayectorias no óptimas. Para el desarrollo del proyecto se ha definido directamente un campo de fuerzas con un perfil Gausiano. Con la forma de este campo de fuerza, se dispone de una respuesta que cambia lentamente en posiciones cercanas al objetivo y rápidamente en posiciones lejanas, en la Ecuación 1, se muestra la definición general de una curva Gaussiana.

$$f(x) = a * \exp^{-\frac{(x-b)^2}{2*c^2}} \quad (1)$$

En esta expresión, a es la máxima amplitud de la Gaussiana, b es la posición del pico central, y c es el parámetro que define la anchura de la Gaussiana. Para el desarrollo del proyecto, se generaron tres diferentes campos potenciales, el primer campo potencial corresponde al generado por los obstáculos, el segundo corresponde a la meta y el último campo corresponde al generado por el carro. Para obtener

el comportamiento de los vectores a lo largo y ancho de todo el plano se hace una suma de los campos como se muestra en la ecuación 2.

$$P_{total} = P_{repulsivos} + P_{atractivos} \quad (2)$$

Para generar los campos potenciales es necesario obtener la información de la segmentación realizada con anterioridad, la obtención de los centros de los distintos elementos que se tienen en el plano, permitirá la generación de los campos. Para esto es necesario tener en cuenta y saber la posición de los centros de todos los obstáculos, la meta y el robot. En la Figura 13, se muestra el código utilizado en Python para la generación del campo potencial para la meta, la estructura del código es similar para la generación del campo con respecto al carro, pero con diferentes valores.

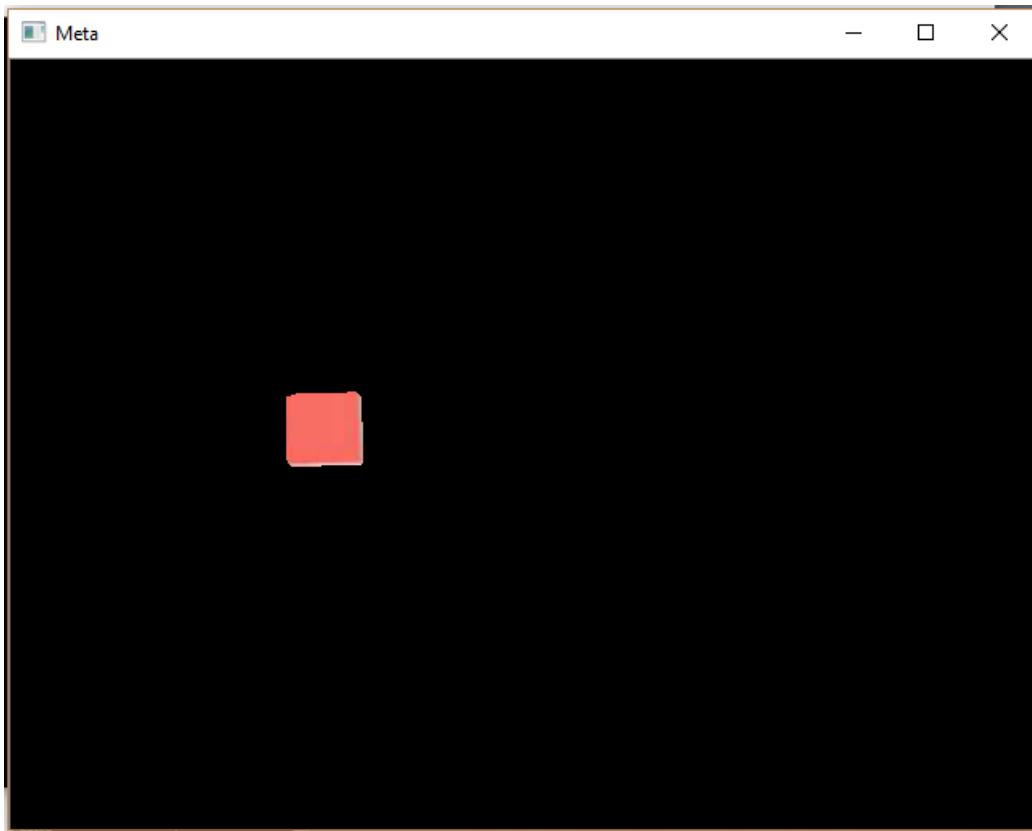


Figura 13: Código en Python para la generación del campo potencial concerniente a la meta.

Para la generación del campo potencial de los obstáculos es necesario realizarlo con un ciclo que permita calcular todos los campos con respecto a todos los obstáculos, en la Figura 14 se muestra el código utilizado para esto. Para la generación del gradiente se usa una función predeterminada de Python que se llama `gradient()`.

Para la generación de los campos como se trabaja en un plano cartesiano es necesario tener en cuenta que para poder realizar cualquier cálculo con los valores de los pixeles provenientes de la imagen, el origen de la imagen se tiene que ajustar, en un principio el origen se encuentra situado en la parte superior izquierda de la imagen y para reubicar el origen solo es necesario hacer una resta con respecto al eje x, la resta está en función del ancho de la imagen. Las coordenadas del plano

```

-----Campo potencia de los obstaculos
alpha_obstacle, a_obstacle, b_obstacle = 1.0, 1.9e3, 1.9e3
pobs=0
lim=np.size(xCen0)
yACen2=np.arange(lim)
lim2=np.size(yACen2)
for contador in range(lim-1):
    yACen2[contador]=480-yCen0[contador]
    x_obstacle = xCen0[contador-1]
    y_obstacle = yACen2[contador-1]
    p1= -alpha_obstacle * np.exp(-((x - x_obstacle)**2 / a_obstacle + (y - y_obstacle)**2 / b_obstacle))
    pobs=p1+pobs

```

Figura 14: Código en Python para la generación del campo potencial concerniente a los obstáculos.

cartesiano van a variar dependiendo del ancho de la imagen y el video con el que se trabajará durante el movimiento del robot, en este caso, las fotografías y el video fueron tomadas con una resolución de 480x640, esto debe considerarse al inicio de la generación de los campos, para definir el área de trabajo.

Para el cálculo del campo potencial generado por los obstáculos durante la prueba, se muestra en la Figura 15, con valores de $a=1$, $b=1.9e3$, $c=1.9e3$. El resultado puede apreciarse con la dirección de las flechas que los distintos campos son repulsivos.

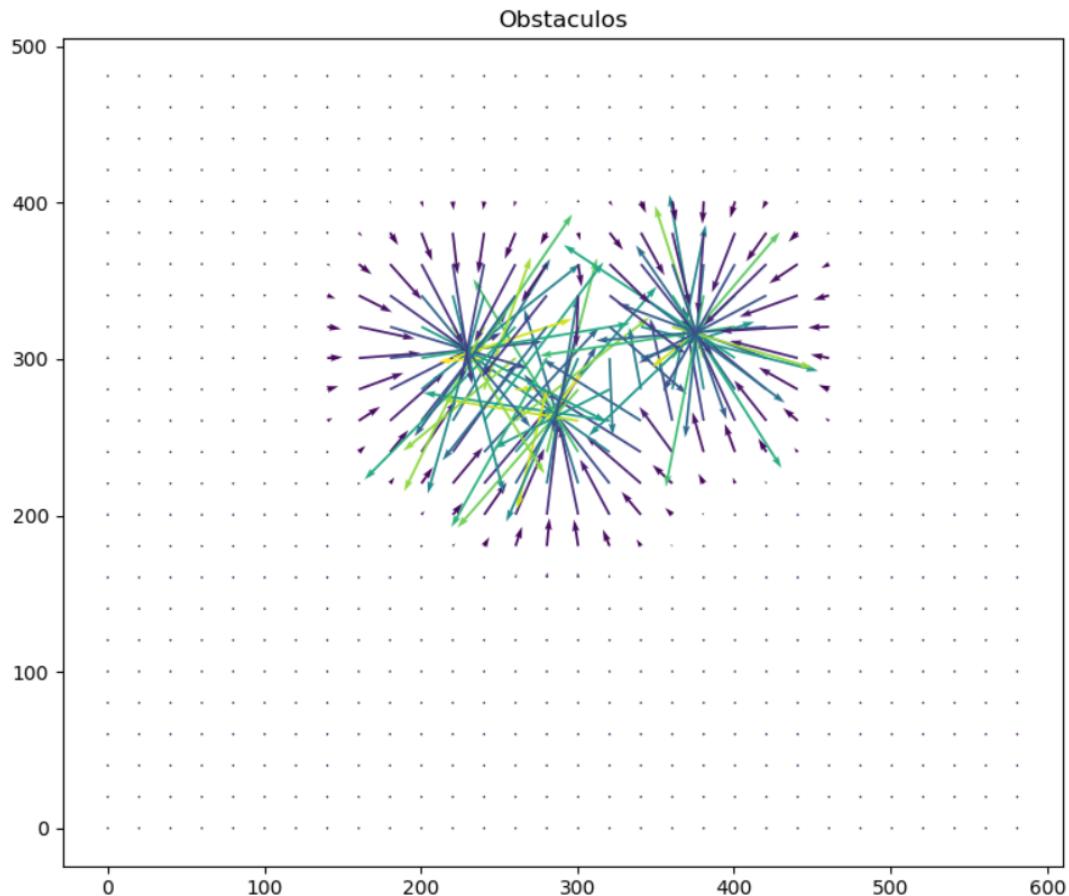


Figura 15: Campo potencial de la meta

Para el cálculo del campo potencial generado por la meta durante la prueba, se muestra en la Figura

16, con valores de $a=1$, $b= 3e4$, $c= 3e4$. El resultado puede apreciarse con la dirección de las flechas que el campo generado es atractivo.

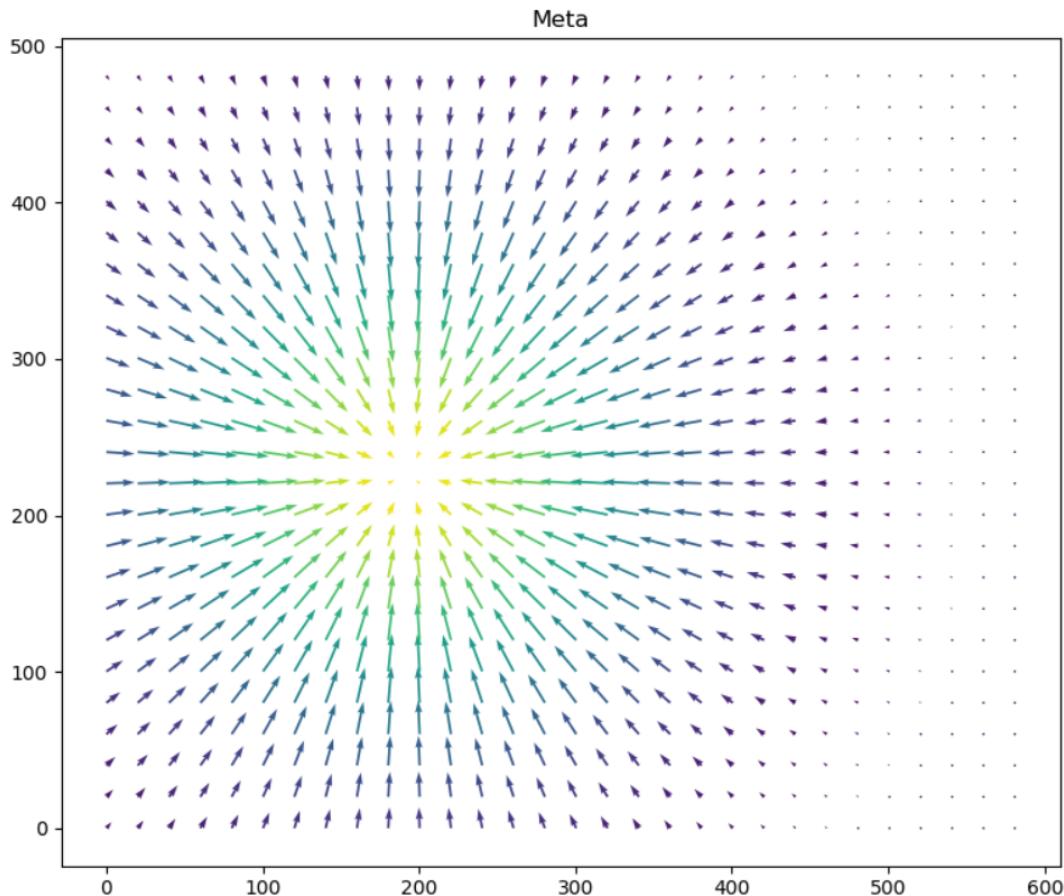


Figura 16: Campo potencial de la meta

Para el cálculo del campo potencial generado por el robot durante la prueba, se muestra en la Figura 17, con valores de $a=1$, $b= 1.9e3$, $c= 1.9e3$. El resultado puede apreciarse con la dirección de las flechas que el campo generado es repulsivo.

Para mostrar los campos potenciales anteriores el algoritmo prácticamente consiste en calcular el efecto que causará un punto el resto del plano, esto se realiza con la Ecuación 2. En la Figura 18 se muestra el resultado final de sumar los campos de la meta, obstáculos y el robot. Los campos potenciales van a ir variando conforme el robo se mueve, ya que mediante el video se obtendrá el centro del robot y será posible determinar los nuevos valores que se tiene que mover.

Para determinar la posición real en la que se encuentra el robot se utiliza el centro del robot y el centro de una marca, color rosado, las cuales sirven para la ubicación espacial del robot, ya que con ello es posible determinar la orientación. Para calcular la orientación del robot se ocupa la tangente inversa que permite obtener el ángulo que se encuentra rotado el robot, con respecto a la horizontal del plano. En la Ecuación 3 se muestra cómo es que puede ser calculado este ángulo, teniendo en cuenta la ubicación del centro del robot y la marca, donde $xcenR1$, $ycenR1$ corresponden a la marca y $xcenR$,

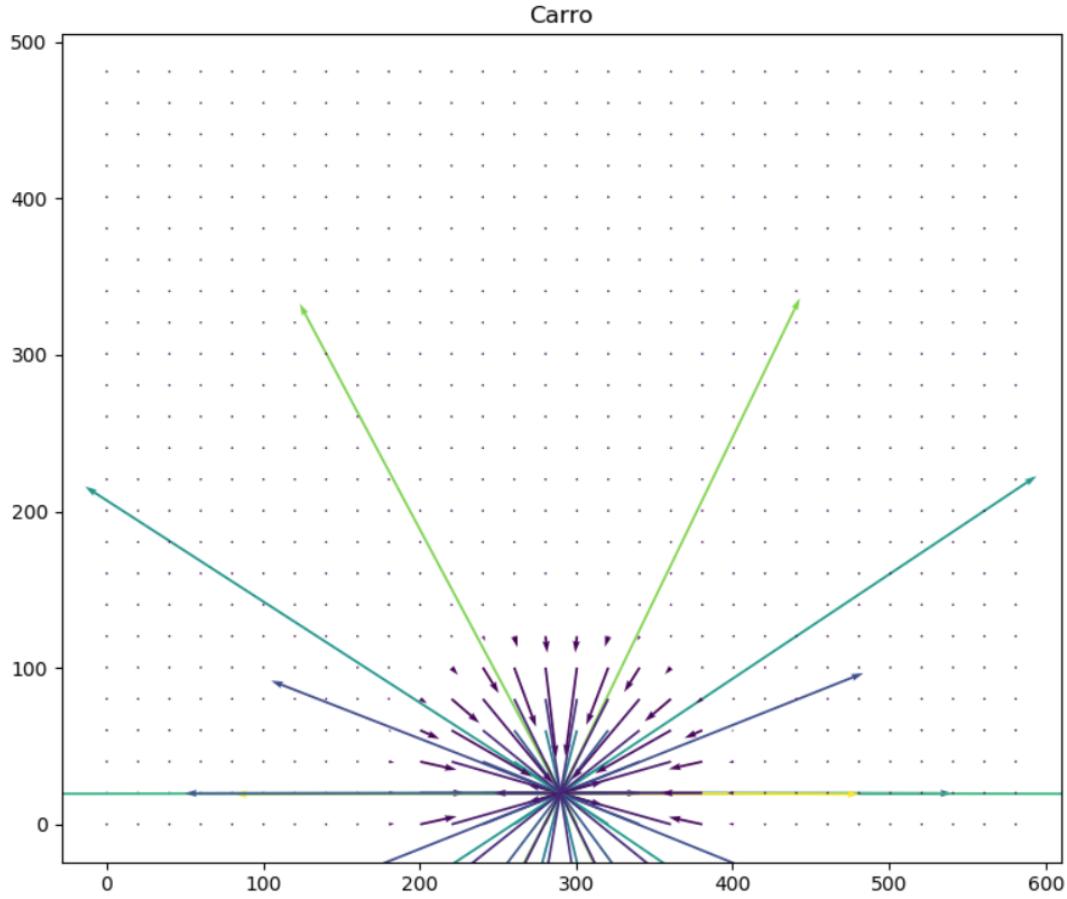


Figura 17: Campo potencial de la meta

y_{cenR} corresponden al centro del robot.

$$Angulo_Anterior = \text{atan}\left(\frac{(480 - y_{cenR1}) - (480 - y_{cenR})}{(x_{cenR1} - x_{cenR})}\right) * \frac{180}{3,1416} \quad (3)$$

robot, se obtiene de los diferenciales de (x , y) obtenidos con la función del gradiente. Estos valores se encuentran contenidos en una matriz, donde cada punto (fila, columna) representa un valor en el plano cartesiano, en la Ecuación 4 se muestra como fue obtenido este ángulo y el cual es nombrado como AnguloDeseado.

$$Angulo_Deseado = \text{atan}\left(\frac{(dy[480 - y_{cenR1}][x_{cenR1}])}{(dx[480 - y_{cenR1}][x_{cenR1}])}\right) * \frac{180}{3,1416} \quad (4)$$

El ángulo real que se debe mover el carro es obtenido mediante la Ecuación 5:

$$Angulo_Mover = Angulo_Deseado - Angulo_Anterior \quad (5)$$

De esta forma permite obtener el ángulo al que debe dirigirse el robot tomando en cuenta el valor real al que se encuentra, teniendo un rango de error menor. Posteriormente de obtener esto, los valores

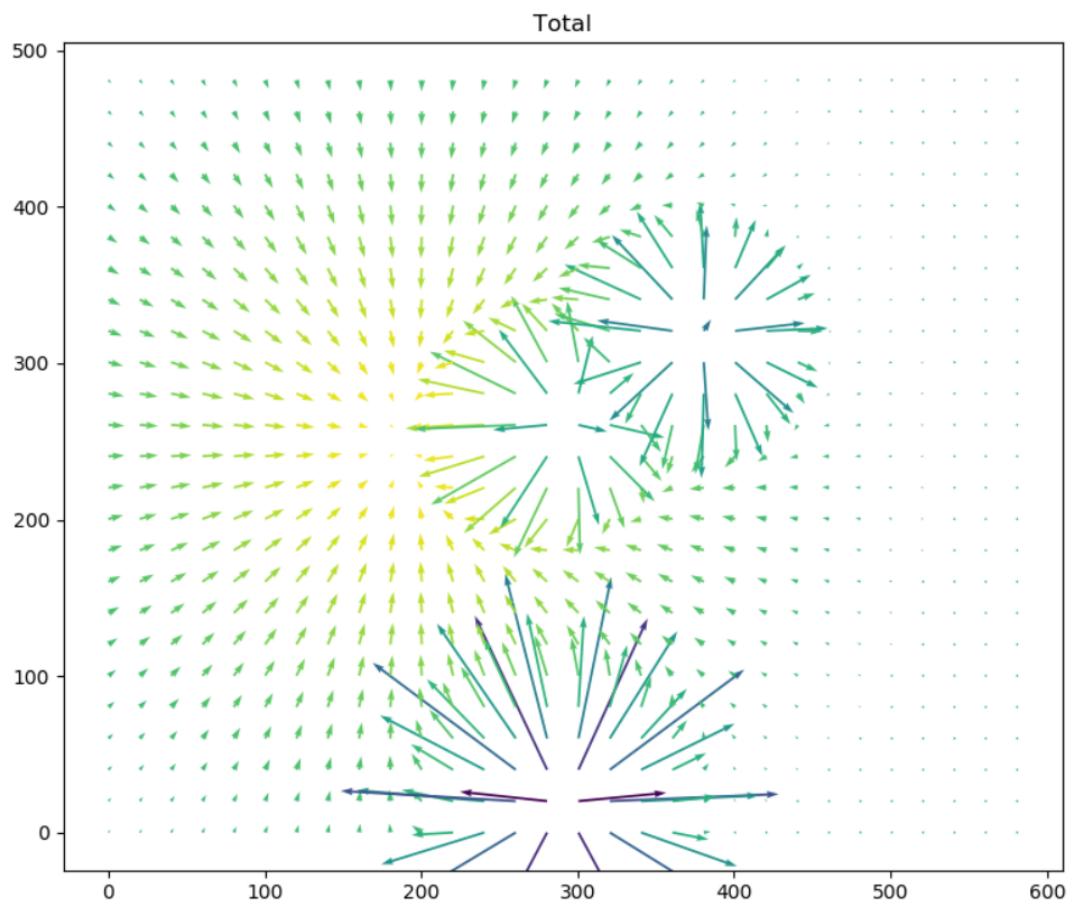


Figura 18: Campo potencial de la meta

son enviados mediante comunicación bluetooth con el robot.

6. Desarrollo de software para la comunicación.

Para la comunicación remota con el carrito, se estableció la conexión vía Bluetooth con ayuda del módulo y la tarjeta Arduino. Al mismo tiempo, debido que se trabajó en el entorno de programación de Python, era necesario establecer la comunicación entre Python y Arduino, en la figura 19 se muestran las etapas implementadas para la comunicación.

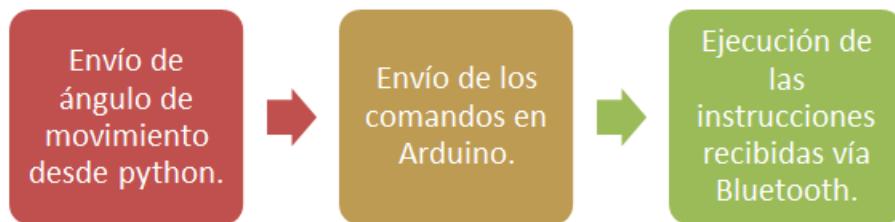


Figura 19: Etapas implementadas para la comunicación entre la plataforma móvil y la planta de trabajo.

Para comprender la manera en la que se realiza la comunicación del carrito con la comunicación a la estación de trabajo, es necesario entender el funcionamiento de la comunicación serial.

Un puerto es el nombre genérico con que se denomina a las interfaces, físicos o virtuales, que permiten la comunicación entre dos ordenadores o dispositivos. Un puerto serie envía la información mediante una secuencia de bits. Para ello se necesitan al menos dos conectores para realizar la comunicación de datos, RX (recepción) y TX (transmisión), veáse la figura 20.

COMUNICACIÓN SERIE

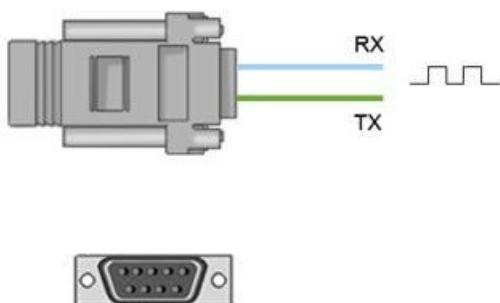


Figura 20: Conductores de comunicación en un puerto serie.

Un ordenador convencional dispone de varios puertos de serie. Los más conocidos son el popular USB (universal serial port) y el ya casi olvidado RS-232 (el de los antiguos ratones). Sin embargo, dentro del ámbito de la informática y automatización existen una gran cantidad adicional de tipos de

puertos serie, como por ejemplo el RS-485, I2C, SPI, Serial Ata, Pcie Express, Ethernet o FireWire, entre otros.

En la literatura, se reportan los puertos de serie como UART (universally asynchronous receiver/transmitter), la cual, es una unidad que incorporan ciertos procesadores, encargada de realizar la conversión de los datos a una secuencia de bits y transmitirlos o recibirlas a una velocidad determinada. Por otro lado, podrá ser encontrado en la literatura, el término TTL (Transistor-Transistor logic), en el cual la comunicación se realiza mediante variaciones en la señal entre 0V y Vcc (donde Vcc suele ser de 3.3V o 5V).

El modelo de Arduino Uno con el que se estará trabajando, dispone de una unidad UART que operan a nivel TTL (0V/5V), haciéndolos compatibles con la conexión USB. En este modelo de Arduino, los pines empleados son 0(RX) y 1(TX), con los cuales se estará trabajando para realizar la transmisión de los datos recibidos a través del módulo bluethoot.

El módulo Bluethoot HC-05 con el que se cuenta, realiza su transmisión de datos por medio de comunicación serial, por lo que será posible que enviar datos desde la pc a la plataforma móvil, ésta comunicación también puede ser alámbrica.

Para controlar Arduino con Python y realizar el envío de la información por medio del módulo Bluetooth, es necesario que se cuente con la librería Pyserial de python.

El sketch implementado se muestran en la figura 21

```
#include <SoftwareSerial.h>
SoftwareSerial BT(0,1);
double option;
//Motor 1
int pinAIN1 = 9; //Dirección
int pinAIN2 = 8; //Dirección
int pinPWMA = 3; //PWM
//Motor 2
int pinBIN1 = 7;
int pinBIN2 = 12;
int pinPWMB = 5;
//Standby
int pinSTBY = 4;

static boolean turnCW = 0; //Para la ft
static boolean turnCCW = 1;
static boolean motor1 = 0;
static boolean motor2 = 1;
long oldPosition = -999;

void setup()
{
    BT.begin(9600);
    pinMode(pinLED, OUTPUT);
    pinMode(pinPWMA, OUTPUT); //Motor 1
    pinMode(pinAIN1, OUTPUT);
    pinMode(pinAIN2, OUTPUT);
    pinMode(pinPWMB, OUTPUT); //Motor 2
    pinMode(pinBIN1, OUTPUT);
    pinMode(pinBIN2, OUTPUT);
}

void loop(){}
    if (BT.available() >0)
    {
        option = BT.read();
        if (option > 0)//Izquierda
        {
            time_=option*(3800)/90;
            digitalWrite(pinLED, HIGH);
            motorDrive(motor1, turnCCW, 23);
            motorDrive(motor2, turnCCW, 10);
            delay(time_);
            digitalWrite(pinSTBY, LOW);
            motorDrive(motor1, turnCCW, 20);
            motorDrive(motor2, turnCW, 18);
            delay(2000);
            digitalWrite(pinSTBY, LOW);
        }
        if (option < 0)//Derecha
        {
            time_=option*(4200)/90;
            digitalWrite(pinLED, LOW);
            motorDrive(motor1, turnCW, 10);
            motorDrive(motor2, turnCW, 20);
            delay(800);
            digitalWrite(pinSTBY, LOW);
            motorDrive(motor1, turnCCW, 20);
            motorDrive(motor2, turnCW, 18);
            delay(2000);
            digitalWrite(pinSTBY, LOW);
        }
        if (option == '0'){ //Detener
            digitalWrite(pinSTBY, LOW);
        }
    }

void motorDrive(boolean motorNumber, boolean motorDirection, int motorSpeed)
{
    boolean pinIn1;
    if (motorDirection == turnCW)
        pinIn1 = HIGH;
    else
        pinIn1 = LOW;

    if(motorNumber == motor1)
    {
        digitalWrite(pinAIN1, pinIn1);
        digitalWrite(pinAIN2, !pinIn1);
        analogWrite(pinPWMA, motorSpeed);
    }
    else
    {
        digitalWrite(pinBIN1, pinIn1);
        digitalWrite(pinBIN2, !pinIn1);
        analogWrite(pinPWMB, motorSpeed);
    }
    digitalWrite(pinSTBY, HIGH);
}
```

Figura 21: Programa implemento en Arduino

Es necesario saber que al enviarse los datos (en caracteres ASCII), se deberá restar el valor '0' al dato recibido para recuperar el valor numérico enviado.

El programa muestra que, al recibir un ángulo mayor que 0 (giro a la izquierda), se accede a una condición en la cual se determina el tiempo de giro de la plataforma para girar el ángulo determinado. Esto se determinó, realizando pruebas de movimiento en la plataforma, y debido a que las llantas

utilizadas se resbalaban haciendo su avance más lento.

Así, se observó que la plataforma invertía 3800ms para lograr girar 90 grados a la izquierda, y 4200ms para girar a la derecha. De esta manera, al enviar el ángulo de giro desde el programa en python, al ser recibidos en arduino, este lo escalaría al tiempo necesario para dar el giro solicitado.

Para realizar pruebas de este programa, se implementó el Script de prueba mostrado en la figura 22:

```
import serial, time
arduino = serial.Serial("COM3", 9600)
time.sleep(1)
arduino.write(b'45')
arduino.write(b'0')
arduino.write(b'-15')
arduino.write(b'0')
arduino.write(b'10')
arduino.write(b'-45')
arduino.write(b'0')
arduino.close()
```

Figura 22: Script de prueba implementado en Python.

En el cual, primero es necesario importar la librería PySerial y es creado un objeto de tipo Serial, en el cual se indican los valores del puerto seria que se esté utilizando, en este caso “COM3”, ya que es el puerto asignado a la conexión Bluethooth, estas configuraciones se pueden ver en “configuraciones avanzadas de Bluethoot”.

Así mismo, es necesario importar la librería time, para poder hacer uso de la función sleep y dar tiempo entre el inicio de la conexión del puerto serie y el envío de datos.

Como se observa, el comando para enviar datos al arduino se realiza mediante arduino.write(b’x’). Esta función envía bytes, por lo que es necesario convertir el valor antecediendo una b al valor enviado.

Finalmente, se cierra el puerto serie con la función close()”.

Los valores reales utilizados para las pruebas finales, fueron enviadas desde el programa final implementado, mostrado en el Anexo A.1 .

Una manera de verificar que la comunicación y el envío de datos se está realizando, es observando el estado del módulo a través de la secuencia de parpadeo del led que tiene integrado, los cuales son los siguientes:

- Estado Desconectado: El módulo entra en este estado tan pronto como sea alimentado y cuando no se ha establecido conexión con algún otro dispositivo. Se observará que el LED del módulo parpadea rápidamente.
- Estado Conectado o de comunicación: Entra en este estado cuando se establece una conexión con otro dispositivo bluethoot (la PC en este caso). El LED realiza un doble parpadeo. Todos

los datos que se ingresen al Hc-05 por el pin RX se transmiten por bluetooth al dispositivo conectado, y los datos recibidos se devuelven por el pin TX.

p

7. Resultados

En esta sección se muestra cada uno de las imágenes obtenidas, mencionadas en la sección de Segmentación.

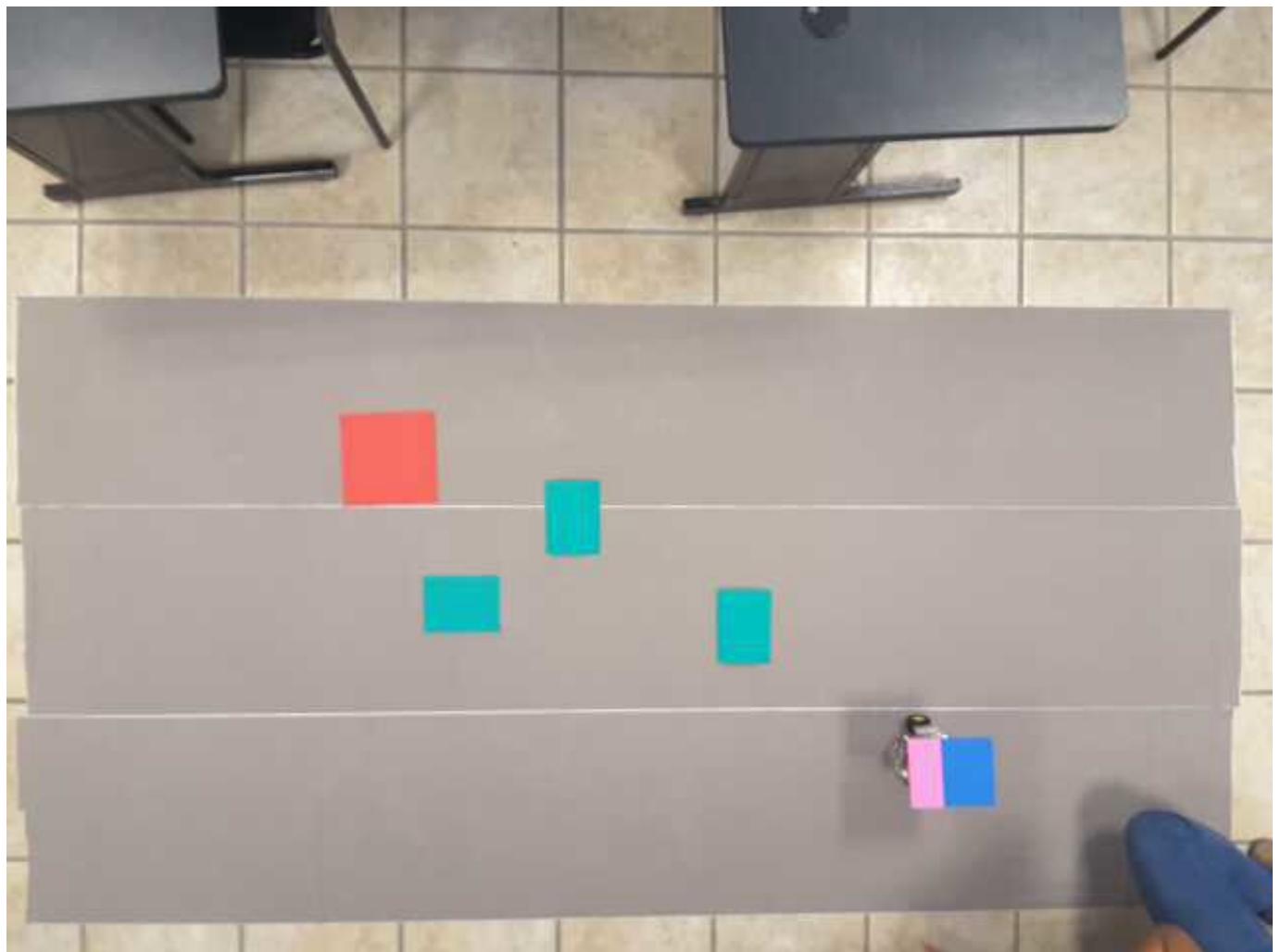


Figura 23: Imagen de la pista sin segmentar, con el robot en la posición inicial.

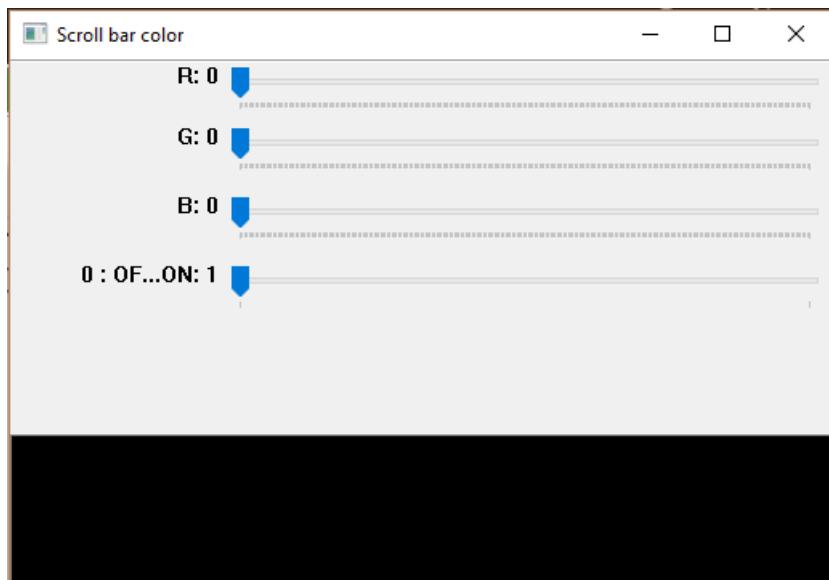


Figura 24: Barra para seleccionar el color RGB a segmentar.

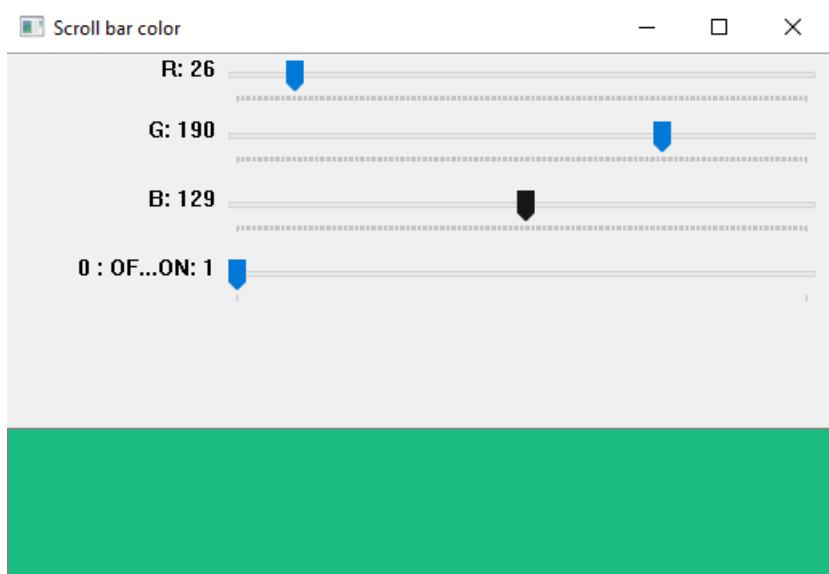


Figura 25: Barra con el color verde en RGB seleccionado para los obtáculos.



Figura 26: Imagen con los obstáculos segmentados en el espacio HVS.

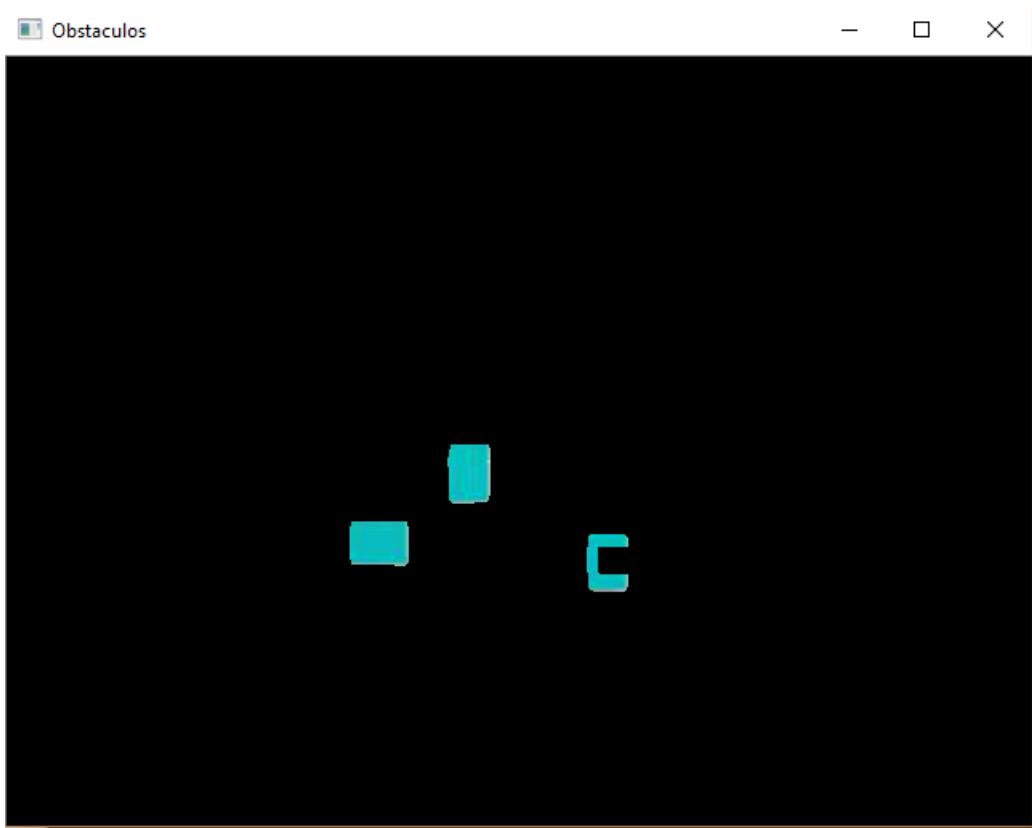


Figura 27: Imagen con los obstáculos segmentados.

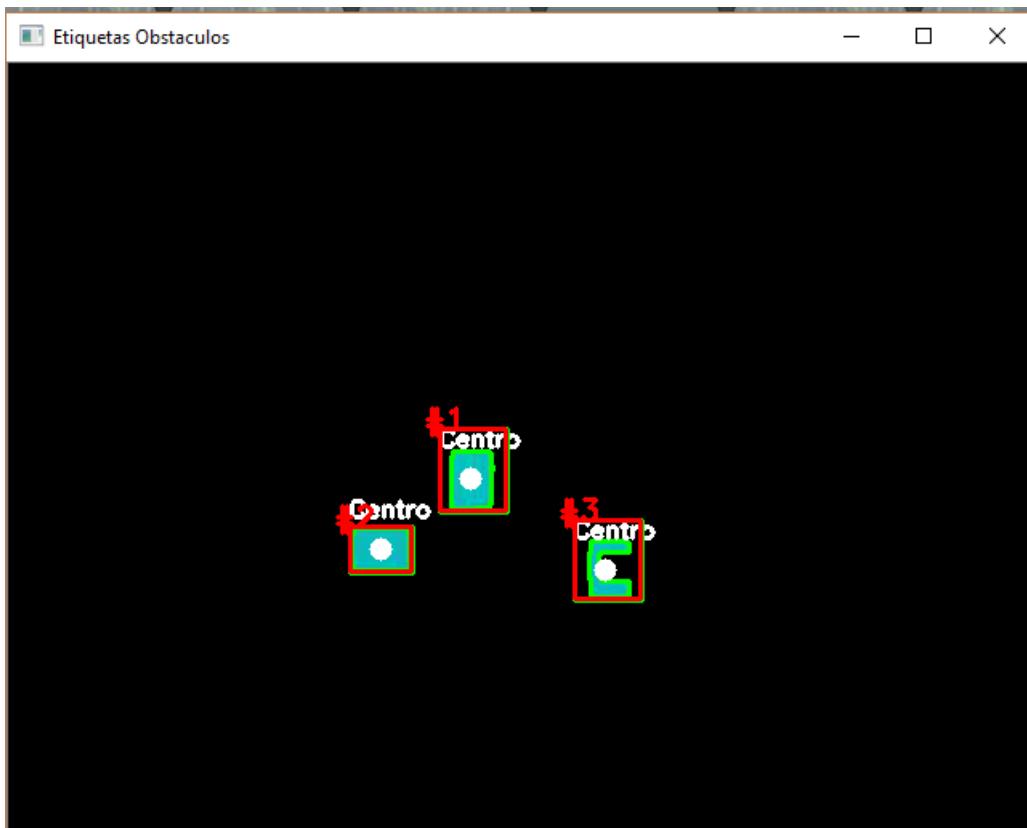


Figura 28: Imagen con las etiquetas de los obstáculos.



Figura 29: Imagen con los centros de los obstáculos.

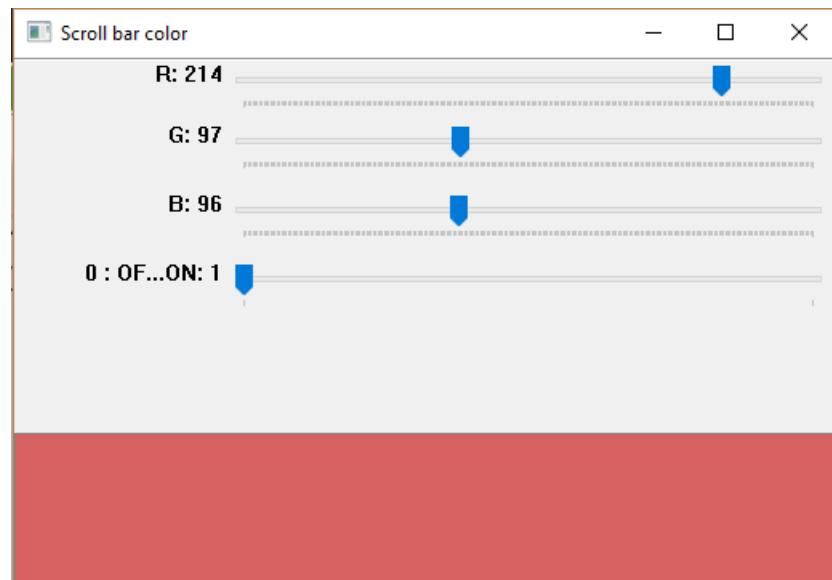


Figura 30: Barra con el color rojo en RGB seleccionado para meta.

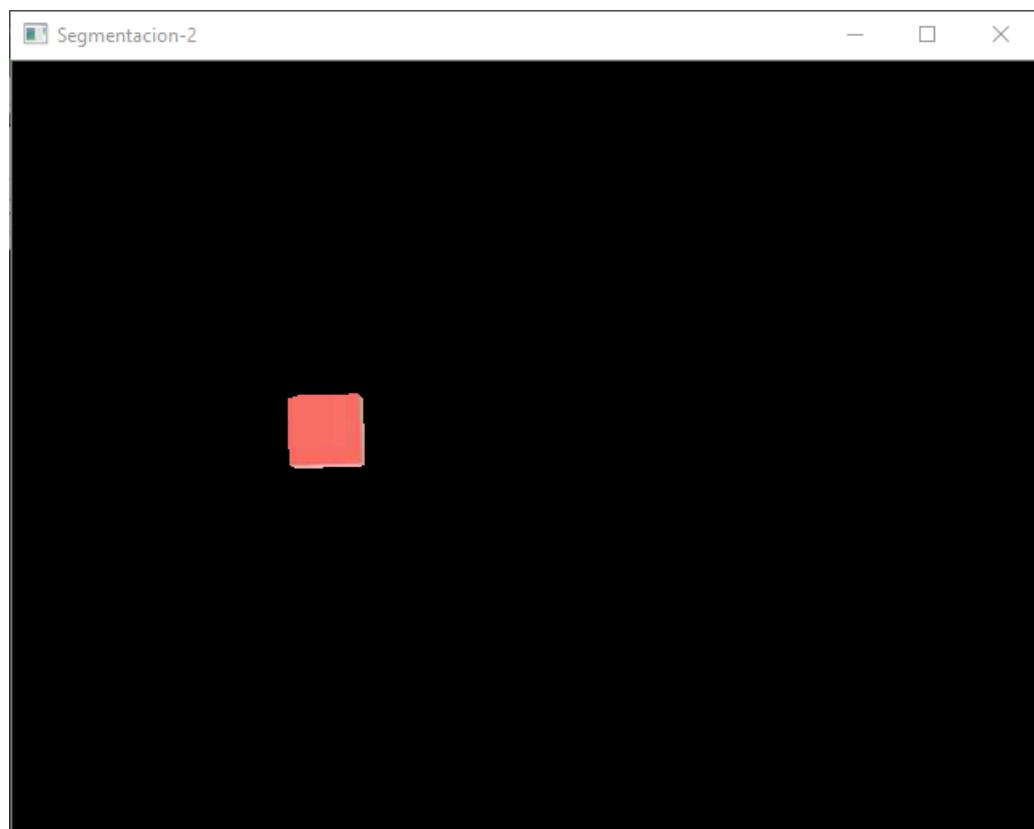


Figura 31: Imagen con la meta segmentada en el espacio HVS.

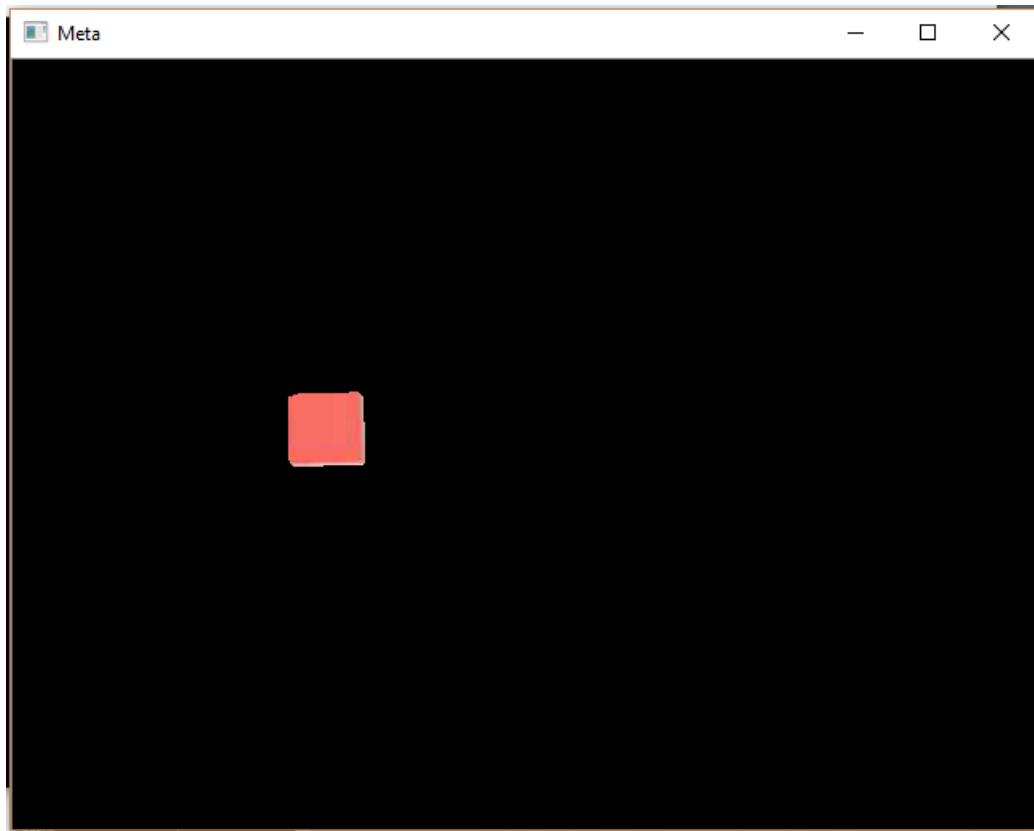


Figura 32: Imagen con la meta segmentada.

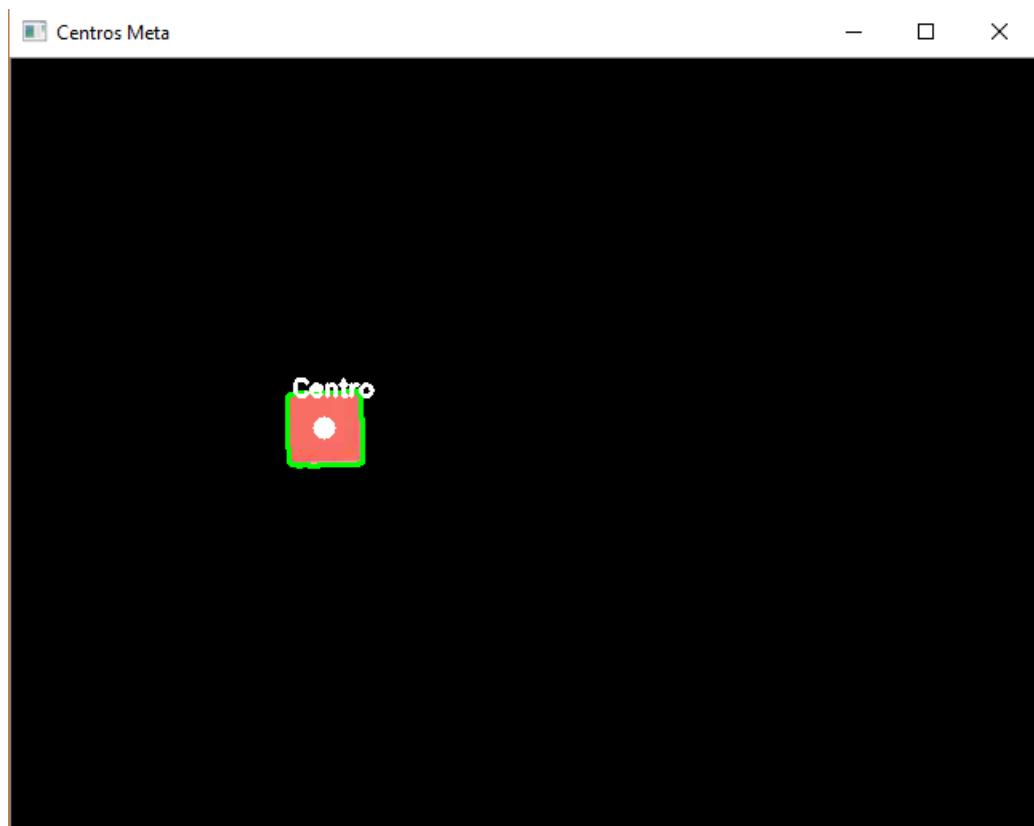


Figura 33: Imagen con el centro de la meta.



Figura 34: Barra con el color azul en RGB seleccionado para el robot.

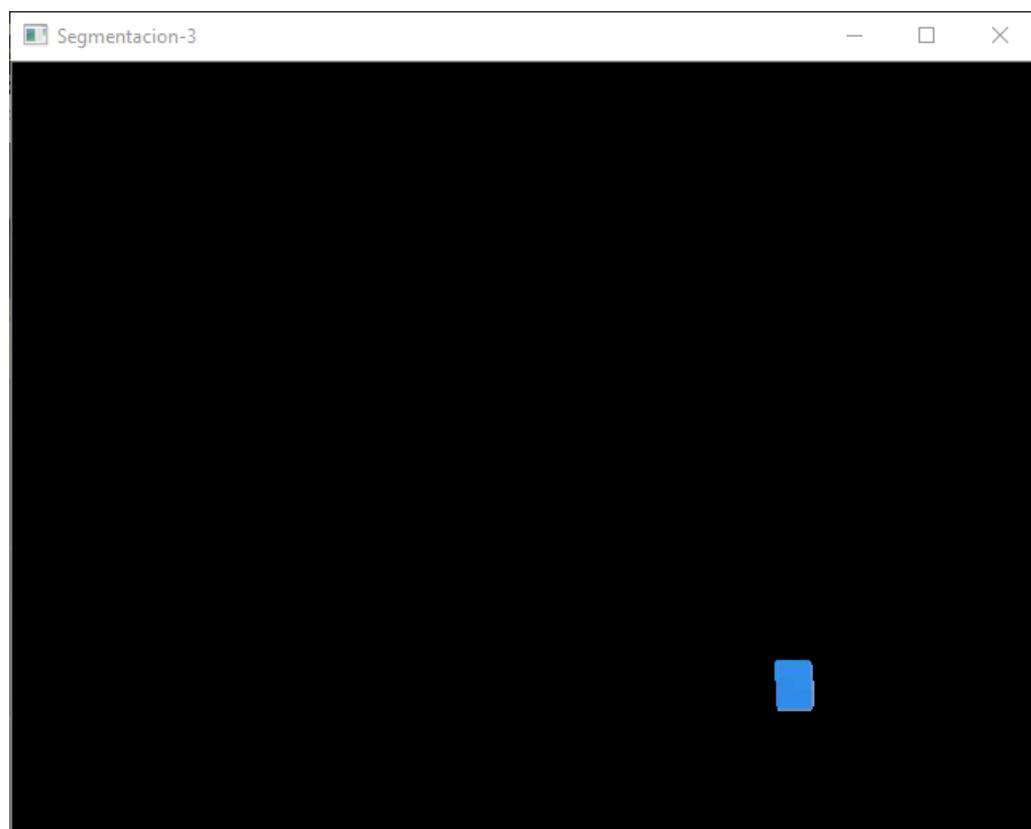


Figura 35: Imagen con el robot segmentado en el espacio HVS.

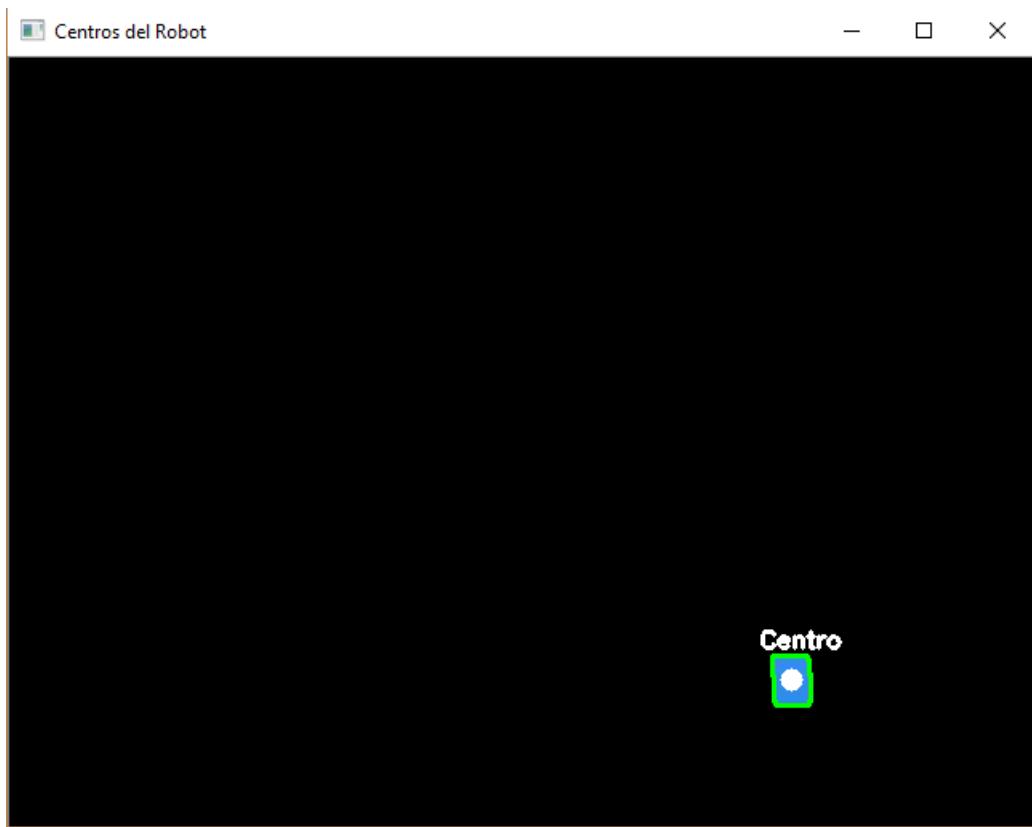


Figura 36: Imagen con el centro del robot.

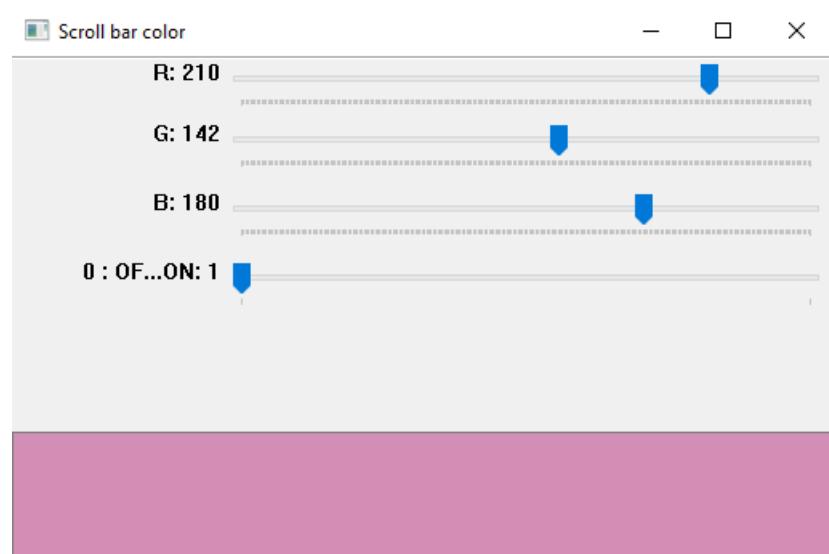


Figura 37: Barra con el color rosa en RGB seleccionado para la marca de orientación.

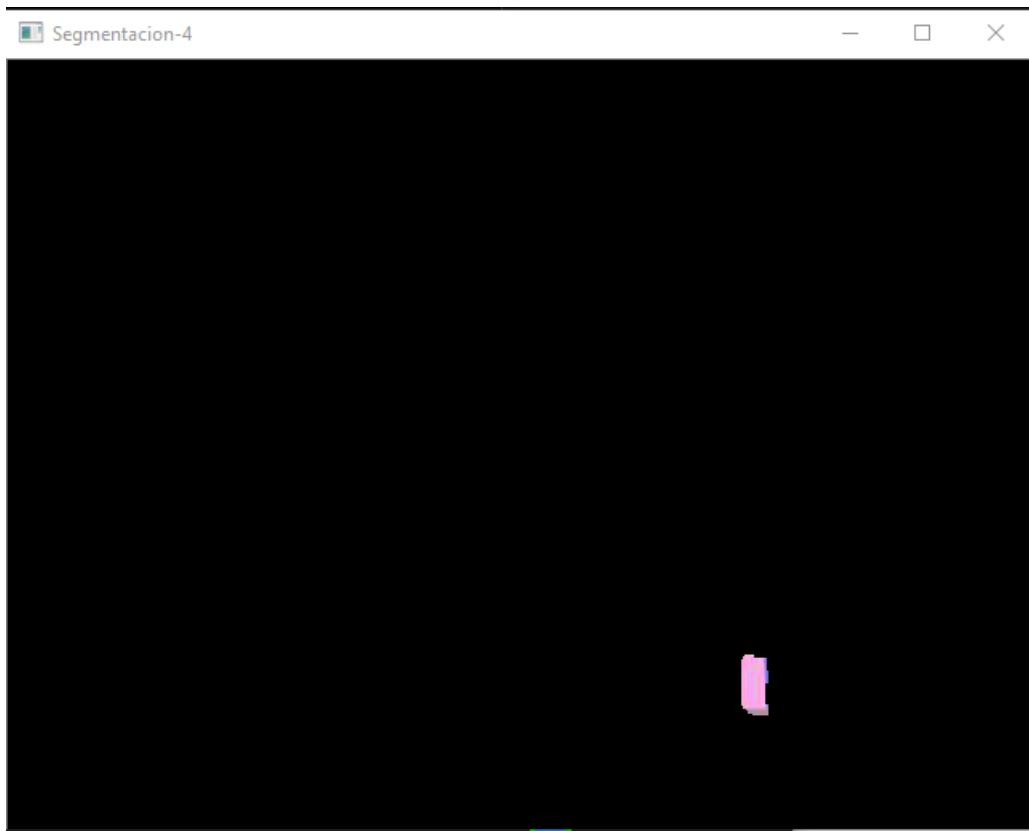


Figura 38: Imagen con la marca de orientación en el espacio HVS.

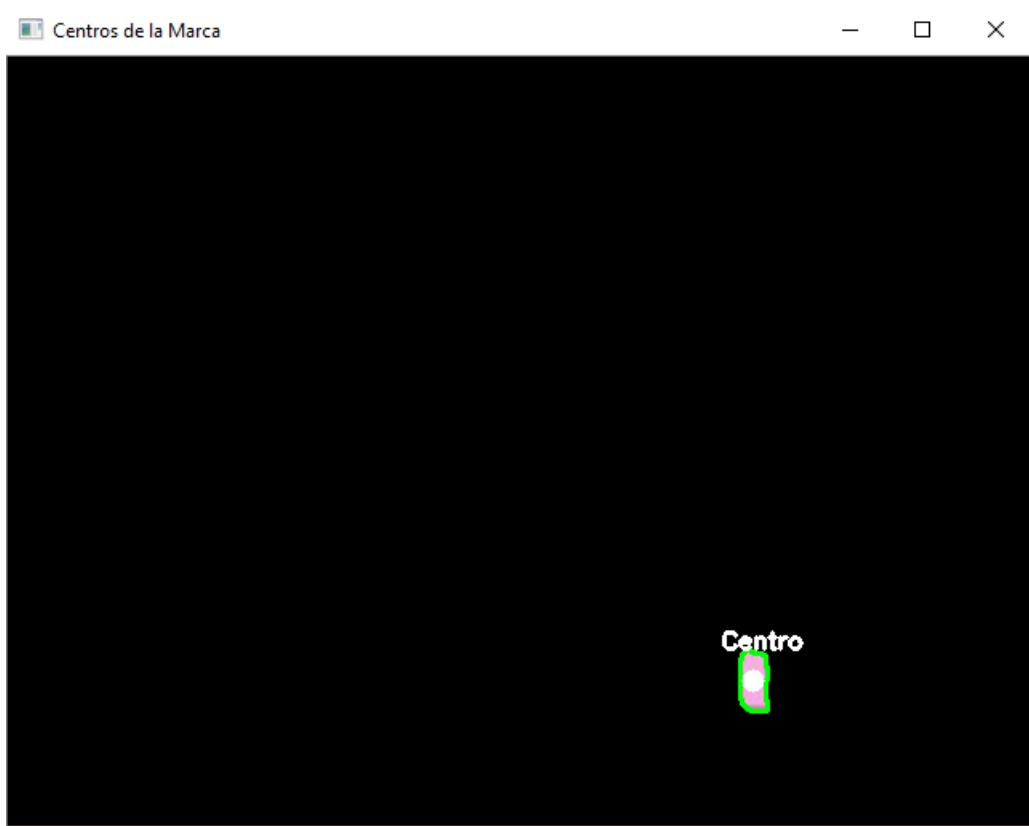


Figura 39: Imagen con el centro de la marca de orientación.

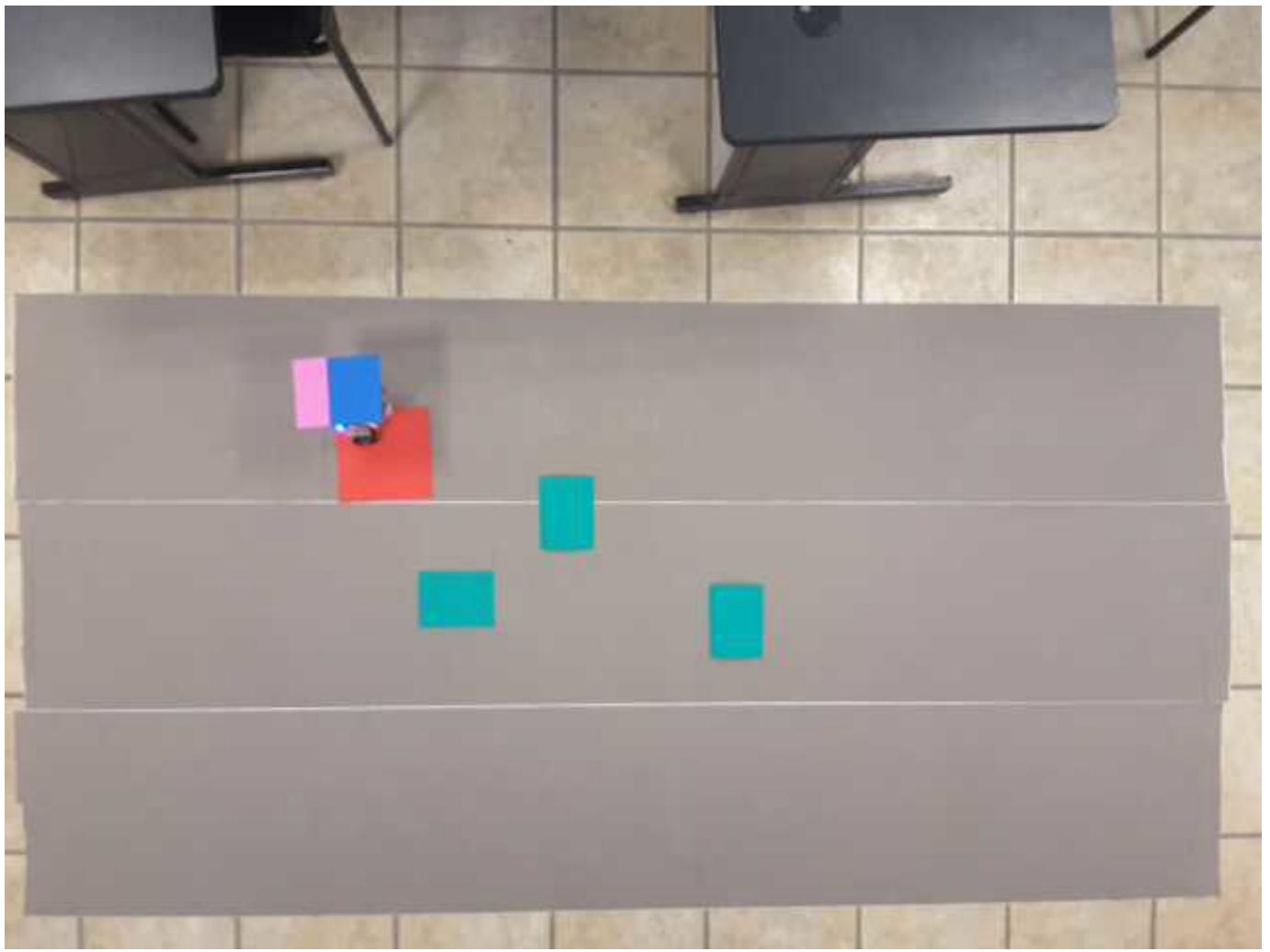


Figura 40: Imagen de la pista cuando el robot ha llegado ala meta.

8. Conclusiones

Se construyó una plataforma móvil capaz de recorrer un escenario con obstáculos y llegar a la meta establecida, se usó campos potenciales para la generación de vectores atractivos y repulsivos, para meta, para los obstáculos y robot, respectivamente.

Se implementó la interfaz en *PythonTM* para obtener la información de la ubicación de los centros de los obstáculos y la meta de una imagen (foto), así mismo, esta imagen se utilizó para encontrar los niveles de color del robot y la marca de orientación, para usar esos datos en la ubicación de los centros en la imagen en tiempo real (video), luego se hizo uso de toda la información anterior para generar sus respectivos campos potenciales y realizar los cálculos de la dirección hacia donde debería dirigirse, para evitar los obstáculos y llegar a la meta. Para el envío esa información al robot móvil desde la estación de trabajo (computadora), se utilizó una comunicación serial con el módulo bluetooth HC-05 de arduino, montado en la robot móvil. La comunicación establecida entre la computadora y el robot estable.

La decisión de utilizar *PythonTM* para la implementación del programa principal, en el cual se realizó la segmentación de las imágenes y la generación de los campos potenciales, fue debido al tiempo de cómputo menor que presenta, en comparación a los realizados en otros softwares.

La integración del programa principal con el realizado en la plataforma arduino, presentó algunos inconvenientes debido a la extensión de los datos que se intentaba transmitir, por lo que fué necesario implementar condicionales para el envío de la instrucción deseada, como se mencionó en la sección 4.2.1.

Las pruebas finales, se realizaron en el aula del instituto de posgrado de la Universidad Tecnológica de la Mixteca, en un ambiente controlado, por lo que, cualquier cambio de iluminación generará una variación en los umbrales de los canales de color para la segmentación, pero esto no representa un problema grave, ya que es posible utilizar las barras deslizantes para elegir los colores a segmentar, como se menciona en la sección 4.2.3.

Como propuesta para la mejora de este proyecto, se propone trabajar con un espacio de color más estable a los cambios de iluminación, como el espacio Cie L*a*b. Así como la implementación de un módulo MPU6050, el cual contiene un acelerómetro y giroscopio, para la determinación exacta del movimiento de la plataforma.

A. Anexos

A.1. Programa para segmentación y navegación usando campos potenciales:

```
1 import cv2
2 import numpy as np
3
4 from skimage.feature import peak_local_max
5 from skimage.morphology import watershed
6 from scipy import ndimage
7 import pylab as pl
8 from skimage.feature import peak_local_max
9 from skimage.morphology import watershed
10 from scipy import ndimage
11 import time
12 import matplotlib.pyplot as plt
13 import matplotlib.patches as patches
14 import math
15
16 import imutils
17 from contextlib import contextmanager
18
19 import serial,time
20
21 \makeatletter contextmanager
22 def keep_plots_open(keep_show_open_on_exit=True, even_when_error=True):
23     """
24     To continue executing code when plt.show() is called
25     and keep the plot on displaying before this context manager exits
26     (even if an error caused the exit).
27     """
28     import matplotlib.pyplot
29     show_original = matplotlib.pyplot.show
30     def show_replacement(*args, **kwargs):
31         kwargs['block'] = False
32         show_original(*args, **kwargs)
33     matplotlib.pyplot.show = show_replacement
34
35     pylab_exists = True
36     try:
37         import pylab
38     except ImportError:
39         pylab_exists = False
40     if pylab_exists:
41         pylab.show = show_replacement
42
43     try:
44         yield
```

```

45     except Exception as err:
46         if keep_show_open_on_exit and even_when_error:
47             print ("*****")
48             print ("Error early edition while waiting for show():")
49             print ("*****")
50             import traceback
51             print (traceback.format_exc())
52             show_original()
53             print ("*****")
54             raise
55     finally:
56         matplotlib.pyplot.show = show_original
57         if pylab_exists:
58             pylab.show = show_original
59     if keep_show_open_on_exit:
60         show_original()
61
62 def tomar_foto(file):
63     foto = cv2.VideoCapture('http://192.168.43.236:8080/shot.jpg')
64
65     retval, im=foto.read()
66     cv2.imwrite(file, im)
67     image = cv2.imread(file)
68     cv2.imshow("Foto Pista", image)
69     #del(foto)
70     return image
71
72
73
74 def scroll_bar(hsv,image,nameWin):
75     res_c=0
76     def nothing(x):
77         pass
78     img = np.zeros((300,512,3), np.uint8)
79     cv2.namedWindow('Scroll bar color')
80
81     # create trackbars for color change
82     cv2.createTrackbar('R','Scroll bar color',0,255,nothing)
83     cv2.createTrackbar('G','Scroll bar color',0,255,nothing)
84     cv2.createTrackbar('B','Scroll bar color',0,255,nothing)
85
86     # create switch for ON/OFF functionality
87     switch = '0 : OFF \n1 : ON'
88     cv2.createTrackbar(switch, 'Scroll bar color',1,0,nothing)
89
90     while(1):
91         cv2.imshow('Scroll bar color',img)
92         k = cv2.waitKey(1) & 0xFF
93         if k == 27:

```

```

94         break
95
96     # get current positions of four trackbars
97     r = cv2.getTrackbarPos('R','Scroll bar color')
98     g = cv2.getTrackbarPos('G','Scroll bar color')
99     b = cv2.getTrackbarPos('B','Scroll bar color')
100    sw = cv2.getTrackbarPos(switch,'Scroll bar color')
101
102    if sw == 1:
103        img[:] = [b,g,r]
104        #print(b,g,r)
105        bgr = np.uint8([[b,g,r ]])
106        bgr_hsv = cv2.cvtColor(bgr,cv2.COLOR_BGR2HSV)
107        #print(bgr_hsv)
108        h=bgr_hsv[0][0][0]
109        s=bgr_hsv[0][0][1]
110        v=bgr_hsv[0][0][2]
111        #print(h,s,v)
112        if h<235 and h>20:
113            hU=h+20
114            hL=h-20
115        else:
116            hU=255
117            hL=0
118
119        if s<225 and s>30:
120            sU=s+30
121            sL=s-30
122        else:
123            sU=255
124            sL=0
125
126        if v<215 and v>40:
127            vU=v+40
128            vL=v-40
129        else:
130            vU=255
131            vL=0
132        #print(hL,sL,vL)
133        #print(hU,sU,vU)
134
135        res_c=seg_foto(hsv,hL,sL,vL,hU,sU,vU,image)
136        cv2.imshow(nameWin, res_c)
137
138    else:
139        img[:] = 0
140        break
141
142    return res_c,hL,sL,vL,hU,sU,vU

```

```

143
144 def seg_foto(hsv ,hL ,sL ,vL ,hU ,sU ,vU ,image):
145
146     #Rango de colores detectados:
147     #Verdes:
148     color_bajos = np.array([hL, sL, vL], dtype=np.uint8)
149     color_altos = np.array([hU, sU, vU], dtype=np.uint8)
150
151     #Crear las mascaras
152     mascara_color = cv2.inRange(hsv, color_bajos, color_altos)
153
154     #Filtrar el ruido aplicando un OPEN seguido de un CLOSE
155     kernel = np.ones((6,6),np.uint8)
156     mascara_color = cv2.morphologyEx(mascara_color, cv2.MORPH_CLOSE, kernel)
157     mascara_color = cv2.morphologyEx(mascara_color, cv2.MORPH_OPEN, kernel)
158
159     # Bitwise-AND mask and original image
160     res_Seg = cv2.bitwise_and(image,image, mask= mascara_color)
161     #cv2.imshow("Segmentacion", res_Seg)
162     return res_Seg
163
164
165 def centro(nombre,colorCent):
166     gray = cv2.cvtColor(colorCent, cv2.COLOR_BGR2GRAY)
167     blurred = cv2.GaussianBlur(gray, (5, 5), 0)
168     thresh = cv2.threshold(blurred, 60, 255, cv2.THRESH_BINARY)[1]
169
170     # find contours in the thresholded image
171     cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.
172                             CHAIN_APPROX_SIMPLE)
173     cnts = cnts[0] if imutils.is_cv2() else cnts[1]
174
175     MIN_THRESH=0.00000001
176     #####
177     ####Para Unir los vectores
178     rec=0;
179     pos=0;
180     xCen=0
181     yCen=0
182     xACen=0
183     yACen=0
184     # loop over the contours
185     for c in cnts:
186         if cv2.contourArea(c) > MIN_THRESH:
187             # process the contour
188             # compute the center of the contour
189             M = cv2.moments(c)
190             cX = int(M["m10"] / M["m00"])

```

```

191
192     xCen=cX
193     yCen=cY
194     # print(xCen,yCen)
195
196     # draw the contour and center of the shape on the image
197     cv2.drawContours(colorCent, [c], -1, (0, 255, 0), 2)
198     cv2.circle(colorCent, (cX, cY), 7, (255, 255, 255), -1)
199     cv2.putText(colorCent, "Centro", (cX - 20, cY - 20), cv2.
200                 FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)
201
202     xVec=cnts[rec]
203
204     n=np.shape(xVec)
205     xCor=np.zeros((1,n[0]))
206     yCor=np.zeros((1,n[0]))
207
208     for i in range (0,n[0]):
209         xCor[0][i]=xVec[i][0][0]
210         yCor[0][i]=xVec[i][0][1]
211
212     if rec==0:
213         xAux=xCor
214         yAux=yCor
215         xACen=xCen
216         yACen=yCen
217     else:
218         xAux=np.append(xAux.T,xCor.T)
219         yAux=np.append(yAux.T,yCor.T)
220         #print(xACen,yACen)
221         xACen=np.append(xACen,xCen)
222         yACen=np.append(yACen,yCen)
223
224     rec=rec+1;
225
226     cv2.imshow(nombre, colorCent)
227     return xACen,yACen
228
229 def etiquetado(nombre,resultado):
230     w=0
231     shiftedR = cv2.pyrMeanShiftFiltering(resultado, 21, 51)
232     grayR = cv2.cvtColor(shiftedR, cv2.COLOR_BGR2GRAY)
233     threshR = cv2.threshold(grayR, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU
234                                         )[1]
235     DR = ndimage.distance_transform_edt(threshR)
236     localMaxR = peak_local_max(DR, indices=False, min_distance=20, labels=
237                                 threshR)
238     markersR = ndimage.label(localMaxR, structure=np.ones((3, 3)))[0]
239     labelsR = watershed(-DR, markersR, mask=threshR)

```

```

237     if nombre!="Etiquetado Robot":
238         print("[INFO] {} unique segments found".format(len(np.unique(labelsR)
239             ) - 1))
240
241     for label in np.unique(labelsR):
242         # if the label is zero, we are examining the 'background'
243         # so simply ignore it
244         if label == 0:
245             continue
246
247         # otherwise, allocate memory for the label region and draw
248         # it on the mask
249         maskR = np.zeros(grayR.shape, dtype="uint8")
250         maskR[labelsR == label] = 255
251
252         # detect contours in the mask and grab the largest one
253         cntsr = cv2.findContours(maskR.copy(), cv2.RETR_EXTERNAL, cv2.
254             CHAIN_APPROX_SIMPLE)[-2]
255         cr = max(cntsr, key=cv2.contourArea)
256
257         #draw a rectangle enclosing the object
258         x,y,w,h = cv2.boundingRect(cr)
259         rectan=cv2.rectangle(resultado,(x,y),(x+w,y+h),(0,255,0),2)
260         #print(rectan)
261         rect = cv2.minAreaRect(cr)
262         box = cv2.boxPoints(rect)
263         box = np.int0(box)
264         cv2.drawContours(resultado,[box],0,(0,0,255),2)
265         cv2.putText(resultado, "#{}".format(label), (int(x) - 10, int(y)),cv2
266             .FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)
267         #print(box)
268         cv2.imshow(nombre, resultado)
269
270     return w
271
272
273
274 def main():
275     arduino=serial.Serial("COM3",9600)
276     y, x = np.mgrid[0:480:480j, 0:640:640j]
277
278     nombre_foto="pista-3B.jpg"
279     image=tomar_foto(nombre_foto)
280     hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
281
282     nombre1="Segmentacion-1"
283     res_0,hL,sL,vL,hU,sU,vU=scroll_bar(hsv,image,nombre1)
284     ##    print("Limites",hL,sL,vL,hU,sU,vU)
285     cv2.imshow('Obstaculos', res_0)
286     nombre=("Centros Obstaculos")

```

```

283 xCen0,yCen0=centro(nombre,res_0)
284 nombre="Etiquetas Obstaculos"
285 w0=etiquetado(nombre,res_0)
286 cv2.destroyAllWindows(nombre1)
287 print(xCen0)
288 print(yCen0)
289 xCen0=np.append(xCen0,0)
290 yCen0=np.append(yCen0,0)
291 print(xCen0)
292 print(yCen0)
293
294 #-----Campo potencia de los obstaculos
295 alpha_obstacle, a_obstacle, b_obstacle = 1.0, 1.9e3, 1.9e3
296 pobs=0
297 lim=np.size(xCen0)
298 print('Longitud del vector ',lim)
299 yACen2=np.arange(lim)
300 lim2=np.size(yACen2)
301 print('Longitud del vector ',lim2)
302 for contador in range(lim-1):
303     yACen2[contador]=480-yCen0[contador]
304     x_obstacle = xCen0[contador-1]
305     y_obstacle = yACen2[contador-1]
306     p1= -alpha_obstacle * np.exp(-((x - x_obstacle)**2 / a_obstacle + (y
307         - y_obstacle)**2 / b_obstacle))
308     pobs=p1+pobs
309
310 nombre1="Segmentacion-2"
311 res_M,hL,sL,vL,hU,sU,vU=scroll_bar(hsv,image,nombre1)
312 cv2.imshow('Meta',res_M)
313 nombre=("Centros Meta")
314 xCenM,yCenM=centro(nombre,res_M)
315 nombreM="Etiqueta Meta"
316 wM=etiquetado(nombreM,res_M)
317 cv2.destroyAllWindows(nombre1)
318 print(xCenM,480-yCenM)
319
320 nombre1="Segmentacion-3"
321 res_R1,hL1,sL1,vL1,hU1,sU1,vU1=scroll_bar(hsv,image,nombre1)
322 ##    print("Limites",hL1,sL1,vL1,hU1,sU1,vU1)
323 cv2.destroyAllWindows(nombre1)
324 nombre1="Segmentacion-4"
325 res_R2,hL2,sL2,vL2,hU2,sU2,vU2=scroll_bar(hsv,image,nombre1)
326 ##    print("Limites",hL2,sL2,vL2,hU2,sU2,vU2)
327 cv2.destroyAllWindows(nombre1)
328
329 captura = cv2.VideoCapture('http://192.168.43.236:8080/video')
330
#Rango de colores detectados:

```

```

331 #Azules para el robot:
332 azul_bajos = np.array([hL1,sL1,vL1], dtype=np.uint8)
333 azul_altos = np.array([hU1,sU1,vU1], dtype=np.uint8)
334 #Azules para la marca de orientación:
335 rosa_bajos = np.array([hL2,sL2,vL2], dtype=np.uint8)
336 rosa_altos = np.array([hU2,sU2,vU2], dtype=np.uint8)
337
338 nombre="Centro del Robot"
339 nombre2="Centro dos"
340
341
342
343 #-----Campo potencial de la meta
344 x_meta = xCenM
345 y_meta = 480-yCenM
346 alpha_meta, a_meta, b_meta= 1.0, 3e4, 3e4
347 pmeta = alpha_meta * np.exp(-((x - x_meta)**2 / a_meta + (y - y_meta)**2
348 / b_meta))
349 contador2=0
350
351 while(1):
352     #Capturamos una imagen y la convertimos de RGB -> HSV
353     ret, imagen = captura.read()
354     cv2.imshow('Imagen Original', imagen)
355     hsv = cv2.cvtColor(imagen, cv2.COLOR_BGR2HSV)
356
357     #Crear las mascaras
358     mascara_azul = cv2.inRange(hsv, azul_bajos, azul_altos)
359     mascara_rosa = cv2.inRange(hsv, rosa_bajos, rosa_altos)
360
361     #Filtrar el ruido aplicando un OPEN seguido de un CLOSE
362     kernel = np.ones((6,6),np.uint8)
363     mascara_azul = cv2.morphologyEx(mascara_azul, cv2.MORPH_CLOSE, kernel
364         )
365     mascara_azul = cv2.morphologyEx(mascara_azul, cv2.MORPH_OPEN, kernel)
366     kernel = np.ones((6,6),np.uint8)
367     mascara_rosa = cv2.morphologyEx(mascara_rosa, cv2.MORPH_CLOSE, kernel
368         )
369     mascara_rosa = cv2.morphologyEx(mascara_rosa, cv2.MORPH_OPEN, kernel)
370
371     # Bitwise-AND mask and original image
372     res_b = cv2.bitwise_and(imagen,imagen, mask= mascara_azul)
373     res_r = cv2.bitwise_and(imagen,imagen, mask= mascara_rosa)
374
375     # show the image
376     #cv2.imshow("Imagen Azul", res_b)
377
378     #Ubicar el centro del robot
379     xcenR,ycenR=centro(nombre,res_b)

```

```

377     xcenR1,ycenR1=centro(nombre2,res_r)
378     #print("Centro Robot",xcenR,480-ycenR)
379     #print("Centro Marca",xcenR1,480-ycenR1)
380
381     print(wM)
382
383     if contador2>=100:
384         if np.size(xcenR)==1 and np.size(xcenR1)==1:
385             #Campo potencial del carro
386             x_carro = xcenR
387             y_carro = 480-ycenR
388             alpha_carro, a_carro, b_carro= 1.0, 3.9e3, 3.9e3
389             pcarro = -alpha_carro * np.exp(-((x - x_carro)**2 / a_carro +
390                                         (y - y_carro)**2 / b_carro))
390
391             #-----Campo potencia de los obstaculos
392             alpha_obstacle, a_obstacle, b_obstacle = 1.0, 1.9e3, 1.9e3
393             pobs=0
394             lim=np.size(xCen0)
395             yACen2=np.arange(lim)
396             lim2=np.size(yACen2)
397             for contador in range(lim-1):
398                 yACen2[contador]=480-yCen0[contador]
399                 x_obstacle = xCen0[contador-1]
400                 y_obstacle = yACen2[contador-1]
401                 p1= -alpha_obstacle * np.exp(-((x - x_obstacle)**2 /
402                                              a_obstacle + (y - y_obstacle)**2 / b_obstacle))
402                 pobs=p1+pobs
403
404             #-----Campo potencial de la meta
405             x_meta = xCenM
406             y_meta = 480-yCenM
407             alpha_meta, a_meta, b_meta= 1.0, 3e4, 3e4
408             pmeta = alpha_meta * np.exp(-((x - x_meta)**2 / a_meta + (y -
409                                         y_meta)**2 / b_meta))
410
411             ptotal=pmeta+pobs+pcarro          #Suma de todos los campos
412
413             print("Centro Robot",xcenR,480-ycenR)
414             print("Centro Marca",xcenR1,480-ycenR1)
415
416             #Genera los gradientes
417             dy, dx = np.gradient(ptotal, 1,1)
418             skip = (slice(None, None, 3), slice(None, None, 3))
419             fig,ax = plt.subplots()
420             ax.quiver(x[skip], y[skip], dx[skip], dy[skip], ptotal[skip])
421             ax.set(aspect=1, title='Quiver Plot2')
422             print (dx[480-ycenR1][xcenR1], dy[480-ycenR1][xcenR1])

```

```

423     #Calcula el angulo anterior
424     Angulo_Anterior=angulo=math.atan(((480-ycenR1)-(480-ycenR))/(
425         xcenR1-xcenR))*180/3.1416
426     print ('Angulo_Anterior', Angulo_Anterior)
427
428     #Calcula el angulo deseado
429     Angulo_Deseado=angulo=math.atan((dy[480-ycenR1][xcenR1])/(
430         [480-ycenR1][xcenR1]))*180/3.1416
431     print ('Angulo Deseado', Angulo_Deseado)
432
433     #Angulo a mover
434     Angulo_Mover=Angulo_Deseado-Angulo_Anterior
435     print('Angulo a mover', Angulo_Mover)
436     time.sleep(1)
437     print("limite x-", (x_meta)-(wM*.5))
438     print("limite x+", (x_meta)+(wM*.5))
439     print("limite y-", (y_meta)-(wM*.5))
440     print("limite y+", (y_meta)+(wM*.5))
441
442     if ((480-ycenR)>((y_meta)-(wM*.5)) and (480-ycenR)<((y_meta) +
443         +(wM*.5))):
444         if (xcenR>((x_meta)-(wM*.5)) and xcenR<((x_meta)+(wM*.5)))
445             :
446                 arduino.write(b'3')
447
448                 if Angulo_Mover <0:
449                     arduino.write(b'5')
450
451                 if Angulo_Mover >0:
452                     arduino.write(b'4')
453
454                 #if Angulo_Mover >= 0 and Angulo_Mover<15:
455                 #    arduino.write(b'1')
456
457                 contador2=0
458                 #####
459                 contador2=contador2+1
460                 tecla = cv2.waitKey(5) &      0xFF
461                 if tecla == 27:
462                     break
463
464             if __name__ == "__main__":
465                 main()

```

Referencias

- [1] T. D. Technologies, *www.tevo3dprinterstore.com*, 2017, consultado [30/07/2017].
- [2] P. Corporation, *www.pololu.com/product/2598*, consultado [10/03/2018].
- [3] ——, *www.pololu.com/product/713*, consultado [10/03/2018].
- [4] Arduino, *www.store.arduino.cc/usa/arduino-mega-2560-rev3*, consultado [08/03/2018].
- [5] P. S. Foundation. (2018, Jul.) Tutorial de python 3.6.3. [Online]. Available: <http://docs.python.org.ar/tutorial/3/real-index.html>