Tecnológico de Monterrey

Implementación de robótica inteligente

(Gpo 501)

Navegación autónoma de vehículos

Puzzlebot

Equipo 5

Guillermo Emanuel Ayala Martínez	A01197833
Mario Javier Cervantes Hernández	A00711813
Miguel Octavio Ramirez Romero	A00828311
Ricardo Gonzalo Cruz Castillo	A01284147
José Miguel Varela Muñoz	A01620480

Introducción

El reto es desarrollar un vehículo autónomo de dos ruedas que están unidos a un eje, cuyo frame tiene un subensamble de policarbonato y acrílico sobre el cual está un arreglo de una computadora Jetson, una cámara de una lente y un embebido para detectar posición e inercia del vehículo en su navegación. Asimismo, se tiene una pista armada tipo circuito sobre la cual se desplaza el vehículo.

El propósito de el reto es programar la computadora Jetson para que controle la cámara y el embebido para desarrollar un algoritmo de visión computacional que a través de yolo (software de reconocimiento) y Machine Learning se entrene para seguir las líneas de la pista, los señalamientos de tránsito tales como; semáforo (luz roja, verde, amarilla), vuelta a la izquierda/derecha, señalamiento de alto, siga y precaución.

El algoritmo de control y visión computacional, se desarrolla en un scripts dentro de ROS y con lenguaje de Python.

Objetivos principales y aprendizajes iniciales

Los tres objetivos, su metodología y sus aprendizajes correspondientes fueron:

- **Seguidor de Lineas**: En esta aplicacion, es necesario poder distinguir una linea continua con la misma intensidad de color a lo largo de esta. La línea que se está por seguir tiene que resaltar en comparación de su entorno, debe quedar claro que es la línea y que no es parte de la línea. Antes de poder identificar el contorno de la línea a seguir es necesario preprocesar la imagen que se recibe. Primeramente, se convierte a escala de grises la imagen. Consecuentemente, se aplica un filtro gaussiano para reducir el ruido y después se aplica un 'threshold' binario. Esto ultimo convierte la imagen en escala de grises a blanco y negro, es decir "es la linea" o "no es la linea". Finalmente, con la librería 'OpenCV' se utiliza la función 'findCountours()' para poder obtener el contorno de la línea.

- **Semáforo**: Para poder detectar los colores del semáforo se sigue una lógica parecida a la del seguidor de líneas, 'o es x color o no lo es'. Se aplica una máscara binaria para cada color donde existe un threshold de valores RGB donde entran los colores que se pueden clasificar como rojo, amarillo o verde y se elimina el ruido de la imagen con un filtro Gaussiano. Sin embargo, esto no es suficiente debido a que también tomaría como semáforos los objetos con estos colores. Es por esto por lo que se utiliza la transformada de círculos de Hough, la cual sirve para poder detectar círculos dentro de una imagen a escala de grises. Con la función de 'HoughCircles()' de OpenCV es que podemos segmentar las figuras circulares dentro de la imagen ya previamente procesada. Finalmente, se hacen distintas condicionales donde se actúa dependiendo del tipo de círculo que fue encontrado dentro de la imagen.

- **Señalamientos de Tránsito**: Para poder identificar las señales de tránsito se entrenó un modelo en Roboflow. Para este supervised learning se utilizó un dataset de más de mil imágenes para entrenar el modelo. Después de esto simplemente fue integrarlo a un código y hacer que el vehículo tome una decisión dependiendo de la señal, se modifica la dirección y velocidad del robot.

Descripción del proyecto

El proyecto fue resuelto tomando como base el funcionamiento en ROS. Inicialmente se plantearon la creación de 4 nodos, 3 nodos para los 3 objetivos principales y 1 nodo que permita la integración y el control de las velocidades, lineal y angular, del puzzle bot. Sin embargo, al comenzar a desarrollar nodos más complejos nos dimos cuenta que la memoria de la tarjeta Jetson se saturaba rápidamente, por lo que se decidió integrar todos los nodos en un solo, de modo que la imagen pueda ser leída y procesada en un único nodo, y esta no tenga que ser publicada en ros topic.

Pseudocódigo del nodo

Como se mencionó anteriormente, para eficientar el procedimiento se agrupó todos los objetivos en un solo nodo que realiza lo siguiente:

Función principal:

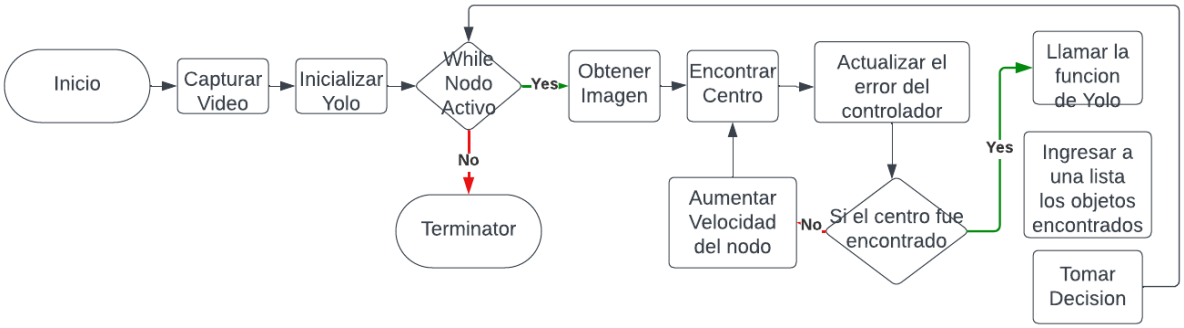
1. Inicializar captura de video
2. Inicializar Yolov5, cargando los pesos del modelo
3. Mientras el nodo este activo
    - 3.1. Obtener imagen
    - 3.2. Encontrar el centro del carril
    - 3.3. Actualizar el error del controlador
    - 3.4. Si el centro del carril fue encontrado:
        - 3.4.1. Llamar a la función de detección de Yolo
        - 3.4.2. Ingresar a una lista los objetos encontrados
        - 3.4.3. Tomar una decisión según sea el objeto
    - 3.5. Si el centro del carril no fue encontrado:
        - 3.5.1. Aumentar la velocidad del nodo para lograr encontrar el carril

Función de detección de línea

1. Reducir el tamaño del frame
2. Obtener la región de interés (parte central inferior)
3. Convertir la imagen a blanco y negro
4. Aplicar un filtro bilateral para reducir ruido pero preservar los bordes
5. Obtener el promedio de intensidad de píxeles por columna
6. Identificar los máximos del vector de intensidades
7. Encontrar el centro de ambos máximos
8. Identificar si los píxeles cercanos corresponden a la línea central del carril.

Nota: la detección del semáforo y su color se hace igualmente con el modelo de YOLO

Diagramas de bloques del código desarrollado



Resultados obtenidos

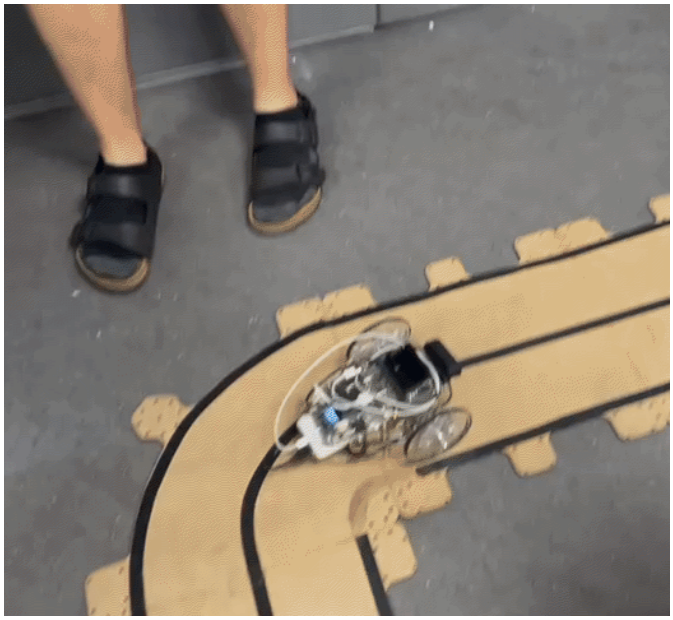
El diseño, desarrollo e implementación de el puzzle bot no se logró al 100 %, debido a varias razones y factores; la capacidad de procesamiento de imágenes de la jetson para correr el archivo .pt de weights entrenados para Yolo no era eficiente, nos daba alrededor de 5 fps, comparado con un 60fps que es lo que el humano necesita para poder distinguir de manera natural las imágenes.

Por otro lado, las versiones de Yolo que se intentaron , V3, V5, V7, V8 no funcionaban del todo bien en la jetson. Para esto buscamos algunas alternativas como correr el algoritmo en una virtual environment a través de otra computadora para liberar el procesamiento de imágenes a la jetson y que está solo corriera el modelo entrenado para identificar las señales de tráfico y semáforo.

La parte que si logramos implementar de manera más consistente y firme fue la del control PD para el seguimiento de líneas en los carriles del track. También se probaron alternativas como PID, P, y finalmente se demostró que PD era la mejor opción para guiar al vehículo del puzzle bot de manera autónoma.

Finalmente a manera de conclusión pudimos observar que la calidad y robustez del hardware, aunado con las variantes del modelo de Yolov5 entrenado no proporcionaban esa garantía de performance de manera estable en el puzzle bot. Es decir, algunas veces funcionaba bien y en otras no, lo que detonaba una baja confiabilidad en el funcionamiento del vehículo.

Nos hubiera gustado mucho seguir probando alternativas, pero creemos que el tiempo ya no fue suficiente para poder cubrir esos gaps de mejora y por otro lado evidentemente se requiere un mejor hardware para incrementar la robustez del sistema.



Repositorio:
https://github.com/PEPE-cyber/puzzlebot-E5

Drive de las evidencias:
https://drive.google.com/drive/folders/1j5jHAmWR0dQURL4t1vat2i7bLKfE1DsM?usp=sharing

Problemáticas y recomendaciones

Rendimiento de la JETSON

La Jetson es una computadora que posee 2GB de capacidad para su procesamiento, ello debería incluir como hardware el procesador memoria en tiempo real. Principalmente es un elemento de hardware. En el entendido que es una placa de desarrollo de AI que puede procesar algoritmos de AI directamente en el dispositivo sin conexión a la nube o internet.

La Jetson de 2GB se basa en la arquitectura de procesamiento de Nvidia y está equipada con un sistema en chip (SoC) Tegra, que incluye una GPU (unidad de procesamiento gráfico) y una CPU (unidad central de procesamiento) en un solo paquete. Es compatible con el entorno de desarrollo de Nvidia, lo que facilita la creación y el despliegue de aplicaciones de IA en el dispositivo. Cuentan con bibliotecas y herramientas de software de Nvidia, como CUDA y cuDNN, para acelerar el entrenamiento y la inferencia de modelos de aprendizaje automático.

En cuanto a su funcionamiento, la Jetson de 2GB ejecuta sistemas operativos basados en Linux y proporciona una amplia gama de puertos de entrada y salida para la conexión de periféricos.

No obstante lo anterior, la jetson no tuvo la capability de procesamiento para el algoritmo de YOLOV5 y su performance fue de bajo nivel. se calentaba demasiado y se sobreprocesaba.

Recomendación

Una de las recomendaciones que pudimos observar fue implementar un sistema de enfriamiento que aumentará el performance del procesamiento. La optimización del rendimiento puede variar según la aplicación y los requisitos específicos. Es recomendable realizar pruebas y experimentar con diferentes técnicas para encontrar la configuración óptima para tu caso de uso en particular. Asi que las siguientes son algunas recomendaciones:

Optimización del código: Asegúrate de que tu código esté bien optimizado para aprovechar al máximo el hardware de la Jetson. Utiliza bibliotecas y funciones de optimización, como las proporcionadas por CUDA, para acelerar los cálculos en la GPU.

Paralelización: La Jetson tiene una potente GPU con múltiples núcleos, por lo que es importante aprovechar la capacidad de paralelización. Divide las tareas en hilos o procesos paralelos para ejecutar cálculos simultáneos y aprovechar al máximo la potencia de la GPU.

Uso eficiente de la memoria: La memoria es un recurso limitado, por lo que es fundamental utilizarla de manera eficiente. Minimiza las transferencias de datos entre la CPU y la GPU para reducir la latencia y optimizar el rendimiento. Utiliza técnicas de almacenamiento en caché y estructuras de datos compactas para ahorrar memoria.

Optimización de modelos de IA: Si estás utilizando modelos de aprendizaje automático en la Jetson, considera la posibilidad de optimizarlos para que sean más eficientes. Puedes utilizar técnicas como la cuantización de modelos, la poda de pesos y la compresión de redes neuronales para reducir el tamaño y mejorar el rendimiento de la inferencia.

Monitoreo del rendimiento: Utiliza herramientas de monitoreo para supervisar el rendimiento de la Jetson. Esto te permitirá identificar cuellos de botella y áreas que requieran optimización. Puedes utilizar herramientas como el NVIDIA System Monitor o desarrollar tus propias soluciones de monitoreo.

Actualización de software: Asegúrate de tener instaladas las últimas versiones de software y controladores para la Jetson. Nvidia suele lanzar actualizaciones que incluyen mejoras de rendimiento y correcciones de errores.

Gestión térmica: La Jetson puede generar calor durante el uso intensivo de recursos. Asegúrate de que el dispositivo tenga una buena disipación de calor y evita la sobrecarga térmica, ya que puede afectar el rendimiento. Considera el uso de disipadores de calor o ventiladores adicionales si es necesario.

Falsos positivos en la detección de señales azules

Después de muchos problemas para la aplicación de yolo en la jetson, encontramos muchos problemas de confusión de señales por parte del modelo de yolo. Esto ya siendo un problema casi al final del tiempo límite fue bastante molesto ya que sin una correcta detección de las señales de vuelta y seguir adelante el puzzlebot era incapaz de seguir la pista.

Tuvimos varios acercamientos a una solución, sin embargo, terminé adaptando un código donde según el comportamiento que ya había tenido yolo anteriormente hacia las señales se adapta para entender realmente cuál era la señal que tenía de frente. Esto funcionaba mediante promedios de veces que yolo registraba cada señal, tomando la decisión de cuál era la señal según el promedio más grande.

Cambio de tiempo de muestreo

Inicialmente, teníamos un seguidor de línea bastante bueno, que lograba recorrer la pista a un buen ritmo y de manera estable. Sin embargo, al comenzar a implementar más nodos o procesamiento más complejos nos dimos cuenta que el seguidor perdía desempeño. Esto se debe a que el control parte de una función de transferencia y, al modificar el tiempo de muestreo, cambia totalmente el desempeño.

Recomendación

Idealmente se debería de correr los nodos de manera independiente, de modo que se logre establecer un tiempo de muestreo fijo y así no no se cambie el comportamiento. Sin embargo, en este caso, por la falta de memoria se descubrió que se obtenía mejor rendimiento con todo el procesamiento en un único nodo. Si se tuviera un mayor poder computacional, se recomienda utilizar nodos independientes con frecuencias definidas.

Cámara

Una de las problemáticas que tuvimos fue la cámara proporcionada.Inicialmente, la problemática era el posicionamiento de esta. Tenía que estar puesta de manera en que la línea y las señales se vieran al mismo tiempo. El puzzlebot fue entregado con la cámara en la parte superior, para poder observar la línea en el suelo el ángulo era muy en picado y por ende ya no se podían apreciar las señales. Además, el mecanismo donde estaba montada la cámara no era muy estable y esta se movía de la posición óptima en la que se había acomodado. Es por esto que se decidió por mover la cámara al centro del puzzlebot donde se podía apreciar tanto la línea como las señales y el espacio era lo suficientemente reducido para que la cámara no se moviera.

Desafortunadamente, conforme pasó el tiempo el pegamento de la cámara se rindió y la esta se caía y se desconectaba. A pesar de que nada garantizaba que la cámara no se iba a desconectar a mitad de la marcha del puzzlebot, este no era tanto problema, solo era cuestión de conectarla. No obstante, este problema recurrente tomó mayor peso ya que empezó a afectar nuestra experimentación y solución de problemas. En un momento en el cual se querían probar cosas nuevas con la cámara, esta se desconectaba constantemente. El problema recaía en el hecho de que al momento de perder la conexión y volver a

conectar la cámara, la Jetson no la reconocía al menos que se reiniciara el sistema. Por ende, se perdió mucho tiempo reiniciando todo cada vez que la cámara se desconectara. Es por esto que se optó por aprovechar los puertos usb de la Jetson y utilizar una webcam.

Implementación de YOLO

En esta implementación del reto tuvimos bastantes problemas a la hora de hacer las pruebas directamente en la Jetson, ya habíamos probado algunas versiones de Yolo para nuestras computadoras durante la clase de Machine learning desde el que nos tocó presentar como equipo la versión 8, como la versión 7, 5 y 3 por el rendimiento que presentaban, en la versiones 7 y 5 se pudo hacer la interconexión con ROS, sin embargo, la estábamos haciendo directamente en la computadora y no teníamos en cuenta que para la Jetson era diferente. Inicialmente la idea era probar con Yolov5 debido a que en las sesiones con el socioformador Manchester Robotics se nos proporcionó dicha versión por lo que se intuyó que esta versión funcionaba correctamente para nuestro caso, primeramente intentamos probar con estos archivos pero tuvimos problemas a la hora de la instalación ya que nos hacían falta algunos paquetes para la interacción con ROS, y la inconsistencia que presentamos es que para esta implementación necesitamos de una versión de python por lo menos de 3.6, mientras que la versión de ROS, en este caso Melodic funciona para python 2.7, lo cual generaba problemas para correr archivos ya que debíamos especificar qué archivos correrían con una versión de python específico con sus respectivos paquetes; después de esto seguimos probando con algunos otros repositorios de Yolov5 que estuvieran funcionando con ROS pero no teníamos éxito, por estos problemas decidimos probar con otras versiones para ver si conseguiamos que funcionara en la Jetson.

Finalmente volvimos a probar uno de los repositorios de la versión 5 solucionando los problemas que generaba al momento de la instalación pudiendo correr la red neuronal en la Jetson y comunicándose con ROS, en primera instancia probamos con solo una imagen y lo hacía de manera correcta solo que demoraba unos segundos en hacer el procesamiento.

El problema principal que presentaba la Jetson es que no cuenta con suficiente memoria para correr un proceso tan pesado como lo es el reconocimiento de imágenes a través de Yolo, además de esto debíamos correr otros pequeños procesos que sumaban memoria y saturaba el sistema por lo que no era óptimo el realizar este proceso tan pesado en la Jetson, se decidió probar el correr dicho proceso en otra computadora y que la Jetson nomas recibiera qué señal se había identificado, pero no se pudo implementar debido a que no contábamos con el tiempo suficiente, finalmente utilizamos el Yolo de manera nativa en la Jetson y se nos devolvía la información correspondiente en la mayoría de los casos, el rendimiento no fue el mejor y es por ello que tenía fallas en la detección y seguimiento del camino.

Aprendizajes Individuales

José Miguel Varela Muñoz

Este reto fue sumamente enriquecedor para mí, me ayudó a aprender bastante teóricos, desde los conceptos básicos de ROS hasta la implementación de algoritmos de visión computacional y machine learning. Sinceramente, la implementación que desarrollamos a pesar de no ser muy sofistica, fue bastante buena para empezar a comprender los conceptos de robótica avanzada.
Por otra parte, yo creo que me llevo bastantes aprendizajes en soft skills, como el manejo del tiempo, trabajo en equipo y resiliencia. Yo creo que la principal fue la resiliencia, especialmente si consideramos todas las situaciones adversas que se presentaron a lo largo del semestre; la demora en la entrega del robot, la poca capacidad de la jetson, la compatibilidad de librerías, por mencionar algunas.
Otro aprendizaje que me llevo es la importancia de entender en donde nuestro desarrollo será desplegado, tanto en que situaciones como en que computadoras. Al principio, yo comencé a codificar un algoritmo para la detección de la línea bastante vigoroso. Este, funcionaba bastante bien y lograba detectar el camino en diferentes situaciones, lo probé en mi computadora y obtuvo resultados óptimos. Empero, al ser implementado en la jetson, el procesador no lograba ejecutarlo a una velocidad adecuada, lo que provocaba que no se lograra seguir el camino. También tuvimos algunas dificultades con las variaciones del camino, la luz de los señalamientos, etc. todo esto porque no consideramos las condiciones a las que se iba a someter el programa.

Ricardo Gonzalo Cruz Castillo

En términos de áreas de conocimiento este proyecto me ayudó a integrar ROS, Visión computacional y Machine learning. Estos tres temas siempre han sido muy sonados a lo largo de la carrera, sin embargo no fue hasta ahora que por fin se pudo tomarlos y desarrollar un proyecto implementado todos estos conocimientos.
Sin embargo, creo que el mayor reto no fue utilizar mis conocimientos para el desarrollo de nuevos algoritmos, o bien, el hecho de tener que coordinarnos como equipo y ser consistentes con las horas que le dedicamos al proyecto o el “troubleshooting” de las librerías y paquetes que se utilizaban. Para mi, el mayor reto fue trabajar con las herramientas y el hardware que se nos proporcionó. No estoy tratando de decir algo por el estilo de “la jetson no funciona y los componentes dejan mucho que desear”, estoy tratando de decir que era nuestro deber, como equipo, optimizar nuestras implementaciones de manera en la que pudieran funcionar dentro de lo que nos fue dado. Mientras que si, integre conocimientos de distintas áreas, pero eso solo es una parte. Lo que debe hacer un ingeniero es trabajar con lo que tiene. Es un hecho que hubo algunas fallas de logística, pero en campus Monterrey todos tuvimos el mismo tiempo y observe cómo es que otros equipos si desarrollaron implementaciones para poder optimizar el rendimiento de los algoritmos. No fue hasta después de las presentaciones que me di cuenta que en realidad, siento que mi desempeño fue algo deficiente, no busque soluciones como las mencionadas en clase por el estilo de: “¿Que hubiera pasado si tu modelo lo entrenabas de tal manera? ¿Por qué no usaste una tarjeta SD como RAM?” Creo que ese fue mi mayor aprendizaje y mi nueva consideración para todos mis proyectos a futuro.

Guillermo Emanuel Ayala Martínez

Personalmente me adentre durante este proyecto a temas que sabía que eran vitales para la robótica pero nunca había tocado. Durante este reto me adentre y trabajé mucho en la parte de visión computacional con un factor de reto muy grande, ya que la parte de visión puede llegar a ser muy demandante para los equipos con los que trabajamos. Siendo esto las primeras veces que el proceso puede sobrepasar a los equipos que utilizamos como computadoras aprendí mucho sobre optimización de recursos y desarrollo de un sistema completo.

Uno de los aprendizajes más importantes para mi fue sobre el acercamiento que le damos a los problemas, ya que muchas veces comenzamos muy arraigados en el tipo de soluciones que le queríamos dar a los problemas cuando tal vez existía algo más sencillo o diferente que se nos presentaba por parte de otros equipos o por foros en internet. El ejemplo más grande creo que seria el tiempo que estuvimos intentando correr soluciones diferentes para la aplicación de yolo, sin embargo al final del tiempo que tuvimos, terminamos optando por yolo nativo en la jetson que fue una sugerencia de un equipo y la misma sugerencia de manchester desde el principio.

Mario Javier Cervantes Hernández

Este proyecto fue un reto que me permitió comprender los alcances de la autonomía vehículos no tripulados que requieren aplicaciones diversos de la robótica; la visión computacional , sistemas de machine learning, deep learning, y sistemas de control basados en modelos matemáticos que definen la dinámica del vehículo que permite modelar la pose y la orientación del mismo.
Entendí que uno de los mayores problemas a la que se enfrenta el campo de la robótica es la visión computacional, el nivel de dataset y weights que necesitan un nivel de entrenamiento donde se requiere muchas imágenes que permitan garantizar los epochs que den confiabilidad a la detección de imágenes para la conducción segura del vehículo de manera segura.
Por otro lado, me dejo entender que la mejor manera de controlar el vehículo es un PD y en ocasiones se puede combinar con un PID.

Miguel Octavio Ramirez Romero
Antes de empezar este bloque sabía que los temas que se tocaban en él eran de suma importancia para la carrera o quizás de los más importantes ya que en sí los temas vistos y la implementación del reto es más aterrizado a la robótica al ver ya propiamente un sistema como el carrito y el software como ROS una de las cosas básicas para un ingeniero en robótica para el modelado y simulado de cualquier sistema, así como también los conceptos de Machine Learning y visión computacional; en lo personal tenía estaba un poco perdido al inicio al volver a utilizar ROS por haber llevado la primera parte hace 1 año y retomarlo ahora, además de implementar nuevos conceptos y algoritmos como los ya mencionados por lo que me llevo muchos aprendizajes tanto teóricos como de aplicación, estos primeros al verlos en clase, hacer investigación y leer, mientras que los segundos con algunas prácticas que tuvimos en esos momentos pero sobre todo durante estas últimas 3 semanas en las que pudimos trabajar con la Jetson y el vehículo. Aprendí la gran importancia que tiene la visión computacional para los

vehículos autónomos no tripulados ya que como su nombre lo indica es la visión que tiene el sistema para el entorno, visto de otra forma son los ojos del sistema; con ello se puede extraer información de lo que se ve y tomar las decisiones correspondientes y óptimas según sea el caso.
Por otro lado, también aprendí que es importante tener en cuenta el hardware con el que estás trabajando y optimizar la mayor parte del software o el programa para que la ejecución sea la más eficiente, si bien en este caso los componentes físicos no eran lo mejor y hubo un poco falta de tiempo considero que podíamos haber buscado opciones para optimizar el vehículo y así su funcionamiento fuese mejor del que obtuvimos al final.

Preguntas sobre Algoritmos

¿ Qué tan demostrativo es el algoritmo ?
El diseño de los algoritmos es bastante demostrativo ya que demuestra el proceso y/o solución a los problemas abordados, en este caso el seguidor de líneas y la detección del semáforo, esta demostración la podemos ver en el funcionamiento ya sea por la computadora de manera separada los códigos o ya implementada con el puzzlebot, además que el algoritmo es fácil de leer por el lenguaje de programación utilizado (python).

¿ Cuáles son los criterios para escoger los ejemplos ?
Los criterios para escoger los ejemplos se realizan primero con las pruebas más sencillas a realizar para ver que la implementación del algoritmo sea correcta y haga su funcionamiento, después de comprobar que funcione de manera general se realizan pruebas más complejas para ver la robustez del algoritmo y de esta manera abarcar la mayoría de casos, por último se toman como ejemplos las pruebas que tengan más dificultad y se hayan realizado de manera correcta.

¿Qué tan eficiente es el programa ?
Hasta el momento nuestro programa es razonablemente eficiente ya que estamos realizando varios procesos a la vez, de manera individual cumple con el propósito de las tareas de manera suficientemente estable, pero al momento de probar varias funciones el programa se ralentiza y no llega a cumplir de manera idónea las tareas.

¿ Requiere acceso a bases de datos o funciones externas?
No, para el funcionamiento no requiere acceso a recursos externos. Sin embargo, en el modelo de reconocimiento de imágenes, utilizado para la detección de los distintos señalamientos viales, se empleó un extenso dataset y una herramienta externa, llamada roboflow, en la cual se subieron muchas imágenes y se realizaron anotaciones. Posteriormente se utilizaron dichas imágenes etiquetas para entrenar el modelo. A pesar del uso inicial de herramientas externas, se pudo generar una serie de ponderaciones de la red neuronal, mismas qué pueden ser descargadas e incluidas en los archivos de la Jetson, de modo que no se tenga necesidad de acceder a recursos externos.

¿Se pueden extender sus capacidades?
Sí, todos los métodos empleados pueden ser mejorados para que logren un mejor desempeño o extender sus capacidades a diferentes entornos o parámetros visuales. Por ejemplo, en caso del método que reconoce los señalamientos viales puede ser mejorado si el modelo es entrenado con un dataset más grande y variado. De igual manera, se puede sintonizar o calibrar de mejor manera los distintos parámetros utilizados en las funciones empleadas en los métodos de reconocimiento de camino y del semáforo. Por ejemplo, se puede variar la longitud mínima de las líneas, el tamaño del kernel del filtro de blur gaussiano, etc. de modo que se pueda mejorar los algoritmos.

¿Considera situaciones nuevas ?
Teóricamente sí, el método debería ser capaz de recorrer cualquier pista, siempre y cuando la línea central sea más oscura que el resto del camino. Igualmente, el reconocimiento del método de YOLO debe ser capaz de reconocer las señales en distintas situaciones, especialmente si consideramos que se utilizó un dataset bastante amplio. Indudablemente, valdría la pena realizar pruebas en distintas pistas para identificar las limitaciones del sistema actual.

¿ Depende de otros métodos o técnicas ?
El reto en sí es una mezcla de dos ramas, visión computacional y machine learning. De las tres implementaciones, el reconocimiento de señales de tránsito es la que ejemplifica esta mezcla de mejor manera. A través de un modelo entrenado previamente es que el algoritmo puede tomar decisiones con base en las señales identificadas, por ende, depende de una técnica de machine learning. Se utiliza el método de terceros, YOLO v5, para el reconocimiento de objetos en las imágenes. Este método fue únicamente empleado y entrenado por nosotros. Por otro lado, utilizamos un controlador PID, un método ampliamente utilizado en control. Este método data del siglo 17, sin embargo, en este proyecto fue codificado por nosotros.

¿ Qué tan eficiente es el programa ?
Sinceramente, yo creo que el programa es bastante eficiente (eficacia) ya que cumple con sus objetivos. Si probamos la eficacia de los distintos objetivos, vemos que el programa logra todos; logra reconocer señales, logra identificar el centro del carril y logra detectar el semáforo en sus distintos colores. Sin embargo, puede que no sea lo suficientemente efectivo y utilice demasiados recursos como para poder ser ejecutado en la jetson ya que al momentos de activar todas la funciones el procesador se satura y no logra cumplir sus objetivos.

¿ Está claro el porqué el programa funciona o no funciona ?
Si, como se ha mencionado anteriormente no funciona por la falta de eficientizar los procesos, lo que provoca que el procesador y la memoria se saturen y no se logre obtener un desempeño óptimo. Valdría la pena probar el programa en un equipo con mayor capacidad de procesamiento.

¿ Se pueden dar generalizaciones a partir de los ejemplos utilizados ?

No, de ninguna de las funciones podríamos generalizar. Por ejemplo, el seguidor de línea actualmente funciona, pero no podemos generalizar que en todas las pistas el puzzlebot lograra mantenerse dentro del camino. Igualmente la detección de las señales, valdría la pena realizar pruebas en distintas circunstancias, con distinta iluminación, algunos daños las señales y otras variaciones para determinar a qué grado la solución es general.

¿ Es fácil determinar las limitaciones del programa ?
No, considerando todos los algoritmos y el uso de machine learning, yo creo que es bastante difícil determinar las limitaciones del programa. Por un lado, podríamos probar el seguidor de línea para conocer qué tipos de trayectorias puede y no puede seguir. O podríamos identificar que objetos el modelo de YOLO logra identificar, cuántos se logran detectar, cuántos falsos positivos, cuántos falsos negativos, etc. Sin embargo, tener un programando todo ejecutándose al mismo tiempo, se presentan un sin fin de posibilidades que pueden afectar el desempeño, por lo que se vuelve complicado enter, calificar o cuantificar las limitaciones del programa.

Si el programa puede ser evaluado por partes, ¿ Se puede determinar el desempeño de unas partes con cambios en otras ?
Si, podemos separar el programa por tareas y evaluar el desempeño de cada una de ellas por separado. Sin embargo, es importante considerar que no porque alguna función tenga un desempeño adecuado cuando esta se ejecuta por separado, esta tendrá un buen desempeño cuando se ejecute simultáneamente con otra. Esto se debe a las limitaciones computacionales, previamente detalladas.

¿ Cuál es el criterio para que el desempeño sea considerado como aceptable ?
En este caso podemos definir que si el puzzlebot logra dar un par de vueltas a la pista, todos los métodos empleados están logrando un desempeño aceptable. En este escenario, no hay mayor problema si algún método no logra obtener resultados favorables por un par de frames, lo importante es que si logre detectar su objetivo a tiempo, y se adapte el control de las velocidad antes de que sea demasiado tarde.

¿Cuál es el nivel de transparencia del método?
El nivel de transparencia de un método de evaluación se refiere a cuán abiertos y accesibles son los procesos, procedimientos y resultados del método para las partes interesadas y los usuarios. Un método de evaluación transparente es aquel que permite una clara visibilidad y comprensión de sus objetivos, metodologías, fuentes de datos, técnicas de análisis y hallazgos. El nivel de transparencia puede variar según el método de evaluación específico y el contexto en el que se aplica.

¿Requiere que los datos de entrada sean pre-procesados de manera automatizada o manual?
En este proyecto se realizan ambos casos. Por ejemplo para el seguidor de líneas, solo es necesario escribir unas cuantas líneas de código para poder pre-procesar los datos de entrada (imágenes de la cámara) y poder detectar los contornos. Sin embargo, para poder detectar las señales de tráfico, fue necesario entrenar un modelo de visión computacional.

Para esto se tuvo que clasificar individualmente y de manera manual cada una de las más de cuatro mil imágenes del dataset utilizado.

¿Podemos evaluar su comportamiento?
Sí, es posible evaluar el comportamiento de una implementación utilizando varios métodos. La elección de los métodos de evaluación depende del contexto específico y de los objetivos de la implementación. Elegir métodos y métricas de evaluación apropiados es crucial para obtener resultados confiables y significativos. Para evaluar el comportamiento del reconocimiento del semáforo y las señales de tráfico es sencillo. Solo es corroborar si el carro está realizando la acción esperada al momento de reconocer cierto tipo de señal. Sin embargo, para el seguidor de líneas, se podría evaluar el comportamiento al realizar algún tipo de función de error que tome en cuenta lo separado que está la línea que se está siguiendo con el centro de la imagen registrada por la cámara.

¿Cómo responde el sistema si las entradas son organizadas de otra manera, que contengan ruido o que vengan incompletas?
Aquí quiero hablar sobre el modelo que entrenamos. En un principio, este sólo reconocía las señales más distintivas como el alto, el dar paso, etc. Sin embargo, las señales azules (forward, rotonda, izquierda, derecha), ya que eran redondas y azules, todas eran clasificadas como 'rotonda'. Aquí se puede considerar que los valores de entrada (el dataset) estaba incompleto. Se tenía un dataset muy pequeño y con imágenes de no muy buena calidad, en un principio eran menos de 150 imágenes. Después se agregaron más de 200 imágenes diferentes para cada uno de las 4 señales azules y con eso fue posible reconocer de manera correcta las señales.

Conclusión general

Un vehículo autónomo es aquel que puede procesar las señales de su ambiente y tomar decisiones con base en ellas sin necesidad de la intervención de un piloto. Para poder desarrollar un vehículo a la escala como la de este proyecto fue necesario conocer sobre control, visión computacional y machine learning. Utilizando ROS, se implementó un seguidor de líneas y un modelo de visión computacional Yolov5 para detección de señales de tráfico dentro de una Jetson Nano. La implementación de los algoritmos creados dentro del hardware no fue lo más sencillo, puesto que habilitar un entorno dentro de la Jetson con todas las dependencias, paquetes y librerías necesarias para poder correr los programas implicó mucho 'troubleshooting'.
La ventaja sobre el desarrollo en ROS es que es escalable. Lo que fue creado para este proyecto puede ser fácilmente adaptado para un vehículo de mayor escala, brindando así una una experiencia enriquecedora que nos acerca aún más a la industria. De igual manera, el desarrollo de este proyecto da fruto a el desarrollo de habilidades blandas. Para poder salir adelante fue necesario organizarse entre equipo, planificar actividades y repartición de tareas, ser perseverantes y pacientes.

Anexos

Video presentación Puzzlebot
https://youtu.be/gcK3CbmAWJc
Slides de presentación Puzzlebot
https://drive.google.com/file/d/1UgAx04QcXBLlPjH2G2OJk9f-M_qx5-Lu/view?usp=sharing
Documento de proyecto xArm
https://docs.google.com/document/d/1jpNdksS-5uZ81nxVJLoRCtUXn9140nT18KMdgyb_vK8/edit?usp=sharing