Final Challenge

Ramírez, Hortencia; Salcedo, Andrea; Buenaventura, José Gustavo; León, José Antonio
Gaeta, Carlos; Antunez, Fernando
School of Engineering and Sciences
Tec de Monterrey
Monterrey, Nuevo Léón, México
amunoz@tec.mx

I. Abstract

La robótica y tecnologías de la información han avanzado exponencialmente en las últimas décadas, lo cuál nos ha permitido montar sistemas complejos en robots cada vez más pequeños. Uno de estos sistemas es el Puzzlebot de Manchester Robotics, que cuenta con una tarjeta de desarrollo NVIDIA Jetson Nano que nos permite correr ROS, Robot Operating System, y modelos de Machine Learning. La integración de ambos nos permite llegar a simular un vehículo autónomo capaz de detectar señales de tránsito, semáforos y mantener una trayectoria dentro de las calles del Puzzletrack.

II. Introducción

La robótica inteligente es el resultado de la aplicación de conceptos, principios y técnicas de la inteligencia artificial tales como aprendizaje computacional, control inteligente y visión computacional, todo esto aplicado nos ayuda a brindar autonomía a dispositivos robóticos y permitirles mejorar con la experiencia. Estos sistemas son capaces de procesar información de su entorno y determinar nuevas estrategias que les permitan alcanzar los objetivos para los cuales fueron desarrollados. Una de las aplicaciones de la robótica inteligente más interesantes y que ha llamado la atención en la última década es la conducción autónoma. Los vehículos autónomos son capaces de asistir en la conducción o hasta manejarse por sí mismos de manera segura en entornos con peatones y tráfico vehicular. Para ello, cuentan con cámaras y sensores que les permiten identificar en tiempo real a personas, objetos, vehículos, señales de tránsito y caminos, con lo cual toman decisiones en cuanto a la velocidad y la dirección que deben tomar para llegar a su destino. La aplicación de elementos de la inteligencia artificial, tales como visión, aprendizaje computacional y control inteligente, son clave para la generación de nuevas propuestas para la navegación autónoma que permitan eventualmente la transformación tecnológica de la movilidad en nuestras ciudades.

III. Descripción del reto

El reto de este bloque era construir un robot móvil que contará con un sistema de navegación y visión que le permitiera transitar autónomamente por una pista, siendo capaz de detectar cruces peatonales, señales de tránsito y semáforos y tomar decisiones de acuerdo a lo que está viendo. Para hacer esto utilizamos el framework ROS, el cual es un conjunto de herramientas y librerías que nos permiten construir aplicaciones robóticas de manera eficiente y sencilla creando una serie de nodos, cada uno encargándose de una parte del procesamiento del robot y siendo capaces de comunicarse entre ellos. También se nos brindo un kit de desarrollo de nvidia, el cual incluia la estructura y motores para mover el robot. El robot se controla a traves de una nvidia jetson nano la cual es capaz de realizar operaciones de inteligencia artificial con redes neuronales.

El socio formador nos dio una pista que habia que seguir, eligieron el recorrido y pusieron una seria de señales de transito y semaforos en ella. El robot debia avanzar hasta llegar al cruce peatonal y ver el semaforo, si esta en verde, el robot debía ver una señal indicando que debia girar a la izquierda y el robot debía girar. Después tenía que seguir la linea en la pista, si detectaba una señal de construcción debía disminuir su velocidad, después sigue el recorrido hasta llegar nuevamente al cruce peatonal, donde lee una señal, si dice que de vuelta la derecha debe hacer nuevamente el recorrido, si es una señal de seguir derecho, avanza hasta llegar a la señal de alto y se termina el recorrido.

IV. Metodología

Para desarrollar el reto del Puzzlebot, utilizamos diversas metodologías que vimos a través del curso de Implementación de Robótica Inteligente. La que más impacto tuvo en el reto fue la visión computacional. Las técnicas principales que empleamos fueron conversiones de color, filtros HSV, filtros por umbrales, erosión y dilatación de imágenes, desenfoque Gaussiano e identifiación de contornos. La otra técnica de mayor impacto fue el control PID, que lo implementamos para el movimiento de las llantas robot, empleando como señal de control la velocidad para las llantas. Finalmente, la otra técnica principal que empleamos fueron redes neuronales convolucionales profundas, en este caso utilizamos el modelo YOLOv5. El ajuste de parámetros lo realizamos utilizando prueba y error, y para entrenar el modelo de predicción para YOLO utilizamos la plataforma de Roboflow.

V. Desarrollo de la solución

El planteamiento de la propuesta para dar solución al reto fue a través de integrar el desarrollo de las actividades realizadas a lo largo de las sesiones impartidas por Manchester

Robotics, las cuales nos permitieron dar una solución mucho más robusta al reto planteado.
En la figura, se puede observar la integración y comunicación con cada uno de los nodos realizados haciendo uso del set de librerías de ROS.

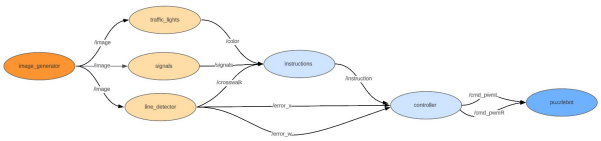

Fig. 1. Diagrama de integración de nodos

De manera incial se tiene un nodo denominado como *image generator*, encargado de abrir la cámara del puzzlebot. Dicho nodo se encarga de publicar el tópico *image* que comparte los frames obtenidos de la cámara hacia los nodos encargados de hacer los diferentes procesamientos de la imágen, como es la detección del semáforo, líneas y de las diferentes señales, este procesamiento se hace en los nodos: *traffic lights, signals y line detector*. La información que es obtenida del procesamiento de la imágen, como el color del semáforo, señal, presencia del crosswalk es publicada al nodo de *instructions* que es donde se concentra la máquina de estados del robot, es decir, a través del procesamiento de la información es como el robot decide que instrucción debe ejecutar, si es que tiene que reducir su velocidad, girar, parar completamente, entre otras.
Por otro lado, se encuentra el nodo de *controller*, que es el nodo qe esta conectado directamente al puzzlebot que funciona en conjunto con la instrucción recibida del nodo de *instructions* y con el de *line detector*, puesto que através de los tópicos de error x y error w, es el ajuste que realiza a la velocidad de cada uno de los motores para que el puzzlebot sea capaz de seguir la línea del camino.

Nodos

El desarrollo de la solución propuesta cuenta con un total de seis nodos que en conjunto permiten darle una solución completa y sobre todo robusta al reto, dichos nodos fueron realizados *from scratch*, es decir, desde cero, solo tomando como base el esquema que debe tener un programa para poder implementar las librerías de ROS.

El diagrama de funcionamiento del puzzlebot se encuentra en la sección de anexos en la figura 16.

A. Generador de imágen

Se realizó un nodo para configurar la cámara raspberry pi y de esta forma poder abrirla a partir del uso de las librería de OpenCv. Esta implementación fue de gran utilidad ya que consume mucho menos recursos comparado a si lo hubieramos hecho a través del paquete *deep learning* ofrecido por Manchester Robotics.

```python
def gstreamer_pipeline(
    sensor_id=0,
    capture_width=1920,
    capture_height=1080,
    display_width=960,
    display_height=540,
    framerate=30,
    flip_method=0,
):
    return (
        "nvarguscamerasrc sensor-id=%d !"
        "video/x-raw(memory:NVMM), width=(int)%d, height=(int)%d, framerate=(fraction)%d/1 ! "
        "nvvidconv flip-method=%d ! "
        "video/x-raw, width=(int)%d, height=(int)%d, format=(string)BGRx ! "
        "videoconvert ! "
        "video/x-raw, format=(string)BGR ! appsink"
        % (
            sensor_id,
            capture_width,
            capture_height,
            framerate,
            flip_method,
            display_width,
            display_height,
        )
    )
```

Fig. 2. Configuración de la cámara

```python
dispW = 320
dispH = 240
flip = 2
camSet='nvarguscamerasrc !  video/x-raw(memory:NVMM), width=3264, height=2464, format=NV12, framerate=21/1 ! nvvidconv

cap = cv2.VideoCapture(gstreamer_pipeline(flip_method=0), cv2.CAP_GSTREAMER)
```

Fig. 3. Configuración de la cámara

Como resultado de la configuración de la cámara y su inicialización haciendo uso de la librería de openCv, el nodo genera el tópico **image** al que otros nodos encargados de hacer el procesamiento de la imágen se suscriben. Más adelante se detallará el procesamiento que hace cada uno de ellos para realizar el debido procesamiento requerido.

B. Detección del semáforo

Por otro lado, para dar solución a la parte de detección del colores del semáforo se realizó un nodo que con el uso de las librerías que ofrece open Cv es capaz de hacer la detección de colores del semáforo e indicar en que color se encuentra en caso de encontrarse ante su presencia.
Este nodo se encarga de recibir la imágen y convertirla en formato hsv, para poder hacer el procesamiento de colores con las diferentes máscaras de cada uno de los colores, rojo, amarillo y verde. Para la obtención de los colores, la configuración se hizo de forma manual, implementando *trackbars* que dependiendo de la luz del día podiamos ajustar los valores para que el semáforo pueda ser detectado en los diferentes horarios del día bajo los cuales estaba expuesta la pista.

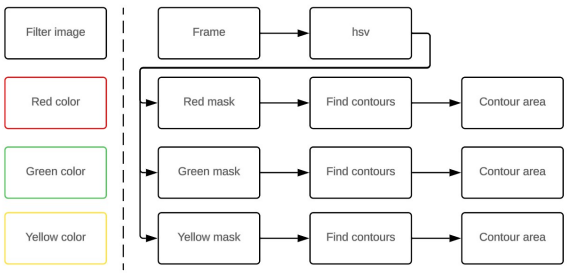

Fig. 4. Diagrama de funcionamiento para detección del semáforo

El resultado de aplicar las máscaras de colores al frame, fueron imágenes que solo detectaban la presencia del color del semáforo, por lo que para que el algoritmo fuera capaz de decidir si existía la presencia de alguno de ellos, se aplicó la detección de contornos, así como también el cálculo del número de pixeles en los contronos, lo que facilito hacer la detección del color del semáforo y al mismo tiempo obtener el área de los contronos ayudó a determinar la distancia que había entre el semáforo y el puzzlebot.

```python
# Find red contours
cnts_r = cv2.findContours(maskRed, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cnts_r = cnts_r[0] if len(cnts_r) == 2 else cnts_r[1]

for c in cnts_r:
    area = cv2.contourArea(c)
    if area > area_threshold_red:
        pub_color.publish("red")
        print("red")
```

Fig. 5. Detección del color rojo en el semáforo

Esto último explicado, se observa la implementación realizada en la figura 5, la detección de contronos, así como filtar aquellos contronos en las que el área sea mayor a la deseada y evitar la detección de color rojo en donde debido a la luz sea una falsa detección.

C. Seguidor de línea

El procesamiento de la imágen para realizar el seguir de líneas se dividió en tres principales fases, el filtrado del ruido, la detección de líneas y encontrar el error de la posición del puzzlebot con respecto al punto deseado.

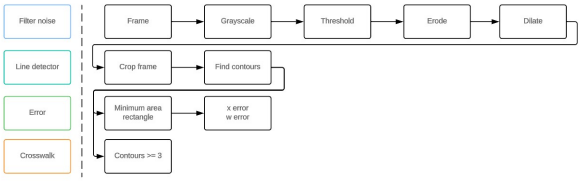

Fig. 6. Diagrama de funcionamiento para detección de líneas

Fases de filtrado

Filtrar ruido

Para filtrar el ruido que tenía la imágen original generada por el nodo de *image generator*, se aplicaron varias capas a la imágen para poder reducir el ruido que era capturado por la cámara del puzzelbot. Primero, la imágen fue transformada a escala de grises, posterior a esto se aplico un filtro threshold para clasificar los pixeles en escala de grises en blanco o negro y de esta forma solo detectar la línea. El resultado obtenido tras realizar este filtrado fue el mostrado a continuación.

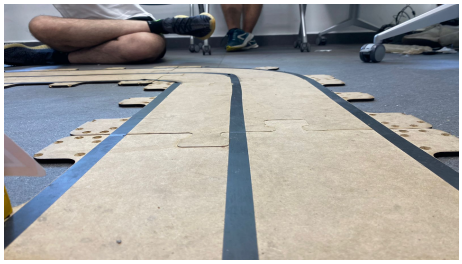

Fig. 7. Frame original detectado por la cámara

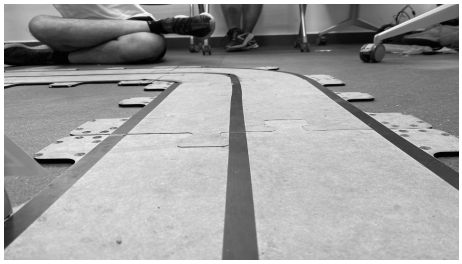

Fig. 8. Frame transformado a escala de grises

Tras aplicar el filtrado hasta obtener un frame con colores solo en blanco y negro, se aplicaron dos filtros más. Las funciones implementadas fueron las de erosionar y dilatar la imágen. La función de erosionar se encarga de reducir el área de alguna regiones de la imágen, permitiendo que aquellas regiones de puntos blancos que se encontraban fuera del rango de la línea detectada desaparezcan.

Fig. 9. Threshold

Fig. 10. Erosión

Por otro lado, se aplicó la dilatación para dilatar algunas secciones de la imágen, esto ayudo a que las partes de la línea que no estaban completamente, se rellenaran.

Fig. 11. Dilatación

Detección de la línea

Una vez que se realizó el filtrado de la imágen, se recorto el frame a solo la sección en donde se encuentra la línea central, apartir de esto se hizó la detección de contornos, lo cual daba como resultado la detección de la sección de la línea que es capturada por el puzzlebot.

Cálculo del error x y el error w

La detección de contronos contribuyó a que pudieramos implementar las funciones que ofrece la librería de open Cv de detectar el área mínima del rectángulo y con ello obtener la posición del centro del rectángulo, así como la inclinación que tiene. Con ello solo se ajustaron los valores para que el algorítmo fuese capaz de obtener el error en x y el error en w con respecto al centro de la imágen.

```
contours_blk, hier = cv2.findContours(th2,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
if len(contours_blk) > 0:
    blackbox = cv2.minAreaRect(np.array(contours_blk[0]))
    (x_min, y_min), (w_min, h_min), ang = blackbox
    if ang < -45 :
        ang = 90 + ang
    if w_min < h_min and ang > 0:
        ang = (90-ang)*-1
    if w_min > h_min and ang < 0:
        ang = 90 + ang
    setpoint = 210
    error = int(x_min - setpoint)
    ang = int(ang)
    box = cv2.boxPoints(blackbox)
    box = np.int0(box)
    cv2.drawContours(frame2,[box],0,(0,0,255),3)
    cv2.putText(frame2,str(ang),(10, 310), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
    cv2.putText(frame2,str(error),(10, 40), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2)
    cv2.line(frame2, (int(x_min),200 ), (int(x_min),250 ), (255,0,0),3)
    cv2.circle(frame2, (box[0][0], box[0][1]), 1, (255,0,255), 3)
    cv2.circle(frame2, (box[3][0], box[3][1]), 1, (0,255,255), 3)
    pub_x.publish(error)
    pub_w.publish(ang)
```

Fig. 12. Ajuste para obtención del error en x y error en w

El resultado obtenido fue el que se muestra en la figura siguiente.

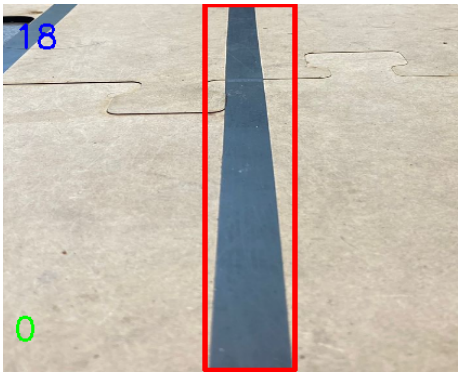

Fig. 13. Error x y error w

Por otro lado, este nodo en forma paralela, análiza el número de contronos que esta detectando, puesto que cuando detecte más de tres contornos sea capaz de mandar una señal que indique que ha encontrado un cruce peatonal. Dicha función funciona como bandera para indicar que es momento de ejecutar otro tipo de instrucciones.

D. Detección de señales

Para la detección de señales, utilizamos la imagen que manda el tópico images para en un nodo aparte pasarla por el modelo. Utilizando un dataset de más de 4000 imágenes, entrenamos un modelo de YOLOv5 utilizando los pesos ”small” para clasificar entre los 7 distintos señalamientos de tránsito. En el nodo de detección de señales, usamos PyTorch para meter la imagen por el modelo y que nos regrese una predicción de qué señal detectó, si es que detectó alguna, y la manda por el tópico de signal. Si no se detecta ningún señalamiento o si la certeza con la que lo detectó es menor a 0.8, se manda un código de error.

E. Instrucciones

Los nodos para realizar el procesamiento de la imágen estan conectados a un nodo que es encargado de ejecutar la máquina de estados bajo el cual funciona la lógica del puzzlebot. Dicho nodo se suscribe a los tópicos de *color, signals y crosswalk*. Recibe de cada uno de ellos el color del semáforo, el nombre de la señal que ha detectado y si hay o no presencia de un cruce peatonal.
El algoritmo funciona a través de publicar la dirección y velocidad a la que deben ir, siendo el semáforo quien indica la velocidad a la que debe ir el puzzlebot y la mayoría de las señales las que indican dirección.

Control de velocidad

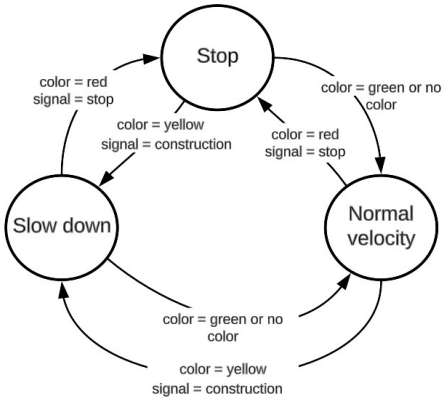

Fig. 14. Máquina de estados para el control de la velocidad

Control de dirección

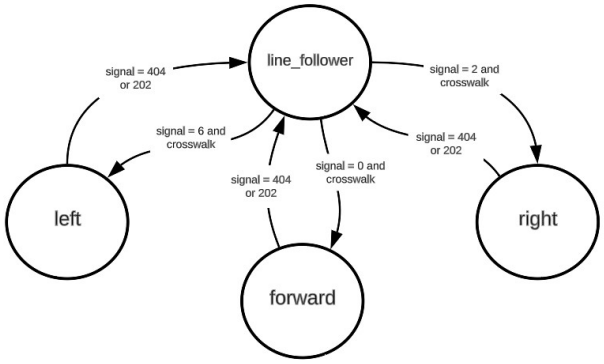

Fig. 15. Máquina de estados para el control de dirección

F. Controlador

Por otro lado, el nodo que esta conectado directamente a los motores del puzzlebot es el nodo del controlador, dicho nodo se suscribe a los tópicos de *instruction*, *error x* y *error w*. Tras recibir dicha información el nodo es capaz de ajustar los valores del controlador p, haciendo uso del error x y del error w.

Controlador P del error en x

$$\text{wr}_x = 0.37 - kr_x \cdot error_x$$
$$wl_x = 0.37 - kl_x \cdot error_x$$

Controlador P del error en w

$$wr_w = -kr_w \cdot error_w$$
$$wl_w = kl_w \cdot error_w$$

Por último, para obtener la velocidad final a la que van a girar los motores del puzzlebot, se obtuvo el promedio de ambas velocidades.

$$\text{wr}_final = \frac{wr_x + wr_w}{2}$$

$$wl_final = \frac{wl_x + wl_w}{2}$$

La velocidad que es publicada a los motores es la resultante del proceso anterior, debido a que uno de los motores gira más rápido que uno, la velocidad resultante del motor izquierdo se multiplico por 0.9 para reducir el error generado por el giro.

Este nodo además de controlar el giro de los motores en la detección de líneas tiene funciones que permiten ejecutar las instrucciones cuando el robot debe girar a la derecha o a la izquierda y cuando este debe seguirse derecho. El algoritmo bajo el cual funcionan estas funciones es a través de una secuencia, en la que por tanto tiempo ejecutan el giro o instrucción deseada.

Por último, si recibe la instrucción de reducir velocidad se reducirá la velocidad a la cual se mueve el puzzlebot en un 10 por ciento.

En la sección de anexos, en la figura 17 se puede observar el diagrama de funcionamiento del controlador completo.

VI. Resultados y logros

Los logros obtenidos fue un robot capaz de realizar la trayectoria considerando las señales y semáforos que iba encontrando en su camino. Los resultados están disponibles en el link del video, así como también el código implementado se encuentra en la liga de github.

Video:
https://youtu.be/bmP49zlKMyg

GitHub:
https://github.com/Jgusbc/Implementacion-de-robotica-inteligente/tree/FinalChallenge

VII. Reflexiones

A continuación se presentan las reflexiones de cada uno de los integrantes del equipo a lo largo del desarrollo del proyecto.

Hortencia.

El desarrollo del proyecto considero que integro mcuhas disciplinas de la robótica, y sobre todo creo que su elaboración y desarrollo fue muy importante para que pudieramos integrar todo lo aprendido a lo largo de la unidad de formación. Este proyecto me gusto mucho ya que nos permitió integrar conceptos de visión, control, redes neuronales y sobre todo creo que el poder integrar todo nos ayudo a trabajar en un ambiente mucho más parecido con el que nos vamos a presentar mucho más a futuro, es decir, con proyectos que integren muchas disciplinas como fue el caso del puzzlebot. De manera particular, este proyecto me gusto mucho, porque al mimso tiempo que iba aprendiendo, también era divertido

implementar las soluciones e ideas que ibamos generando en equipo.

Andrea.

Durante el desarrollo de este proyecto logré desarrollar muchas habilidades de la robótica como lo es la implementación de visión computacional con ROS, de igual manera, logré darme cuenta que los métodos de visión computacional tradicionales suelen ser más útiles de implementar en ciertos escenarios ya que no usan tantos recursos y suelen ser muy exactos. Por otro lado, aprendí que debemos tomar en cuenta las limitaciones que tenemos para poder desarrollar nuestros proyectos con mayor eficacia y sin tener tantos problemas. Finalmente, comprendí que el trabajo en equipo es una parte clave para que el proyecto salga adelante de manera más sencilla.

Carlos.

La integración interdisciplinar que fue este proyecto me ayudó mucho a desarrollar y concretizar los aprendizajes que estuvimos viendo durante todo el semestre. Pero creo que el aprendizaje que más me llevo es el conocer el ambiente en donde vas a desplegar tu aplicación. Nos pasó que por asumir que la Jetson funcionaba similar a como funcionaba la computadora, no le dimos mucha importancia al probar cómo funcionaba YOLOv5, y lo intentamos hacer con 2 días de anticipación, lo cual no funcionó al máximo, pero nos dejó un gran aprendizaje.

Antonio.

Al desarrollar este proyecto pude desarrollar y aplicar distintas habilidades muy importantes en la robotica como el uso del framework ROS, el cual permitio hacer el desarrollo de este reto mas eficiente, tambien aprendí como implementar vision computacional y machine learning. Además desarrolle habilidades como el trabajo en equipo y la organización para asegurarnos que todos estemos en la misma página y saber que esta haciendo cada quien y como se va progresando. Otra cosa que aprendí fue a conocer las limitaciones con las que contamos y como optimizar los recursos para poder trabajar con estas limitantes.

Gustavo.

La elaboración de este reto que se llevó a cabo en estas ultimas 10 semanas del curso de robotica fue algo nuevo en lo que respecta a desafios y aprendizajes pero gracias a este challenge pude poner en practica cada uno de los conocimientos que he visto en mi carrera de una manera practica y realista. El trabajar y llevar modulos donde enseñen ROS, vision computacional, controladores y machine learning fueron de vital importancia en la creación de este reto pues, aunado con el conocimiento de cada uno de mis compañeros de equipo se diseño un proyecto funcional en su totalidad. Pero tambien llevo de aprendizaje el uso de herramientas externas en el entorno de robotica como el uso de YOLOv5 y de Roboflow para la vision del puzzlebot y a su vez el uso de librerias como PyTorch y OpenCV para detección de colores, a su vez logre ver las capacidades y limites de la Jetson de NVIDIA lo cual me hizo hacerme consciente de que siempre tengo que tomar en consideración en cualquier proyecto los limites fisicos (como la memoria y el espacio) de el sistema en el que estoy trabajando.

Fernando.

La introduccion a la robótica desde inicios del semestre nos ha resultado muy gratificante, se nos ha permitido conocer esta área y también conocer todo lo que implica, desde sus funcionalidades, bases y conceptos basicos, hasta el dilema ético de su uso. En este ultimo reto del semestre con el puzzlebot, pudimos indagar mucho mas en el área de automatización, machine learning y vision computacional. De poder aplicar conceptos teóricos de cinematica inversa y física y también programación por el sistema de nodos que ofrece ROS, en una problematica que dia con dia toma mas importancia y relevancia en las tecnologias emergentes. Explorar otras áreas como el Xarm que aunque no es nada parecido a lo que resuelve el puzzlebot, las mismas herramientas y las mismas bases adquiridas en los temas del curso del bloque, sirvieron para resolver su debida problemática. Reforzar estos conocimientos y aplicarlos en situaciones reales, me permitirá desarrollarme mucho más fácil y flexiblemente en mi área de especialización de la carrera o en una maestría.

VIII. Preguntas

¿ Qué tan eficiente es el programa?

El programa no es tan eficiente, utiliza alrededor del 90% de la CPU de la Jetson y 1.6 GB de los 2 GB de memoria que tiene, y en general el programa que más sufre de esto es el detector de señales, que corre a 0.3-0.5 fps

¿ Es más eficiente ?(espacio de memoria ocupada, tiempo de solución, tiempo de implementación)

Después de los cambios que se realizaron del lunes al miércoles, el programa fue un poco más eficiente, ya que pudimos recuperar 600 MB de memoria RAM de la Jetson.

¿ Se pueden extender sus capacidades ?

Sí, con la solución actual no se está utilizando al máximo el rendimiento de la GPU, si en lugar la utilizaramos en vez de utilizar la CPU para ahí correr las inferencias de nuestro modelo de YOLO, los fps podrían incrementar hasta 20.

¿ Se puede utilizar para otros problemas ?

Sí, nuestra solución de siguelíneas se puede aplicar a otros problemas, ya que es un código que no depende específicamente del Puzzlebot y que ajustando unos cuantos parámetros, podría funcionar para cualquier robot de differential-drive. Además, el modelo de YOLO es capaz de

reconocer 4 señales de tránsito además de las que utilizó en el reto.

¿Por qué funciona?

Nuestro programa funciona gracias a que la plataforma de ROS nos permite integrar programas individuales que procesan un poco de datos y los convierte en información, que al final llegan a dar a un nodo central que se encarga de mandar las velocidades a las que se va a mover cada una de las llantas del Puzzlebot.

¿ Podemos evaluar su comportamiento?

Sí, es posible evaluar el comportamiento del Puzzlebot gracias a la guía de lo que tenía que hacer en el reto. En términos muy básicos, podemos calificar en sí y no si cumple con cada una de las metas que tiene, pero más específicamente, podemos evaluar el comportamiento de YOLO con su mean Average Precision y el rendimiento del siguelíneas monitoreando el error que tiene a través del tiempo respecto al centro de la línea.

¿Es fácil determinar las limitaciones del programa ?

Como el reto del Puzzlebot fue pequeño, es fácil determinar nuestras determinantes. Sabemos que únicamente nos podemos mover siguiendo la línea y que dependemos de la luz para seguirla, que nuestra solución únicamente está hecha para detectar colores muy específicos del semáforo y que no podemos identificar más señales de tránsito a parte de las 7 que tiene entrenadas. Adicionalmente, estamos limitados a utilizar Python 3.6 para el reto ya que es la versión que viene con nuestra distribución de Ubuntu, al igual que debemos utilizar ROS Melodic por lo mismo.

¿ Cuál es el criterio para que el desempeño sea considerado como aceptable ?

Para el reto del puzzlebot el criterio que nosotros definimos para que el desempeño fuera considerado estable fue el basarnos en las instrucciones que Manchester Robotics nos pidio para la competencia del puzzlebot, a partir de eso nosotros definimos ”mini-criterios” de aceptabilidad donde definiamos que para que fuera funcional tenia que seguir la linea sin tener perturbaciones u oscilaciones, que no se descarrilara, que identificara correctamente el color y las señales de transito y que cuando las identificara siguiera las instrucciones de detenerse, seguir, reducir la velocidad, etc. dependiendo del tipo de señalamiento que tenia presente.

¿Está claro el por qué el programa funciona o no funciona?

Aunque en un inicio teniamos dudas en equipo de porque en ocasiones el puzzlebot no respondia como nosotros le habiamos indicado el haber analizado estas situaciones nos hizo percatarnos de los ligeros problemas que teniamos que corregir y al investigar en internet poco a poco nos quedo en claro porque no funcionaba. Una vez que lo vimos hicimos las correciones necesarias y realizamos un programa funcional el cual todos como equipo teniamos claro como y porque funcionaba.

¿Cuál es el impacto de hacer ligeros cambios en el programa?

Para nuestro proyecto el ir haciendo ligeros cambios a los alores que recibia para la identificación del color del semaforo fue de gran impacto pues de esta manera el puzzlebot lograba reconocer con mayor exactitud el color y asi seguia las indicaciones que tenia que realizar sin afectar para nada el rendimiento y velocidad del mismo.

IX. Bibliografía

WikiRos. (s.f.). Documentación de ROS. [Pagina Web]. Recuperado de: http://wiki.ros.org

GoogleColab. (s.f.). Train of Yolov5. Recuperado de: https://colab.research.google.com/drive/1yvFOkAAcyW3jP2u5tr76MTPbLYgUNDgcscrollTo=1NcFxRcFdJ$_O$

$Roboflow.(s.f.).TrafficSignalsPuzzlebot.[Dataset].$ $Recuperadode : https : //app.roboflow.com/serviciosocialworkspace/traffic - signals - puzzlebot/deploy/5$

$OpenCV.(s.f.)DocumentacióndeOpenCV.[PaginaWeb].$ $Recuperadode : https : //docs.opencv.org/4.x/$

.

X. ANEXOS

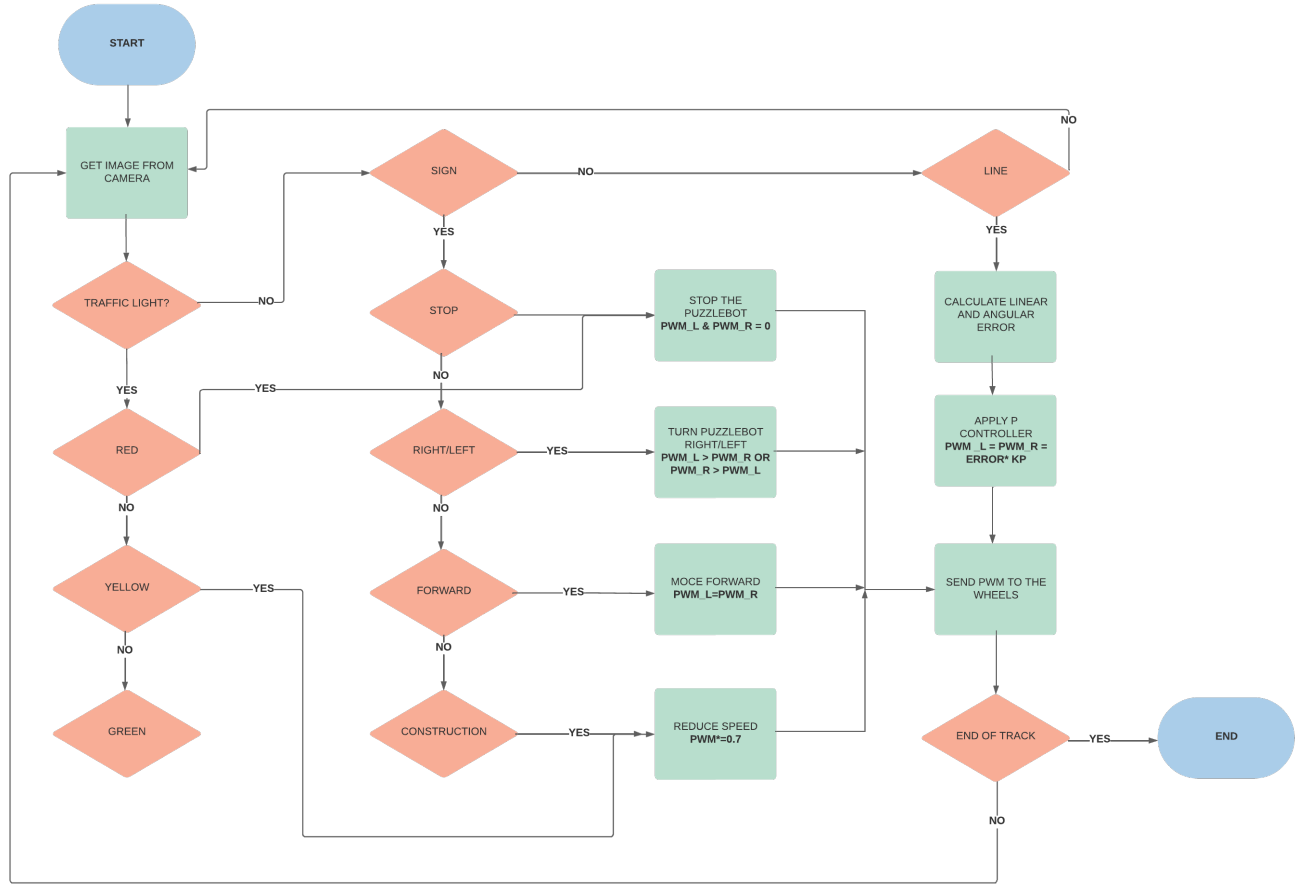

Fig. 16. Diagrama de flujo de funcionamiento del puzzlebot

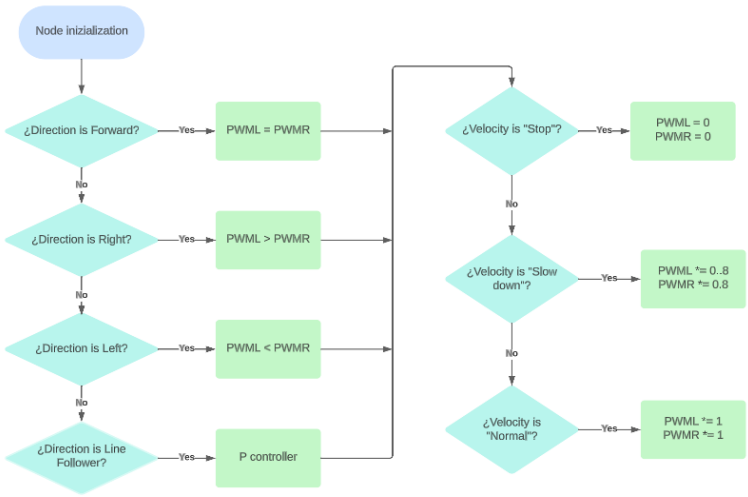

Fig. 17. Diagrama de funcionamiento del controlador