Mini Challenge 4 - Manchester Robotics

Diego García Rueda, Jhonatan Yael Martínez Vargas , Jonathan Josafat Vázquez Suárez, Daniela Berenice Hernández de Vicente

[1] *Instituto Tecnológico de Estudios Superiores de Monterrey, Campus Puebla, Puebla, México*
**Autores correspondientes: A01735217@tec.mx,A01734193@tec.mx,A01734225@tec.mx,A01735346@tec.mx*

Publicado el 28/05/2023

El siguiente reporte pretende explicar la interacción de un robot diferencial en búsqueda de cierta autonomía y toma de decisiones por medio de sensores, lo cual genera un involucramiento en la implementación de un sistema de control inteligente y robusto. Aunado a esto se explicará el funcionamiento de la propuesta para la resolución del reto semanal presentado por manchester robotics.

Diagrama 1.

El diagrama 1 hace referencia al objetivo final del reto presentado por MCR2, el cual se trabajó durante esta última semana, donde se busca realizar la trayectoria planteada por cada equipo al igual que el reconocimiento de los semáforos.

---

1. Introducción.

A. Introducción .

A lo largo de la historia la evolución de la tecnología ha sido inevitable, y con ella han surgido muchos retos relacionados con la misma.

Por lo que la robótica se ha vuelto un pilar destacado en cuestión de avances tecnológicos, de igual manera esta puede generar satisfacción al resolver necesidades donde se busca hacer más eficaz una tarea o un proceso determinado.

Como bien sabemos esta abarca bastantes disciplinas como lo es la programación, la mecánica, la mecatrónica, la electrónica, entre otras, siendo esta una ciencia que busca cierta innovación tecnológica.

Por lo que en la actualidad es un concepto que busca la facilidad de la sociedad y su mejora continua por medio de robots inteligentes y autónomos.

2. Objetivos.

A. Objetivos Generales.

A continuación se presentan los objetivos generales, donde se describe la función general con la que debe cumplir el mini challenge 4. [1]

- En este challenge se pretende que el estudiante revise y comprenda los conceptos introducidos en las sesiones.
- Este challenge tiene como objetivo enseñar el comportamiento de los sistemas de visión en los robots móviles.
- Este challenge se dividirá en diferentes secciones.

B. Objetivos Específicos.

A continuación se presentan los objetivos específicos del mini challenge 4, donde se pretende explicar más a fondo lo que se espera realizar durante el mismo.

- Se debe agregar una capa de seguimiento de línea a la capa de decisión y al algoritmo de navegación punto a punto, los que anteriormente han sido desarrollados, esto con el fin de poder generar el seguimiento de línea mediante la cámara del puzzlebot.

El comportamiento esperado es el siguiente:

- Se espera que el estudiante diseñe un algoritmo robusto, el cual se encargue del seguimiento de una línea mediante la cámara del robot.

- La pista debe estar compuesta por 3 líneas con ciertas medidas (estas se mencionan más adelante).
- La pista debe estar diseñada por los estudiantes, donde se evaluará la complejidad de la misma.
- El comportamiento del algoritmo debe tomar en consideración el challenge de los semáforos.
    - Luz Roja: Detenerse hasta que aparezca una luz verde.
    - Luz Amarilla: Avance despacio hasta que aparezca una luz roja para detenerse completamente.
    - Luz Verde: Continúe por el camino.
- El algoritmo de visión de seguimiento de línea y el controlador de circuito cerrado deben ser ***robustos***.
    - El estudiante debe definir qué significa robusto e implementar estrategias para lograr el éxito con el controlador.
- El algoritmo de visión, los algoritmos de seguimiento de línea y el controlador deben estar sintonizados apropiadamente.
- Es recomendado, más sin embargo no obligatorio, el reconocimiento de la bandera a cuadros para finalizar el recorrido.
- El controlador debe tomar en consideración los cambios del semáforo, perturbación, nolinealidades y el ruido.
- Es recomendado, pero no requerido, que el estudiante use un archivo de configuración o un parámetro en el archivo de lanzamiento para establecer los objetivos de modo que puedan cambiarse fuera del código (no codificados).

3. MARCO TEÓRICO

A. Control en lazo cerrado para un robot móvil.

Un robot móvil se presenta como un control de lazo cerrado, el cual puede realizar un cambio de posición de sus elementos, esto en función de la información captada de su entorno mediante sus sensores. [1]

Es importante reconocer que es casi imposible tunear el controlador para que este genere una trayectoria perfecta hacia la posición deseada en el robot.

Ante esta situación también es importante reconocer que al no ser exactamente perfecto, se busca generar una tasa de error muy baja. [2]

De igual manera es importante destacar que este es un sistema que utiliza fundamentalmente la retroalimentación para mantener una variable controlada, al contar con la medición de dicha variable, esta se compara con el valor de referencia dado anteriormente, y cualquier diferencia entre estos dos valores es utilizada para ajustar el controlador empleado.

B. PID aplicado a un robot móvil diferencial y los temas derivados al cálculo del error.

Como anteriormente se mencionó la programación en ROS no fue lo más importante durante este proyecto, aunque sí es la parte que predomina en este proyecto, ya que la gran parte del proyecto es la creación de los nodos y la comunicación entre ellos, pero el sistema implementado no podría funcionar correctamente sin un controlador que satisfaga las necesidades del mismo.

Por lo que para poder implementar el controlador adecuado para el sistema, fue necesario comprender los tipos de controladores existentes y en qué momento es necesario utilizarlos, al igual que la función de cada una de las variables de los mismos, de igual manera es importante conocer que el diseño del sistema de control consistirá en elegir adecuadamente la localización de los ceros, polos y la ganancia K, por otro lado gracias al controlador podemos llegar a tener cierto tiempo muerto, por lo que es importante conocer el significado de dicho concepto [3].

Ganancia de Proceso(K). Es la relación de cambio de salida al cambio de la variable de entrada. Esta define más específicamente la sensibilidad que tiene la variable de salida a un cambio dado en la variable de entrada. Dicho comportamiento se muestra matemáticamente en la ecuación 1 [3].

$$K = \frac{\Delta \text{ Output}}{\Delta \text{input}}$$

Ecuación 1.

Esta solo puede describirse como un parámetro de estado estacionario y es independiente de las variables de diseño y operación. Cuenta con solo tres componentes, el signo, el valor y las unidades.

El primero indica cómo responderá la salida a la entrada del proceso. El segundo depende del proceso que se esté considerando, por lo que dependen de las variables que se mencionan.

Tiempo Muerto (t0). Sucede cuando se da el cambio en una variable de entrada y cuando inicia la variable de salida. Cabe mencionar que es muy importante, porque afecta a la controlabilidad de nuestro sistema de control, por lo que un cambio en el punto de ajuste no suele ser inmediato gracias a este parámetro. De igual manera, este tiempo muerto debe ser siempre considerado para los procesos de afinación y modelado [3].

Control Proporcional. Es una forma de control de retroalimentación, siendo la forma más fácil de control continuo que se puede utilizar en un sistema de lazo cerrado. También minimiza la fluctuación en la variable de proceso, el lado negativo es que no siempre lleva al sistema al punto de ajuste que se desea [3].

Por otro lado proporciona una respuesta más rápida que la mayoría de los otros controladores, lo que permite inicialmente que el controlador solo P llegue a responder unos segundos más rápido. Aunque el controlador P ofrece como ventaja un tiempo de respuesta más rápido, llega a producir una desviación del punto de ajuste, la cual se conoce como desplazamiento, y es algo que no se desea durante un proceso.

La existencia de ella implica que el sistema no podría mantenerse en el punto de ajuste deseado en estado estacionario. Este se puede minimizar combinando el control P con algún otro controlador (I o D)

Por lo que el controlador P correlaciona la salida del controlador con el error. Dicho comportamiento se muestra matemáticamente en la ecuación 2.

$$c(t) = K_c e(t) + b$$

Ecuación 2.

Control Integral. Es una segunda forma de control de retroalimentación. Usualmente se utiliza porque es capaz de eliminar cualquier desviación que pueda existir, de esta manera el sistema vuelve tanto al estado estacionario como a su configuración original [3].

Por lo tanto, un error negativo hará que la señal al sistema disminuya, mientras que un error positivo hará que la señal aumente. Sin embargo este tipo de controladores suelen ser más lentos en su tiempo de respuesta que los controladores P solo porque dependen de más parámetros. Este tiempo de respuesta se puede aumentar si es combinado con algún otro controlador (P o D). De igual manera suelen usarse por separado cuando las variables medidas deben permanecer dentro de un rango muy estrecho y requieren un ajuste más fino.

Estos suelen afectar al sistema al responder a errores pasados acumulados, por lo que su filosofía es que las desviaciones se verán afectadas en proporción a la suma acumulada de su magnitud. Ahora bien, una gran ventaja de estos es que se eliminara el desplazamiento, mientras que sus grandes desventajas son que puede llegar a desestabilizar el controlador, y que hay una windup del integrador, lo cual aumenta el tiempo que tarda el controlador en realizar ciertos cambios.

El control I correlaciona la salida del controlador con la integral del error, por lo que la integral del error se toma con respecto al tiempo. Es el error total asociado a lo largo de una cantidad de tiempo especificada. Dicho comportamiento se muestra matemáticamente en la ecuación 3.

$$c(t) = \frac{1}{T_i} \int e(t)dt + c\,(t_0)$$

Ecuación 3.

Control Derivativo. Es una forma de control de avance, ya que este anticipa las condiciones del proceso analizando el cambio en el error. Este funciona para minimizar el cambio de error, de esta forma mantiene el sistema en una configuración consistente. Uno de sus principales beneficios es que resisten el cambio en el sistema, siendo el más importante de estas oscilaciones. Ahora bien, la salida del control se calcula en base a la tasa de cambio del error con el tiempo, por lo que mientras más grande sea la tasa de cambio de error, más pronunciada será la respuesta del controlador [3].

A diferencia de los otros controladores, el derivativo no guía al sistema a un estado estacionario, ahora bien, debido a esto se recomienda que deben estar combinados con los controladores P, I o PI para poder controlar de manera correcta el sistema.

El control D correlaciona la salida del controlador con la derivada del error. La derivada del error se toma con respecto al tiempo. Dicho comportamiento se muestra matemáticamente en la ecuación 5.

$$c(t) = T_d \frac{de}{dt}$$

Ecuación 5.

Control PI. El control PI es una forma de control de retroalimentación. Este proporciona un tiempo de respuesta más rápido que el control I debido a la suma de la acción proporcional, aunque sigue siendo hasta un 50% más lento que el control P. Por lo que con el fin de aumentar el tiempo de respuesta, el control PI suele combinarse con el control D. De igual manera este impide que el sistema fluctúe, y es capaz de devolver el sistema a su punto de ajuste [2].

Como se ha mencionado el control PI correlaciona la salida del controlador con el error y la integral del error. Dicho comportamiento se muestra matemáticamente en la ecuación 4.

$$c(t) = K_c \left(e(t) + \frac{1}{T_i} \int e(t)dt + T_d \frac{de}{dt} \right) + C$$

Ecuación 4.

Control PD. El control PD es una combinación entre control de avance y retroalimentación, debido a que opera tanto en las condiciones actuales del proceso como en las condiciones del proceso predichas . En este controlador, su salida es una combinación lineal de la señal de error y su derivada. De igual manera contiene la amortiguación del control proporcional de la fluctuación y la predicción del error del control derivado [2].

Como se ha mencionado el control PD correlaciona la salida del controlador con el error y la derivada del error. Dicho comportamiento se muestra matemáticamente en la ecuación 5.

$$c(t) = K_c \left(e(t) + T_d \frac{de}{dt} \right) + C$$

Ecuación 5.

Control PID. Este es el más utilizado, ya que combina las ventajas de cada tipo de control. Por lo tanto tenemos un tiempo de respuesta más rápido debido al control P, junto con el desplazamiento decrecido/cero de los controladores D e I. Es importante destacar que el desplazamiento anteriormente mencionado se elimina mediante el uso adicional del control I [2].

La adición del controlador D aumenta en gran medida la respuesta del controlador cuando se usa en combinación con los demás, ya que predice la respuesta del sistema midiendo el cambio en el error. Sin embargo, aunque el controlador PID parece ser el controlador más adecuado, también es el más caro. Por lo tanto, realmente no se utiliza a menos que el sistema necesite esa precisión y estabilidad [3].

Por lo que el controlador PID correlaciona la salida del controlador con el error, siendo estos el integral del error y derivada del error. Dicho comportamiento se muestra matemáticamente en la ecuación 6.

$$c(t) = K_c \left(e(t) + \frac{1}{T_i} \int e(t)dt + T_d \frac{de}{dt} \right) + C$$

Ecuación 6.

C. Implementación del sistema de control.

En esta ocasion la propuesta que diseñamos fue la siguiente:

1. El nodo de control recibe el ángulo de inclinación de la linea central que el robot tiene delante.

```
rospy.Subscriber("/set_point", SetPoint, self.set_point_callback)
```

```
def set_point_callback(self, msg):

    print("Nuevo Objetivo Recibido:")
    self.target.id = msg.id
    self.target.orientation = msg.orientation
    print(self.target.id, " --- ",self.target.orientation)
    print("")
```

2. Se calcula el error de orientación, mediante la orientacion actual del robot y la recibida en este mismo nodo.

```
# Calculo de error
self.prev_orientation_error = self.orientation_error

self.current.orientation = self.angle_wrap(self.current.orientation)

self.orientation_error = self.target.orientation - self.current.orientation
self.orientation_error = self.angle_wrap(self.orientation_error)

print("Orientacion Objetivo:")
print(self.target.orientation)
```

3.

4. Se establece un margen de error para determinar si el robot debe girar o no.

5.
```
if self.orientation_error <= -0.02 or self.orientation_error >= 0.02:
```

6. Se indica el sentido de giro y se pasan la POSE actual del robot al controlador PID.

```
    if self.orientation_error > 0.0:

        print("")
        print("Gira Izquierda")
        print("")

        self.angular_control("wl", dt)

        next_orientation = self.current.orientation - (self.r * ((self.wr - self.wl) / self.l) * dt)
        self.current.orientation = next_orientation

        self.last_time = rospy.get_time()

    elif self.orientation_error < 0.0:

        print("")
        print("Gira Derecha")
        print("")

        self.angular_control("wr", dt)

        next_orientation = self.current.orientation + (self.r * ((self.wr - self.wl) / self.l) * dt)
        self.current.orientation = next_orientation

        self.last_time = rospy.get_time()

else:

    print("Trayectoria recta")
```

7. El controlador hace un ajuste unicamente a las velocidades angulares.

```
def angular_control(self, id, dt):

    p = 0
    i = 0
    d = 0
    new_vel = 0

    # Calculo Control Proporcional
    p = self.angular_kp * self.orientation_error

    # Calculo control Integral
    self.integral_orientation += self.orientation_error * dt
    i = self.angular_ki * self.integral_orientation

    # Calculo Control Derivativo
    self.derivative_orientation = (self.orientation_error - self.prev_orientation_error) / dt
    d = self.angular_kd * self.derivative_orientation

    self.vel.linear.x = 0.0

    new_vel = p + i + d

    if id == "wl":

        if new_vel >= self.max_angular_vel:

            self.vel.angular.z = -self.max_angular_vel

        else:

            self.vel.angular.z = -new_vel

    else:

        if new_vel >= self.max_angular_vel:

            self.vel.angular.z = self.max_angular_vel

        else:

            self.vel.angular.z = new_vel
```

8. Se anexa la parte de reduccion de velocidad al recibir alguna de las señales sobre un semaforo.

```
# --- Semaforos ---
if self.target.id == 1:

    self.vel.angular.z *= 1.0

elif self.target.id == 2:

    self.vel.angular.z *= 0.5

elif self.target.id == 3:

    self.vel.angular.z *= 0.0

else:

    self.vel.angular.z *= 0.33
```

D. Procesamiento de imágenes en una tarjeta embebida

Por otro lado también durante este reto se manejo el uso de tarjetas embebidas como lo es la tarjeta Jetson, esto con la finalidad de lograr un buen procesamiento de imágenes, siendo esta una parte fundamental del proyecto.

Por lo que para ello fue necesario el uso de OpenCV, siendo una librería open-source de machine learning y visión artificial. Los algoritmos que la integran son más de 2500, los cuales incluyen machine learning y visión artificial. Estos permiten identificar objetos, clasificar imágenes con respecto a una base de datos, hacer tracking de movimiento de objetos, entre otras cosas. [4]

Ahora bien, de igual manera es importante mencionar que el procesamiento de la imagen recibida durante este reto, no es completo, es decir que se seleccionan específicamente ciertas regiones de interés.

Donde una región de interés (ROI) es una parte de la imagen que se desea filtrar o procesar de alguna manera. Esta se puede representar como máscara binaria. En la misma, los píxeles que pertenecen a la ROI tendrán el valor 1 y los que se encuentran afuera de esta tendrán el valor 0. [5]

De igual manera se destaca que puede haber más de una región de interés, esto es importante recordarlo, ya que en este mini challenge no solo se tienen que identificar los semáforos, sino que también se deben identificar las líneas para poder hacer un seguimiento de las mismas y con esto no solo poder hacer el recorrido, sino que también no salirse del mismo.

E. Interconexión entre jetson y la cámara

Siendo esta la conexión entre la tarjeta NVIDIA y la cámara que el *Puzzlebot* posee, se utiliza un archivo .launch proporcionado por MCR2 que permite inicializar la cámara y enviar las imágenes recibidas por un tópico llamado *video_source_raw* que posteriormente utilizando la librería *cv_bridge* permitirá utilizar librerías de OpenCV a la imagen recibida

F. Detección de contornos o formas en una imagen.

La detección de contorno es parte del proceso de segmentación, el cual consiste en identificar objetos dentro de una imagen.

Ahora bien, como antes se mencionó, la creación de varias ROI hace más sencilla esta parte, ya que si se tomará la imagen completa sería mucho más difícil filtrar los contornos pertinentes, ya que aparecerán muchas más que puede que cumplan con los requerimientos para ser detectados, esta parte será desarrollada en la sección **Procesamiento de Imagenes** del Marco Metodológico..

G. Detección de colores.

Ahora bien, es cierto que primordialmente se busca la detección de bordes para poder detectar el círculo con el algoritmo de visión computacional, sin embargo de igual manera se busca la detección de colores, esto con el fin de realizar la acción necesaria

Por lo que un campo de color se establece como un conjunto de colores primarios a partir de los que mediante ciertas mezclas se pueden obtener otros colores cubriendo de esta forma todo el espectro posible.

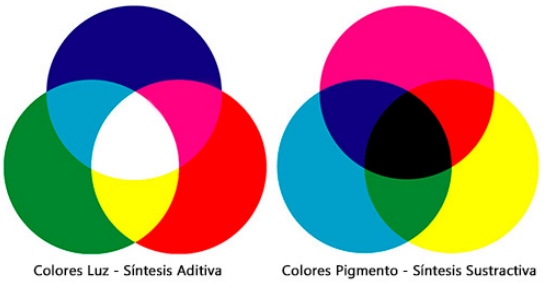


Imagen 1. Círculo cromático.

En la imagen anterior se muestran dos teorías del color completamente distintas, siendo estas la teoría de colores de luz y la teoría de colores pigmento, donde en cada una se tiene colores primarios, ya que para los colores pigmento se tiene como primario al magenta, cian y al amarillo, mientras que para los colores luz se tiene como primordial al rojo, azul y verde.

En este proyecto es primordial enfocarnos la teoría de color en luces, debido a que las camaras actuan de forma similar al ojo humano, ya que este se utiliza cuando se representa color mediante haces de luz.

Por lo que un píxel en un monitor se representaría mediante tres subpíxeles, siendo una roja, una verde y una azul, las cuales corresponden a un LED o diodo emisor de luz.

Ahora bien, teniendo en cuenta que la imagen detectada llega principalmente a nosotros con un campo de color RGB lo que se hace es transformar esta imagen al espectro HSV, (Tonalidad, Saturación, Brillo), este es un derivado del espacio RGB y representa los colores combinando los tres valores mencionados anteriormente.

Estas magnitudes suelen pueden tener los siguientes valores. [6]

- H-Tonalidad: Valores de 0 a 360.
- S-Saturación: Valores de 0 a 100. Por lo que es de menor a mayor cantidad.
- V/B-Brillo: Valores de 0 a 100. Por lo que el color es totalmente oscuro hasta su máxima luminosidad.

H. Robustez en sistemas de procesamiento de imágenes

El sistema de procesamiento de imágenes de este proyecto mostró ser bastante robusto, buscando el equilibrio entre un correcto procesamiento de las imágenes y un tiempo de procesamiento rápido para la obtención de las imágenes en tiempo real, el procesamiento se realiza para el seguimiento de

la trayectoria y para la detección de semáforos, el procedimiento será descrito en la siguiente sección.

4. Marco Metodológico.

Cada uno de los conceptos anteriormente mencionados se han utilizado para la resolución de manera práctica de los diferentes problemas que han surgido durante las 5 semanas del bloque.

A. Funcionamiento del sistema general en ROS.

La estructura de este proyecto fue diferente a la estructura de retos anteriores, anteriormente existía un nodo llamado *Set_Point* que funcionaba como determinación de la trayectoria a seguir del robot ingresando puntos de coordenadas, en esa estructura existían otros 2 nodos: *controller* e *Image_detection* encargados del sistema de control PID de control y la detección de semáforos respectivamente. La estructura anterior puede mostrarse en el siguiente diagrama:

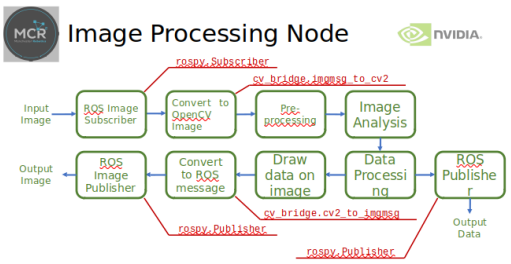


Imagen 2. Estructura anterior del sistema.

El nodo *Set_Point* se eliminó de la estructura siendo innecesario el uso de puntos definidos a seguir por parte del robot ya que en este nuevo reto se usa una trayectoria ya definida. El nodo *Image_Detection* se vio modificado para ahora poder incluir el análisis de la pista además de la detección de semáforos, siendo así este nodo el que manda parámetros al nodo *controller* para que el robot se adecue a la trayectoria.

La nueva arquitectura pasa a ser únicamente de 2 nodos conectados por los tópicos:

- /angle: Se encarga de mandar el ángulo de giro que debe seguir el robot para el seguimiento de la trayectoria
- /set_point: Determina el color del semáforo detectado o si es que no se detecta ningún semáforo, estos valores se representan por números enteros,

Con los 2 tópicos anteriores el nodo *controller* obtiene los valores necesarios para el ajuste del robot y la velocidad en la que recorrerá esta trayectoria (o si el semáforo es de color rojo generará un alto total).

B. Funcionamiento del controlador.

C

C. Procesamiento de imágenes.

En este reto se realizaron 2 procesamientos de imágenes diferentes, en primer lugar para la detección de semáforos se asignan valores en escala HSV para los colores verde, amarillo y rojo

```
self.lower_red = np.array([0, 100, 100])
self.upper_red = np.array([10, 255, 255])
self.lower_yellow = np.array([15, 100, 100])
self.upper_yellow = np.array([35, 255, 255])
self.lower_green = np.array([40, 100, 100])
self.upper_green = np.array([80, 255, 255])
```

Con estos valores se realiza una máscara usando los valores anteriores, a partir de esta máscara se detectan los contornos de la imagen. Aplicando librerías de OpenCV se realiza una limpieza de la imagen, y la obtención del área y del perímetro de la figura detectada, se obtiene la circularidad de la figura, usando la circularidad como parámetro se detectan círculos de los colores anteriormente mencionados a modo de representación de los semáforos.

En el caso de la detección de las trayectorias se realiza un procesamiento diferente, primero la imagen se convierte de escala, pasa de RGB a una imagen en escala de grises, la imagen se limpia utilizando un desenfoque Gaussiano, se obtienen los bordes mediante la librería *cv2.Canny* y finalmente de los bordes detectados se utiliza la función *HoughLInesP.*

Esta librería usa una técnica de detección de líneas en una imagen usando la transformada de Hough probabilística. Su objetivo es encontrar segmentos de líneas rectas en una imagen.La transformada de Hough es un algoritmo que se utiliza para detectar formas lineales en una imagen. La función toma como entrada una imagen binarizada y devuelve una lista de segmentos de líneas detectados en la imagen. Para la binarización de la imagen debido z que se intentan localizar los carriles negros de la pista, se decidió por aplicar una conversiónna escala de grises para posteriormente aplicar un umbralizado que permitiera ubicar unicamente los carriles de color negro.

Después de aplicar la función *HoughLines* se obtienen los ángulos de cada uno de los segmentos de línea detectados, esto aplicando una operación de arcotangente a la diferencia de posición del punto inicial y final de la línea calculando la diferencia en el eje X y en el eje Y. Tras haber obtenido los ángulos de cada una de las líneas se determina el ángulo dominante encontrando el ángulo más común en las líneas detectadas. A continuación, se seleccionan las líneas que tienen un ángulo cercano al ángulo dominante. Estas líneas se consideran como las líneas dominantes. Finalmente, se calcula el ángulo de orientación de la línea dominante en relación con la coordenada vertical predefinida mediante el cálculo del arcotangente de esta línea. El ángulo se redondea a dos decimales y se devuelve como resultado.

Este ángulo es el que se manda a través del tópico: */angle* y que definirá el ángulo en radianes que tiene que girar el robot. Como dato adicional, existe la opción de que no se detecte ningún carril por lo que en este caso no se detectan bordes y por lo tanto tampoco se obtendría un ángulo en este caso se regresa un valor None, cuando se reciba un valor así el robot seguirá la trayectoria del ángulo anterior con un valor diferente de None que se haya recibido.

```
if lines is not None:
    #Resultados ya se encuentran en radianes:
    lines_angles = np.arctan2(lines[:, :, 3] - lines[:, :, 1], lines[:, :, 2] - lines[:,
:, 0])
    unique_angles, counts = np.unique(np.round(lines_angles/7), return_counts=True)
    dominant_angle = (unique_angles[np.argmax(counts)])*7
    dominant_lines = lines[np.abs(lines_angles - dominant_angle) <= 10]

    points = dominant_lines.reshape(-1, 2)
    coefficients = np.polyfit(points[:, 0], points[:, 1], 1)

    x = max(min(np.round((80 - coefficients[1]) / coefficients[0], 2), 320), 0)
    #Angulo ya en radianes (2 decimales)
    angle = np.round(np.arctan2(240 - 80, 160 - x)* 180 / np.pi, 2)

    return angle
```

```
else:
    return None
```

D. Definición de la Región de Interés (ROI).

Es importante tener en cuenta que a partir de la toma de video de la cámara la zona en la que se encuentran las líneas de los carriles es una zona específica, para esto se determina una Región de Interés (Region of Interest o ROI), en esta región únicamente se encontrarán los carriles, establecer una región de este tipo permite que el procesamiento sea más rápido ya que se limita la zona de procesamiento, estos valores tuvieron que ajustarse manualmente de acuerdo a la posición de la cámara en el *Puzzlebot* .

Imagen 3. Ejemplo Visión Camara Puzzle Bot.

Los valores obtenidos que funcionaban de una buena manera fueron los siguientes:

```
self.roi_upper = 0.86
self.roi_lower = 0.99
self.roi_left = 0.35
self.roi_right = 0.65
```

Los valores se encuentran representados a manera de porcentajes debido a que una ROI permite mejorar el tiempo de procesamiento, un reajuste de tamaño de una imagen también acelera el procesamiento al disminuir las dimensiones de la imagen recibida, por lo tanto al usar porcentajes se puede realizar este ajuste independientemente de las dimensiones de la imagen.

Cabe destacar que al momento de hacer pruebas, esta ROI probó no ser la mejor ya que por momento en ella no se obtenía la información necesaria para el seguimiento de la trayectoria debido a la escasa o nula visión del carril, se realizaron varias pruebas con diferentes valores, ninguna demostró servir para todas las situaciones, la ROI que nos mostró mejores resultados es la que utiliza los valores anteriores, la configuración y ajuste de la ROI es uno de los aspectos principales a modificar o mejorar en futuras entregas.

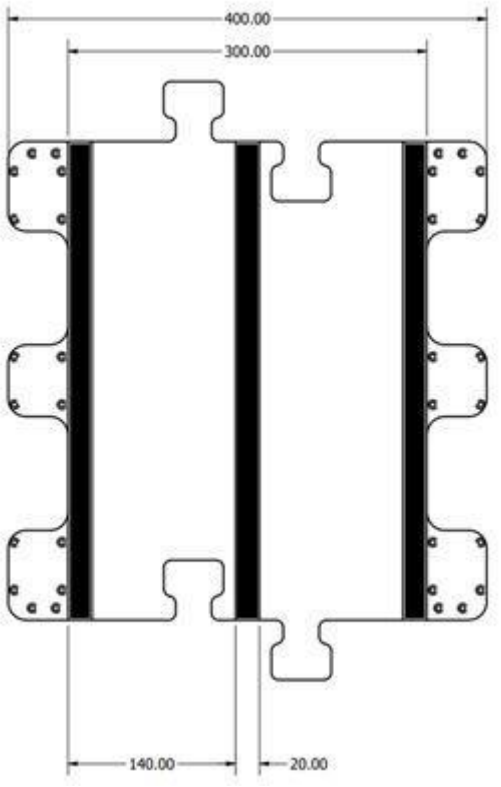
E. Planeación de trayectoria.

Parte fundamental de esta solución fue la planeación de la trayectoria que el puzzlebot debe recorrer.

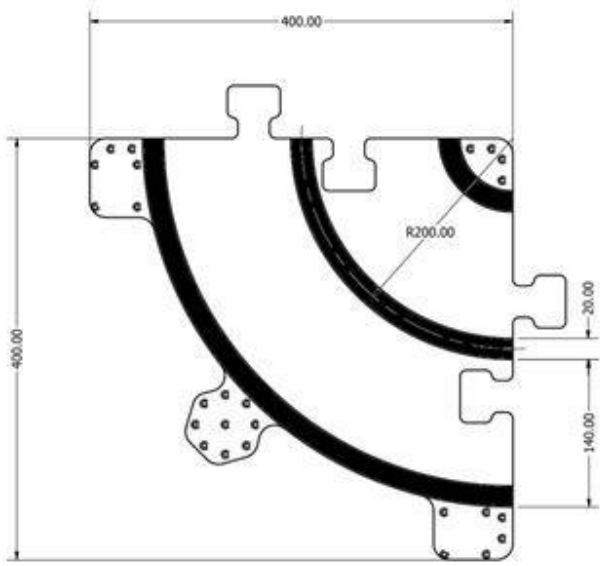
Para ello implementamos la siguiente forma o trayectoria.

Dicha trayectoria está compuesta por 4 semicurvas y 2 curvas completas, las cuales tienen las siguientes medidas entre las curvas.

Medidas para tramos rectos:



Medidas para tramos curvos:



Cabe destacar que estas son medidas dadas en milímetros.

5. Resultados

Una vez realizada la explicación de cada uno de los análisis y cálculos para obtener los valores específicos del controlador y de la ejecución de cada uno de los códigos para los nodos, y para el algoritmo de visión para el reconocimiento de los semáforos y el seguimiento de líneas, se realizaron pruebas para la comprobación del funcionamiento del sistema, los resultados pueden observarse en el segundo enlace del Anexo.

6. Conclusión.

Para poder diseñar un sistema de seguimiento de línea utilizando únicamente la cámara como sensor principal, fue una tarea bastante compleja, sobre todo porque siempre se ha estado lidiando las diferentes exposiciones de luz a las que está expuesta el robot en diferentes horas del día, lo cual conlleva a tener que plantear un algoritmo de cierta manera responsivo que permite cubrir un rango amplio de niveles HSV. Además de que diseñar una estrategia que nos permitiera indicarle al robot cual es la dirección y los pequeños ajustes que este tiene que hacer a su ángulo de orientación mediante la cámara también resultó ser un trabajo complicado de implementar.

7. Referencias.

1. ManchesterRoboticsLtd. (s. f.). TE3002B_Intelligent_Robotics_Implementation/MCR2_Mini_Challenge4.pptx at main · ManchesterRoboticsLtd/TE3002B_Intelligent_Robotics_Implementation. GitHub. https://github.com/ManchesterRoboticsLtd/TE3002B_Intelligent_Robotics_Implementation/blob/main/Week%206/Challenges/MCR2_Mini_Challenge4.pptx
2. Manchester Robotics [MCR2]. (s. f.). Closed Loop Control_v2. Github. Recuperado 25 de abril de 2023, de https://github.com/ManchesterRoboticsLtd/TE3002B_Intelligent_Robotics_Implementation/blob/main/Week%203/Challenges/MCR2_Mini_Challenge2.pdf
3. Libretexts. (2022, 2 noviembre). 9.2: Control de P, I, D, PI, PD y PID. LibreTexts Español. https://espanol.libretexts.org/Ingenieria/Ingenier%C3%ADa_Industrial_y_de_Sistemas/Libro:_Din%C3%A1mica_y_Controles_de_Procesos_Qu%C3%ADmicos_(Woolf)/09:_Control_proporcional-integral-derivado_(PID)/9.02:_Control_de_P,_I,_D,_PI,_PD_y_PID
4. OpenCV. (2023, 24 mayo). OpenCV - Open Computer Vision Library. https://opencv.org/
5. Procesamiento basado en ROI - MATLAB & Simulink - MathWorks América Latina. (s. f.). https://la.mathworks.com/help/images/roi-based-processing.html
6. Color luz y color pigmento. (s. f.). https://www.fotonostra.com/grafico/colorluzpigmento.htm

8. Anexos.

1. Repositorio de GitHub: https://github.com/JM-Yamajo/Puzzlebot_Challenges/tree/main/Week04
2. Link del video explicativo: https://www.youtube.com/watch?v=xY_hdligOOA&ab_channel=JhonatanYaelMart%C3%ADnezVargas