

# Proyecto Semestral:

## Aprendiendo a ver en tiempo real

Yerko Sepulveda Rojas

*ELE4001 – Robótica*

*Escuela de Ingeniería, Universidad de O'Higgins*

30, 06, 2024

### I. PROBLEMATICA

El proyecto “Aprendiendo a ver en tiempo real” consiste en implementar una CNN de bajo coste en lo que a hardware se refiere, siendo el uso de memoria o de potencia computacional las limitantes con la finalidad de detectar objetos como lo son los Duckies y otros vehículos, adicionalmente, se debe implementar Path Planning y Motion Planning. En su conjunto nuestro Duckiebot debería poder movilizarse autónomamente dentro de la ciudad, pudiendo ir desde un punto A al B, evitando obstáculos que fueron detectados debidamente.

### II. OBJETIVOS DEL PROYECTO

El proyecto tiene 3 objetivos principales, la detección de obstáculos en la vía mediante una CNN eficiente, la implementación de un algoritmo de Path Planning y la implementación de Motion Planning. En su conjunto, el objetivo final es que el DuckieBot se pueda desplazar en el entorno simulado de manera eficiente y segura, evitando los obstáculos que pudiesen aparecer en su trayectoria hacia su destino.

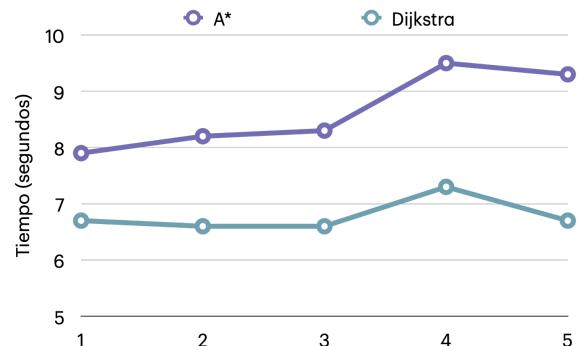
### III. BASE DE DATOS

La base de datos consta en un set de imágenes de 640x480 pixeles, las imágenes pueden mostrar un Duckie, un vehículo o ninguno de estos dos, cada imagen está acompañada de su label respectivo, este label muestra el tipo de objeto y la posición del bounding box, de la siguiente manera, clase x1 y1 x2 y2. El dataset usado es de

creación propia y consta de 543 imágenes para el entrenamiento y de 132 imágenes para la validación.

### IV. ALGORITMOS USADOS

A. *Path Planning: es un algoritmo de planificación de ruta entre un punto A hacia un punto B, para el presente proyecto se implementó Dijkstra y A\*, Dijkstra consiste en expandir en punto de inicio hacia los puntos vecinos, luego los nuevos puntos se vuelven a expandir hasta llegar al destino, por otro lado A\* usa el mismo método pero añade una función heurística que estima la dirección del punto de destino y se expande principalmente hacia esa dirección. Se implementaron Dijkstra y A\*, se testeó el uso de memoria y el tiempo de respuesta para cada algoritmo de planificación de ruta, el uso de memoria es aproximadamente 75mb para ambos*



*algoritmos y el tiempo total para recorrer 100 pares de puntos se puede apreciar en el gráfico (Fig.1). La implementación del Path Planning considera la dirección de las calles y la posición en tiempo real del Duckiebot.*

Fig. 1 Grafico comparativo entre A\* y Dijkstra, 5 test, 100 pares de puntos.



Fig. 2 Imagen del simulador mostrando el uso del Path Planning.

- B. Motion Planning: es el proceso mediante el cual el agente, en este caso el DuckieBot, toma decisiones para llegar de al punto de destino, en el presente proyecto se utilizará la ruta obtenida del path planning, luego se le pasan los 3 primeros elementos a una variable que se utilizará como buffer; constantemente de la ruta original se elimina la posición actual en la que nos encontramos si esta está en la ruta, del buffer verificamos si los siguientes puntos a visitar están en linea recta a mi posición actual, si lo están, entonces nuestro nuevo destino es el punto de buffer[-1], en caso contrario es buffer[0], de esta manera se puede navegar de manera mas rápida y suave. se utilizan 2 PIDs para ajustar la velocidad y el ángulo hacia donde mira el DuckieBot, el PID de velocidad consiste en tener una velocidad deseada y entregarle la velocidad actual, el parámetro resultante se limita dependiendo del ángulo entregado por el PID del ángulo, este PID considera el ángulo hacia la posición deseada desde la posición actual, y se le entrega el ángulo real del robot para calcular el error, el resultado se limita entre los valores de -3 y 3.
- C. CNN: Una Convolutional Neural Network es una red neuronal profunda que se utiliza para procesar y analizar datos, que en este caso es detectar objetos en una imagen y generar bounding box alrededor del objeto detectado. Originalmente se quería implementar una CNN que utilizaba el método de Bitnet, sin embargo, este modelo presentó problemas durante el entrenamiento, durante el primer epoch el modelo se comportaba de manera correcta, pero después los valores se desestabilizaba dando NaN, infinito o similares, por lo que debido al tiempo para realizar el proyecto, este modelo se descarto. De esta manera, se optó por utilizar el modelo de detección popular llamado YOLO, en diferentes versiones para

comparar la eficiencia del modelo resolviendo el problema del proyecto, también se utiliza RT-DETR (Real Time Detection Transformer) que es otro modelo basado en encoder híbridos y redes convolucionales. Estos modelos fueron entrenados con el dataset personalizado, y no se utilizaron modelos pre entrenados, esto tuvo un impacto en el tiempo total de entrenamiento haciendo que aumentara. Para seleccionar el modelo a usar, se consideran los siguientes parámetros, accuracy general del modelo y tiempo de detección, de esta manera se buscará el modelo más eficiente para utilizar en tiempo real.

Algunos parámetros a considerar para seleccionar el modelo de detección son el costo computacional que se basa en el peso del modelo que es proporcional a la cantidad de parámetros, la precisión y el tiempo de respuesta para un conjunto de datos que consiste en un sub dataset de 132 imágenes.

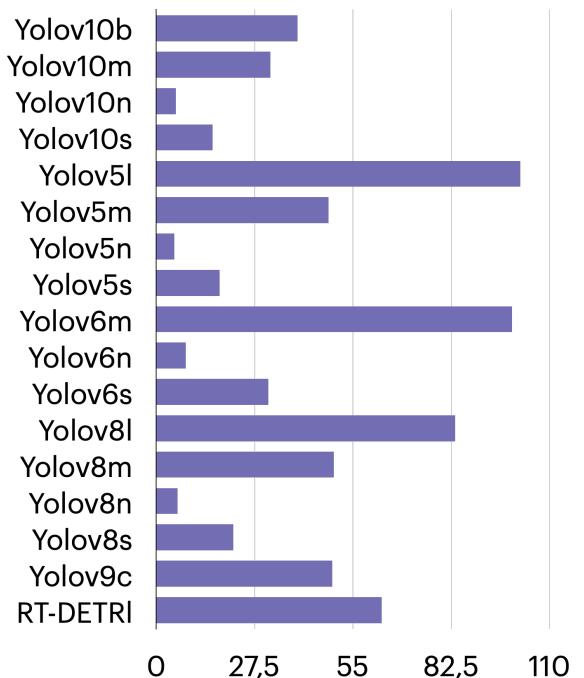


Fig. 3 Costo computacional (peso del modelo proporcional a la cantidad de parámetros).

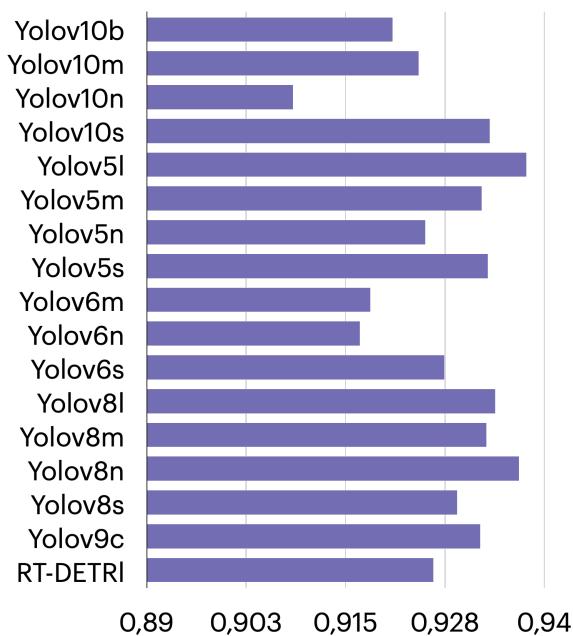


Fig. 4 Precision de los modelos

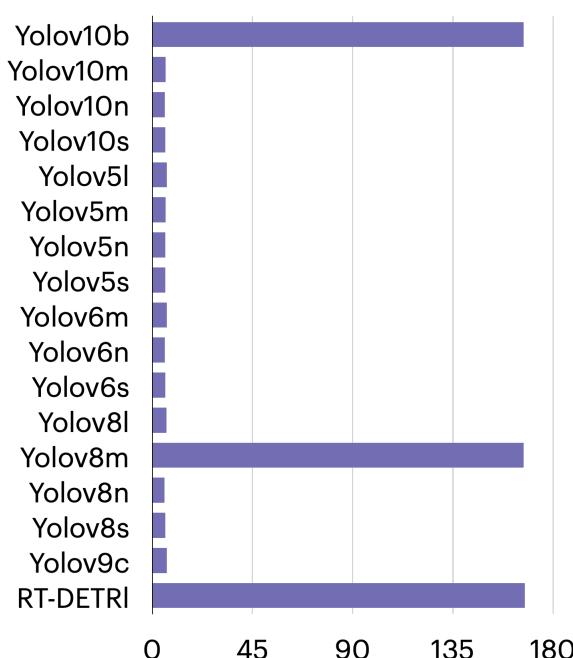


Fig. 5 Tiempo de respuesta

Con los datos anteriormente presentados, podemos concluir que varios modelos cumplen con el principal objetivo, un modelo eficiente, sin embargo, se eligira YOLOv8n, el cual tiene un peso de tan solo

6 mb, una precisión de casi 94% y el tiempo que le tomo para predecir 134 imágenes fue de menos de 6 segundos, lo que lo transforma en un gran candidato. Para la detección se uso FastApi con la finalidad de generar un servidor que recibiera imágenes y entregara el resultado de la detección.



Fig. 6 Detección de elementos en el simulador.

## V. CONCLUSIONES

En el proyecto "Aprendiendo a ver en tiempo real" se ha logrado implementar con éxito que un DuckieBot sea capaz de movilizarse de manera autónoma en un entorno simulado, detectando y evitando obstáculos de manera eficientemente. Los objetivos principales, que incluían la detección de obstáculos mediante una CNN de bajo coste, la implementación de algoritmos de Path Planning y Motion Planning, han sido cumplidos.

En cuanto a los algoritmos de Path Planning, se implementaron y compararon Dijkstra y A\*, concluyendo que ambos tienen un uso de memoria similar (~75MB), Dijkstra presenta mejores tiempos de ejecución.

El Motion Planning se optimizó utilizando un buffer que permite al DuckieBot tomar decisiones de navegación más rápidas y

suaves. Además, se emplearon controladores PID para ajustar la velocidad y el ángulo de dirección del robot, lo cual mejoró su estabilidad y precisión en movimiento.

En el aspecto de detección, a pesar de los problemas iniciales con el modelo Bitnet, se optó por utilizar modelos populares como YOLO y RT-DETR, entrenándolos con un dataset propio. Se concluyó que YOLOv8n es el modelo más eficiente, debido a su bajo peso (6MB), alta precisión (~94%), y rápido tiempo de respuesta (menos de 6 segundos para 134 imágenes).

Finalmente, para la detección en tiempo real se utilizó FastAPI, creando un servidor que recibe imágenes y entrega los resultados de la detección, completando así la funcionalidad del sistema.