Implementación de robótica inteligente

Reyna Reyes D., Barrón Martínez S., Hoyo García J.

Departamento: Computación
Curso: TE3001B - Fundamentación de Robótica
Instructor: Rolando Bautista Montesano
Fecha: February 27, 2023

Abstract

This report details the implementation of an autonomous vehicle using tools such as the Robot Operating System and the OpenCV library. The robot used is a Puzzlebot Jetson Edition from Manchester Univeristy.

1 Introducción

Durante 10 semanas se trabaja un reto enfocado en el desarrollo de un prototipo de vehículo con navegación autónoma, con un sistema que identifique de manera confiable y precisa otros vehículos, señales, la pista entre otros objetos. A su vez, debe ser capaz de tomar decisiones en tiempo real para garantizar el correcto seguimiento del recorrido. Algunos beneficios de los vehículos autónomos son la reducción de accidentes viales, mayor comodidad y movilidad para todas las personas, agilización de las vías urbanas, el cuidado del medio ambiente por mencionar algunas. (Big 2020)

2 Identificación y análisis del problema

El vehículo autónomo diferencial se implementará con visión computacional apoyándonos en una cámara montada en el frente. Esta va conectada a una placa JETSON de NVIDIA que incluye una Unidad de Procesamiento Gráfico o GPU con la cual procesamos y analizamos las imágenes. Posteriormente, se envían comandos al controlador de motores para la velocidad linear y angular. Al combinar todo lo anterior con el algoritmo PURE PERSUIT, se crearon nodos utilizando ROS para mantener un control autónomo de seguimiento de línea, reconocimiento de señalamientos y semáforos durante el recorrido.

2.1 Aspectos éticos

Un gran avance en vehículos autónomos para la población en general traería una disminución de hasta 300 mil vidas cada 10 años solo en los Estados Unidos. Una investigación del MIT usando el proyecto MORAL MACHINE, buscando un consenso moral para mejorar el comportamiento podría decidir la vida de muchas personas en el futuro. Estás elecciones captan factores como edad, género, cantidad, profesión, peatones o ocupantes, entre otras. Los resultados arrojaron que en edad se prescindía más de la vida mientras mas edad la persona, sin embargo, en cuestiones mas complejas como paso ilegal o cruce legal estaba en 50%. Finalmente se tienen diferentes posturas por la diversidad de valores y reglas morales, es decir, no será posible encontrar un consenso moral. Por lo

que debemos seguir investigando para lograr programar moralmente un vehículo autónomo, con el que la gran mayoría de la población este de acuerdo. (Xakata 2019)

3 Análisis ingenieril

Lo requerimientos para el vehículo autónomo se pueden observar en la Table 1 donde encontramos algoritmos, SOFTWARE y HARDWARE necesario que se explica a continuación.

Table 1: Requerimientos

Requerimiento	Descripción
Funcional	PUZZLEBOT sigue la línea
Desempeño	PUZZLEBOT se detiene completamente con semáforo en rojo
Desempeño	PUZZLEBOT reanuda movimiento con semáforo en verde
Restricciones de diseño	PUZZLEBOT respeta los señalamientos viales
Funcional	PUZZLEBOT navega exitosamente el cruce
Funcional	PUZZLEBOT navega exitosamente toda la pista

4 Marco teórico

4.1 Cinemática y dinámica

La cinemática en un robot estudia sus movimientos, analizando la posición, velocidad y aceleración de cada elemento sin considerar la fuerza que causa el movimiento. La relación entre fuerzas de movimiento y torsión constituyen la dinámica. Transformación de matrices, álgebra vectorial y transformación de coordenadas nos facilitan los cálculos para llegar a una posición local, desde una posición global. La matriz de transformación homogénea nos representa la posición y orientación relativa. Al tener un robot diferencial, se toman en cuenta para calcular el movimiento las velocidades angulares de la Ecuación 1 de cada rueda o lineal que es el promedio de las velocidades como se muestra en la Ecuación 2. (Vargas 2020)

$$W = R * (\frac{V_R + V_L}{L}) \tag{1}$$

Donde W es la velocidad angular, L la distancia entre ruedas, R es el radio de las ruedas, V_R velocidad angular rueda derecha, V_L velocidad angular rueda izquierda.

$$V = R * (\frac{V_R + V_L}{2}) \tag{2}$$

Por consiguiente al integrar la velocidad obtenemos la posición, con lo cual tenemos el sistema inercial del robot móvil en la Ecuación 3.

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos\theta & 0 \\ sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R * (\frac{V_R + V_L}{2}) \\ R * (\frac{V_R - V_L}{L}) \end{bmatrix} \tag{3}$$

4.2 ROS

En la robótica se ha concebido un meta-sistema operativo llamado ROS o ROBOT OPERATING SYSTEM, dado al gran avance del mundo creaba enormes retos para el progreso de software debido al aumento de la comunidad. ROS es de código abierto, con abstracción de hardware, un control amigable de dispositivos, comunicación de mensajes entre procesos junto con el manejo de nodos. Por otra parte, tiene librerías y herramientas para obtener, crear y ejecutar entre varias computadoras. Tomando en cuenta un robot con sensores y actuadores, la transmisión entre procesos se dificulta, incluso entre enjambres de robots,por lo tanto, la productividad de ROS se comprende al especificar el funcionamiento de los nodos en nuestro sistema. (Okane 2014)

4.3 Odometría

En el cálculo de la posición del robot diferencial en la navegación, para ello utilizamos la información sobre el giro de las ruedas evaluando el cambio de la posición respecto al tiempo a partir de su posición inicial. Apoyado en la transformación de las revoluciones a desplazamiento lineal relacionado a la superficie, a pesar de que, se acumula el error debemos tener referencias con sensores para mantener una posición segura. (Ramirez 2019)

4.4 Visión

La visión computacional estudia los procesos de los organismos para reconocer y localizarse en el ambiente, mientras los entendemos y construimos robots con una capacidad similar. El procesamiento de imágenes es un área importante de la visión, para mejorar la calidad para su análisis e incluso utilizar máscaras, cortes o detectores de líneas para solo obtener cierta información de la imagen. La biblioteca OPEN SOURCE COMPUTER VISION LIBRARY, mejor conocido como OpenCV para visión computacional y aprendizaje automático de código abierto. El objetivo es proveer un soporte común para aplicaciones de visión y agilizar el uso de la percepción del robot en los productos comerciales. Con cerca de 2500 algoritmos optimizados, que abarcan algoritmos clásicos y de nueva generación para detectar, reconocer caras, identificar objetos y la identificación de colores. (S. Gómez 2012)

4.5 Algoritmo de seguimiento

El algoritmo PP o PURE PERSUIT ha sido utilizado en varios Institutos de Robótica en diferentes condiciones y se encontró que es una gran herramienta de seguimiento para propósito general. Este funciona calculando la curvatura que va a mover el vehículo desde su posición actual hasta un punto meta, se tiene una distancia frente al vehículo llamada LOCKAHEAD DISTANCE en la Ecuación 4. El auto está "persiguiendo" un punto dentro del camino trazado, una distancia delante de el auto. El arco que une el punto actual en la figura 1, con el punto meta encontrado que está dentro del camino trazado dentro de la distancia LOOKAHEAD. Las ecuaciones 1 se utilizan para calcular la curvatura del arco. (Coulter 1992)

$$d = r - x \longrightarrow 2 * r * x = l^2 \longrightarrow r = \frac{l^2}{2 * x} \longrightarrow \gamma = \frac{2 * x}{l^2} \tag{4}$$

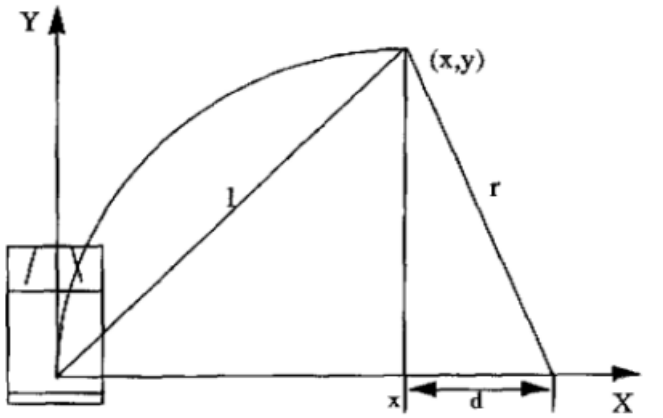

Figure 1. Geometry of the Algorithm.

Figure 1: Geometría del algoritmo

4.6 Control difuso

El control difuso o FUZZY CONTROL tiene origen en el sistema basado en reglas para tomar decisiones o lógica difusa para la toma de decisiones. Este tipo de controlador toma las variables de entrada, se dividen en grupos según el criterio del programador; estos criterios deben tener un pico en su valor más alto o varios valores pero siempre los máximos deben tener el mismo nivel en cada conjunto. Las reglas que se usan para los conjuntos se basan más en sentido común que de ecuaciones y modelos. (C. Gómez 2016)

5 Explicación del sistema

5.1 Procedimiento

Para poder generar el algoritmo que realice la conducción autónoma, se realizaron diferentes tipos de procesamiento, que a su vez fueron dividíos en tópicos al implementarlos en ROS, sin embargo, en esta subsección nos enfocaremos en el procesamiento de datos realizado, y en la siguiente subsección se hablará de su implementación en ROS.

A continuación se explica el procesamiento necesario para cumplir con todos los requerimientos de la Tabla 1.

5.1.1 Requerimiento: PUZZLEBOT sigue la línea

Para que nuestro robot diferencial siguiera la línea, se usó una cámara del modelo RASPBERRY PI CAMERA MODULE V2. Esta cámara es el sensor utilizado para detectar la línea.

Debido al uso de imágenes se usó la librería OPENCV para realizar este procesamiento, el cual consiste de:

- Reducir el tamaño de la imagen a un tercio.
- Convertir la imagen a escala de grises.
- Aplicar un difuminado de tipo Gaussiano.
- Recortar la imagen para solo mantener la parte inferior central de ella.
- Aplicar un umbral, para generar una imagen binaria.
- Aplicarle un algoritmo de esqueletización a la imagen binaria.
- Usar el algoritmo HOUGHLINESP para obtener las líneas presentes en el esqueleto de la imagen.
- Obtener la línea más grande.
- Obtener el punto de la línea más grande que esté más lejano de la parte inferior media de la imagen.
- Obtener la distancia en PIXELES a la que se encuentra este punto del punto medio de la imagen (horizontalmente).
- Multiplicar este error por una velocidad angular máxima para obtener la velocidad angular de nuestro robot.

5.1.2 Requerimientos: PUZZLEBOT se detiene completamente con un semáforo rojo y PUZZLEBOT reanuda movimiento con semáforo en verde

Para cumplir este requerimiento, se hace uso nuevamente de la cámara para hacer detección de contornos y de colores, usando OPENCV, con los pasos mostrados a continuación:

- Reducir el tamaño de la imagen a un tercio.
- Recortar la imagen dependiendo si el semáforo se encuentra a la izquierda de la imagen o enfrente.
- Convertir la imagen a escala de grises.
- Aplicar un umbral, para generar una imagen binaria.
- Usar el algoritmo FINDCONTOURS para obtener los contornos de las figuras vistas en el umbral.
- Usar el algoritmo BOUNDINGRECT para obtener un rectángulo que contenga cada uno de los contornos encontrados.
- Obtener la densidad de color verde, rojo y amarillo, dentro de cada uno de los rectángulos encontrados (Usando la imagen en formato HSV y el algoritmo INRANGE).
- Obtener el rectángulo más grande.
- Determinar si es una luz verde con la posición del rectángulo y la densidad de color verde.
- Repetir el paso anterior con el color rojo y amarillo

5.1.3 Requerimiento: PUZZLEBOT respeta los señalamientos de tránsito

Este requerimiento hace uso de la cámara para realizar detección de descriptores y emparejamiento usando OPENCV, con los pasos mostrados a continuación.

- Obtener los descriptores de las plantillas de las señales de tránsito usando el algoritmo ORB.DETECTANDCOMPUTE.
- Obtener los descriptores de la imagen de la cámara usando el algoritmo ORB.DETECTANDCOMPUTE.
- Hacer un emparejamiento entre los descriptores de las señales de tránsito y la imagen de la cámara usando el algoritmo FLANN.KNNMATCH.
- Usando la distancia entre los vecinos obtenidas por el paso anterior, determinar si es un buen emparejamiento o no.

- Determinar la señal que se esta viendo si es la que tiene la mayor cantidad de buenos emparejamientos.
- Si ninguna de las plantillas tuvo la suficiente cantidad de buenos emparejamientos, determinar que no se está viendo ninguna de las plantillas.

Por otro lado, otro de los señalamiento viales presentes se tratan de los cruces peatonales, para hacer uso de estos se usó la cámara para realizar detección de contornos y posiciones de los mismos usando OPENCV, como se muestra a continuación:

- Reducir el tamaño de la imagen a un tercio.
- Convertir la imagen a escala de grises.
- Aplicar un difuminado de tipo Gaussiano.
- Recortar la imagen para solo mantener la parte inferior.
- Aplicar un umbral, para generar una imagen binaria.
- Obtener los bordes de los objetos vistos en el umbral usando el algoritmo CANNY.
- Obtener las líneas vistas en los bordes, usando el algoritmo HOUGHLINESP, para determinar si se está viendo la linea central o una mayor cantidad de objetos
- Si se detectó una cantidad de líneas considerables, se usa el algoritmo FINDCONTOURS para obtener los contornos de la imagen tras pasar por el umbral.
- Usar el algoritmo BOUNDINGRECT para obtener un rectángulo que contenga cada uno de los contornos encontrados.
- Obtener la desviación estándar de la coordenada vertical del centro de los rectángulos encontrados.
- Si la desviación estándar es menor a un umbral y hay una suficiente cantidad de rectángulos encontrados, se determina que se está viendo un cruce peatonal.

5.1.4 Requerimiento: PUZZLEBOT navega exitosamente el cruce.

Para cumplir con este requerimiento se usaron dos algoritmos, ambos usan la odometría y puntos objetivos para determinar el comando de velocidad que se le debe enviar a nuestro PUZZLEBOT.

El primero de estos algoritmos es PURE PURSUIT, el cual funciona siguiendo los pasos mostrados a continuación:

- Generar una trayectoria en escuadra hasta al punto objetivo. La distancia entre los puntos de la trayectoria debe ser de 10 cm.
- Obtener el punto más cercano a nuestro vehículo que se encuentre frente a nosotros en la trayectoria.
- Rotar el punto visto desde el sistema global al sistema de referencia del robot.
- Calcular la velocidad angular del robot usando (4).
- Enviar una velocidad lineal constante.

Nuestro otro algoritmo utiliza Lógica difusa para determinar el comando de velocidad que se le envía al PUZZLEBOT, siguiendo los siguientes pasos.

- Determinar las funciones de membresía de los conjuntos de nuestras entradas y salida, las cuales se pueden observar en la Table 2.

Table 2: Funciones de membresía para cada una de las entradas y salidas

Entrada/Salida	Nombre de la variable	Nombre del conjunto	Tipo de función	Rango
Entrada	Distancia lineal al objetivo	Distancia corta (d0)	Gaussiana	$[0, 0.5]m$
		Distancia semi-corta (d1)	Gaussiana	
		Distancia media (d2)	Gaussiana	
		Distancia semi-larga (d3)	Gaussiana	
		Distancia larga (d4)	Gaussiana	
Entrada	Diferencia angular al objetivo	Diferencia negativa alta (a0)	Gaussiana	$[-180, 180]$
		Diferencia negativa media (a1)	Gaussiana	
		Diferencia baja (a2)	Gaussiana	
		Diferencia positiva media (a3)	Gaussiana	
		Diferencia positiva alta (a4)	Gaussiana	
Salida	Velocidad lineal	Velocidad baja (D0)	Trapezoidal	$[0, 0.1]m/s$
		Velocidad semi-baja (D1)	Triangular	
		Velocidad media (D2)	Gaussiana	
		Velocidad semi-alta (D3)	Triangular	
		Velocidad alta (D4)	Trapezoidal	
Salida	Velocidad angular	Velocidad negativa alta (A0)	Triangular	$[-0.3, 0.3]rad/s$
		Velocidad negativa media (A1)	Triangular	
		Velocidad baja (A2)	Triangular	
		Velocidad positiva media (A3)	Triangular	
		Velocidad positiva alta (A4)	Triangular	

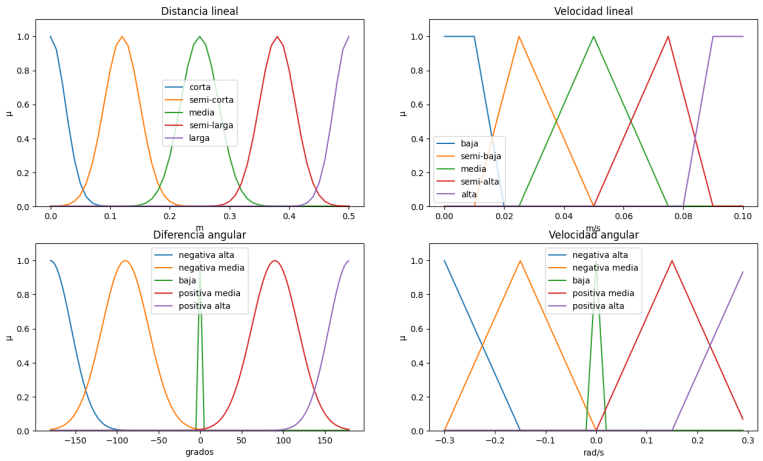

Figure 2: Funciones de membresía de las entradas y salidas

.

- Definir el sistema de inferencia de salidas, en la Tabla 3 podemos observar el sistema de inferencia para la velocidad lineal y en la Tabla 4 podemos observar el mapa para la velocidad angular. En la Fig. 2 se puede observar una gráfica de los conjuntos.

Table 3: Mapa de la salida de velocidad lineal vs las entradas

	a0	a1	a2	a3	a4
d0	D0	D0	D0	D0	D0
d1	D0	D1	D1	D1	D0
d2	D1	D2	D2	D2	D1
d3	D2	D3	D3	D3	D2
d4	D2	D3	D4	D3	D2

Table 4: Mapa de la salida de velocidad angular vs las entradas

	a0	a1	a2	a3	a4
d0	A0	A1	A2	A3	A4
d1	A0	A1	A2	A3	A4
d2	A1	A1	A2	A3	A3
d3	A1	A1	A2	A3	A3
d4	A1	A1	A2	A3	A3

- Utilizar el método de promediado de peso (5) para calcular la salida discreta de nuestro controlador.

$$z* = \frac{\sum \mu_c(\bar{z}) - \bar{z}}{\sum \mu_c(\bar{z})} \tag{5}$$

- Calcular las superficies de control vistas en la Fig. 3 y la Fig. 4.

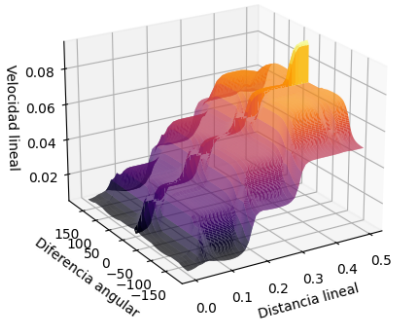

Figure 3: Superficie de control de la velocidad lineal

.

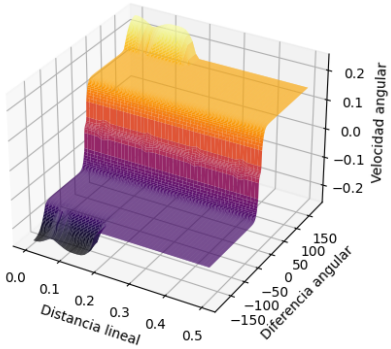

Figure 4: Superficie de control de la velocidad angular

.

- Obtener el valor de la superficie de control de los valores actuales del robot.

5.1.5 Requerimiento: PUZZLEBOT navega exitosamente toda la pista

Para cumplir con este requerimiento se utilizó un controlador central que recopila la información de todos los requerimientos anteriores.

Debido a la complejidad de este algoritmo, se representa su funcionamiento con un diagrama de flujo, visto en la Fig. 5.

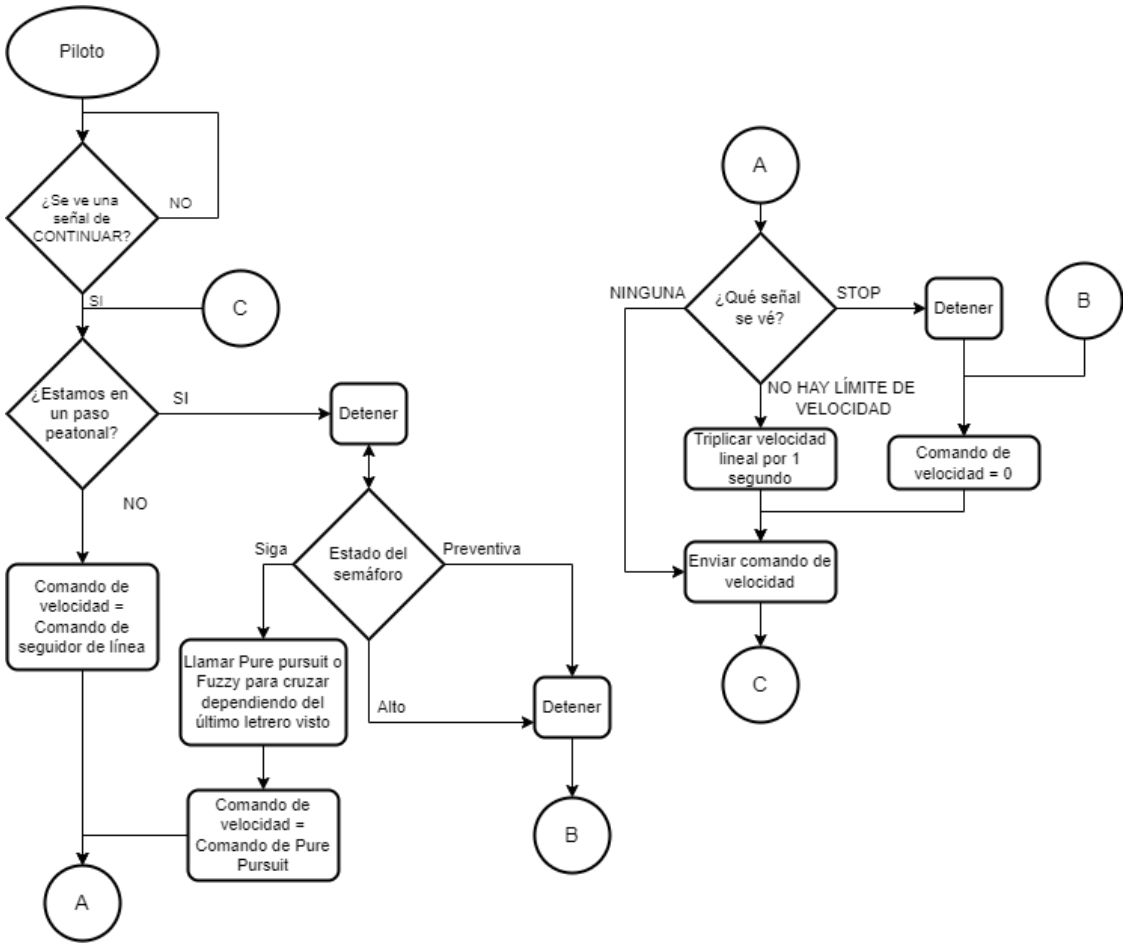

Figure 5: Diagrama de flujo del controlador principal de nuestro algoritmo

.

5.2 Nodos y tópicos

Para implementar este algoritmo, se utilizaron los nodos y tópicos vistos en 6.

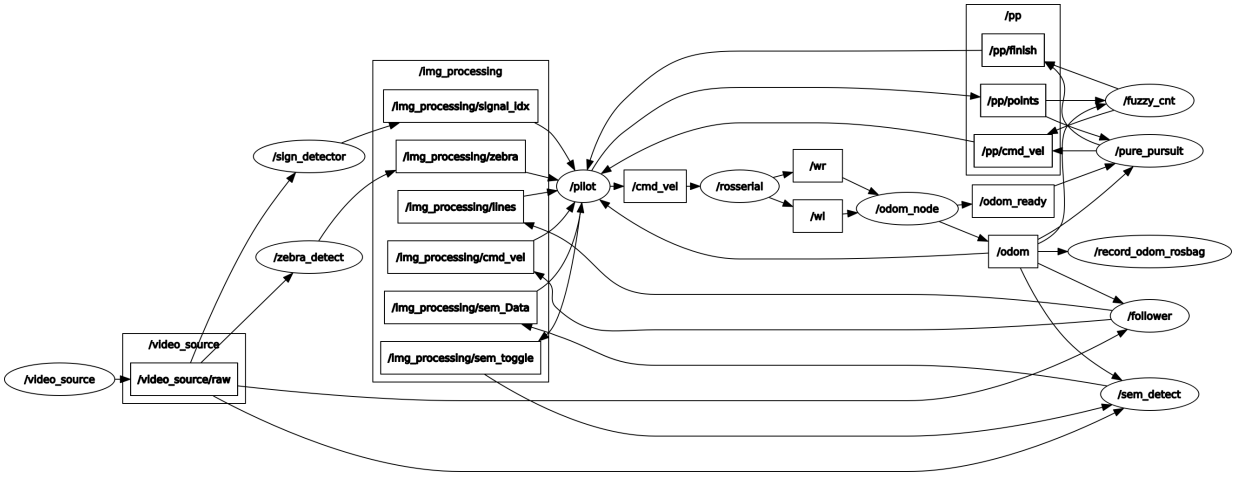

Figure 6: Nodos y tópicos utilizados para la implementación

.

En la Fig. 6 ver podemos los siguientes nodos relevantes:

- $/video\_source$

  Este nodo se encarga de recuperar y enviar la imagen capturada por la cámara integrada al PUZZLEBOT. Publica al tópico $/video\_source/raw$.

- $/rosserial$

  Este nodo se encarga del control del PUZZLEBOT. Se usa este nodo para comunicar nuestra tarjeta JETSON con la tarjeta micro controladora que controla los motores. Este nodo se encarga de recibir los comandos destinados a los motores y de calcular la velocidad angular de ambas ruedas y publicarlas.

- $/odom\_node$

  Este nodo fue el creado para calcular la odemetría usando las velocidades angulares de las ruedas. Está subscrito a los tópicos de las velocidades angulares ($/wr$ y $/wl$) y publica la odometría en $/odom$.

- $/follower$

  Este tópico obtiene el procesamiento de la imagen necesario para poder seguir la línea central para controlar el PUZZLEBOT, usando el procedimiento de la Sección 5.1.1 Se subscribe al tópico $/video\_source/raw$ y publica al tópico $/img\_processing/cmd\_vel$ para controlar el movimiento.

- $/sign\_detector$

Este tópico obtiene el procesamiento de la imagen necesario para poder detectar los señalamientos de tránsito que ve el PUZZLEBOT, usando el procedimiento de la primera parte de la Sección 5.1.3. Se subscribe al tópico $/video\_source/raw$ y publica al tópico $/img\_processing/signal\_idx$ para enviarle al piloto.

- $/sem\_detect$

  Este tópico obtiene el procesamiento de la imagen necesario para poder detectar la luz del semáforo que ve el PUZZLEBOT, usando el procedimiento de la Sección 5.1.2. Se subscribe al tópico $/video\_source/raw$ y publica al tópico $/img\_processing/sem\_idx$ para enviarle al piloto.

- $/zebra\_detect$

  Este tópico obtiene el procesamiento de la imagen necesario para poder detectar el paso peatonal que ve el PUZZLEBOT, usando el procedimiento de la segunda parte de la Sección 5.1.3. Se subscribe al tópico $/video\_source/raw$ y publica al tópico $/img\_processing/zebra$ para enviarle al piloto.

- $/pure\_pursuit$

  Este tópico ejecuta el algoritmo PURE PURSUIT de la primera parte de la Sección 5.1.4, para enviarle el comando de velocidad al piloto. Se subscribe al tópico $/odom$ y publica al tópico $/pp/cmd\_vel$.

- $/fuzzy\_cnt$

  Este tópico ejecuta el algoritmo del controlador difuso de la segunda parte de la Sección 5.1.4, para enviarle el comando de velocidad al piloto. Se subscribe al tópico $/odom$ y publica al tópico $/pp/cmd\_vel$.

- $/pilot$

  Este tópico ejecuta el algoritmo visto en la Sección 5.1.5, para enviarle el comando de velocidad al PUZZLEBOT. Se subscribe los tópicos de salida de los demás tópicos y publica al tópico $/cmd\_vel$.

De los tópicos que se observan en la Fig. 6 se enlistan a continuación los más importantes:

- $/video\_source/raw$

  En este tópico se publica la imagen de la cámara del PUZZLEBOT.

  Este mensaje es de tipo: $sensor\_msgs : Image$

- $/wl$

  En este tópico se publica la velocidad angular de la llanta izquierda.

  Este mensaje es de tipo: $std\_msgs : Float32$

- $/wr$

  En este tópico se publica la velocidad angular de la llanta derecha.

  Este mensaje es de tipo: $std\_msgs : Float32$

- $/odom$

  En este tópico se publica las coordenadas y el ángulo calculado por odometría por nuestro nodo $/odom_node$.

  Este mensaje es de tipo: $std\_msgs : Pose2D$

- $/img\_processing/odom_ready$

  En este tópico se publica cuando la odometría ya comenzó a calcularse.

  Este mensaje es de tipo $std\_msgs : UInt8$.

- $/cmd\_vel$

  En este tópico se publican los comandos para controlar la velocidad angular y linear del PUZZLEBOT.

  Este mensaje es de tipo $geometry\_msgs/Twist$.

- $/pp/cmd\_vel$

  En este tópico se publican los comandos para controlar la velocidad angular y linear del PUZZLEBOT, según los algoritmos de PURE PURSUIT y el controlador difuso.

  Este mensaje es de tipo $geometry\_msgs/Twist$.

- $/pp/points$

  En este tópico se publican el punto objetivo al que se quiere que lleven los algoritmos de PURE PURSUIT y el controlador difuso.

  Este mensaje es de tipo $std\_msgs : Float32MultyArray$.

- $/pp/finish$

  En este tópico se publican un mensaje que indica que los algoritmos de PURE PURSUIT o el controlador difuso han llegado a su punto objetivo.

  Este mensaje es de tipo $std\_msgs : UInt8$.

- $/img\_processing/cmd\_vel$

  En este tópico se publican los comandos para controlar la velocidad angular y linear del PUZZLEBOT, según el nodo de $/follower$.

  Este mensaje es de tipo $geometry\_msgs/Twist$.

- $/img\_processing/signal\_idx$

  En este tópico se publica el índice de la señal de tránsito observada, según el nodo de $/sign\_detector$.

  Este mensaje es de tipo $std\_msgs : UInt8$.

- $/img\_processing/zebra$

  En este tópico se publica si se observa o no un cruce peatonal, según el nodo de $/zebra\_detect$.

  Este mensaje es de tipo $std\_msgs : UInt8$.

- $/img\_processing/lines$

  En este tópico se publica el número de líneas encontradas por el nodo de $/follower$.

  Este mensaje es de tipo $std\_msgs : UInt8$.

- $/img\_processing/sem\_Data$

  En este tópico se publica el índice de la luz del semáforo observada, según el nodo de $/sem\_detect$.

  Este mensaje es de tipo $std\_msgs : UInt8$.

- $/img\_processing/sem\_toggle$

  En este tópico se publica el índice de posición del semáforo (enfrente a la derecha o directamente enfrente) para que el nodo de $/sem\_detect$ recorte la imagen correcta.

  Este mensaje es de tipo $std\_msgs : UInt8$.

6 Resultados

Al iniciar el recorrido el Puzzlebot espera hasta ver la señal de continue, esta puede verse iluminada en la Figura 7, en ese momento se inicia el seguimiento de la línea central del circuito.

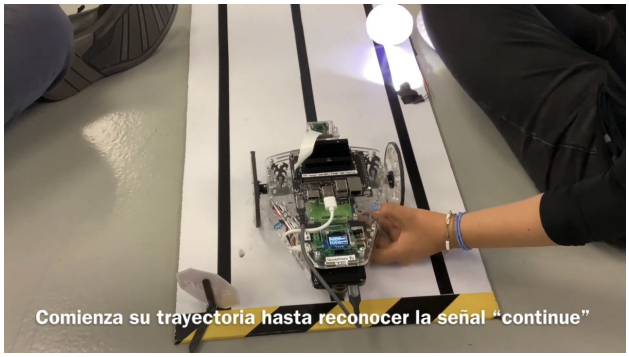

Figure 7: Señal "Continue"

.

Al llegar al paso peatonal el robot se detiene hasta que el semáforo cambie a verde, en la Figura 8 se observa cuando está detenido con el semáforo en rojo.

Figure 8: Semáforo

.

Posteriormente realiza el cruce y continúa con el seguimiento de línea, al dar la primera vuelta a la izquierda se detecta la señal de NO SPEED LIMIT como se observa en la Figura 9 y el vehículo acelera el triple de su velocidad linear.

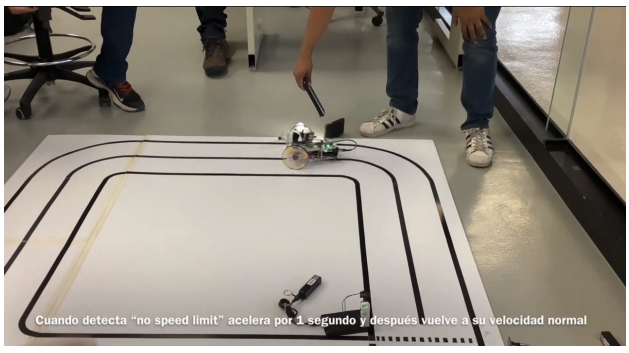

Figure 9: Señal "No Speed Limit"

.

Al completar la vuelta al circuito se llega al segundo paso peatonal, donde se detiene como se ve en la Figura 10 y debe ver la señal de vuelta a la derecha y esperar que el semáforo cambie a verde.

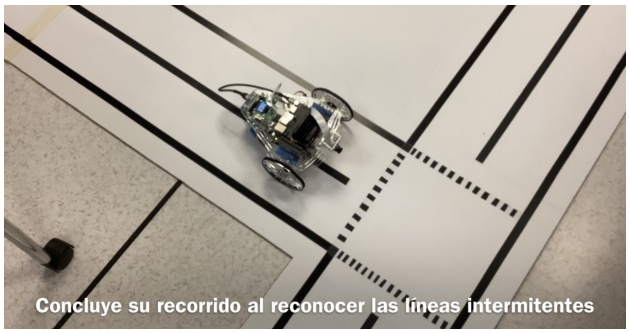

Figure 10: Líneas de paso peatonal

.

Finalmente sigue la línea hasta encontrar la señal STOP como se muestra en la Figura 11 y en ese momento se detiene completamente.

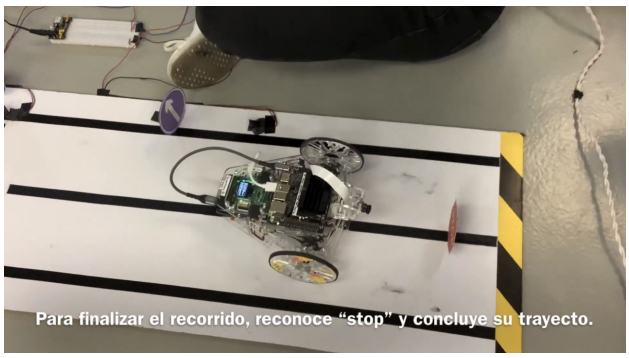

Figure 11: Señal "Stop"

.

7 Conclusiones

7.1 Diego Reyna:

En este proyecto pude observar la importancia que tiene la tolerancia al error de nuestro sistema, ya que la iluminación afectó gravemente el resultado obtenido, debido a la dependencia de la cámara. De igual forma, fue interesante observar como se pueden juntar diferentes tipos de algoritmos para generar un resultado

funcional, esto es interesante, ya que fue un reto en el que debimos implementar los conocimientos obtenidos en cada uno de los módulos.

Sin embargo, me quedo con ganas de seguir refinando el algoritmo, ya que perdimos mucho tiempo de desarrollo corrigiendo problemas con la cámara y con el algoritmo para seguir la línea. Esto nos redujo tiempo para mejorar el funcionamiento del detector de señales o del controlador difuso, e incluso, con problemas de la odometría.

7.2 Samantha Barrón:

La implementación del funcionamiento en nuestro reto final fue compleja, al mismo tiempo se trabajo progresivamente durante las diez semanas. Generamos distintos nodos de trabajo para que todas las señales necesarias sean recibidas en un nodo pilo, este es el principal que tiene todo el control de conducción en el recorrido. Durante la implementación presentamos distintos retos con el entorno, la iluminación en especial. La cámara con poco definición y ligera decoloración. Aún así se encontraron soluciones oportunas con el software. Es satisfactorio ver el trabajo final y ver el uso de estas herramientas para un futuro laboral. Al igual estoy agradecida con el equipo de trabajo que formamos en el que todos nos dedicamos al máximo para entregar un trabajo de calidad.

7.3 Jorge Hoyo:

Durante el reto hubo muchos momentos difíciles como el uso de Pure Persuit ya que teníamos muchos problemas con la odometría, también al reconocer la línea nuestro robot oscilaba mucho ya que el error era muy grande y el control no era suficiente. Se agregó un filtro mediana para mejorar dicho control, por otro lado, agregar el control difuso para la última vuelta a la derecha nos ayudó a que funcionará una de cada 5 veces, ya que sin esto fallaba 4 de 5 veces. Finalmente, la parte que mas me agrado fue la facilidad de crear tópicos y nodos para poder partir los códigos y tener mejor trazabilidad de lo que se realizaba para cada requerimiento durante el recorrido.

8 Repositorio

https://github.com/Equipo1IRSTecCCM/final-project-IRI

References

[1] Thinking Big. *Ventajas coches autónomos*. 2020. URL: https://blogthinkbig.com/ventajas-coches-autonomos.

[2] Coulter. *Implementation of Pure Pursuit*. 1992. URL: https://www.semanticscholar.org/paper/ImplementationofthePurePursuitPathTrackingCoulter/ee756e53b6a68cb2e7a2e5d50.

[3] C. Gómez. *Fuzzy Control*. 2016. URL: https://www.frba.utn.edu.ar/wp-content/uploads/2021/02/Fuzzy-Control.pdf.

[4] Sucar Gómez. *Visión*. 2012. URL: https://ccc.inaoep.mx/~esucar/Libros/vision-sucar-gomez.pdf.

[5] Okane. *Introduction to ROS*. 2014. URL: http://lsi.vc.ehu.es/pablogn/invest/IntroductionToROS.pdf.

[6] K. Ramirez. *Odometría*. 2019. URL: http://www.kramirez.net/Robotica/Material/Presentaciones/Odometria.pdf.

[7] L Vargas. *Diseño de un móvil con tracción diferencial*. 2020. URL: https://cici.unillanos.edu.co/media2020/memorias/CICI_2020_paper_125.pdf.

[8] Xakata. *MAyor estudio sobre ética de coches autónomos*. 2019. URL: https://www.xataka.com/automovil/el-mayor-estudio-sobre-la-etica-de-los-coches-autonomos-trae-malas-noticias-para-los-ancianos-e-incertidumbre-para-la-industria.