Pumas Auto Model Car; Pumatroca

Negrete, Marco [1] Robles, Héctor [1] Rosario, Omar [1] Solano, Jorge [1] Vazquez, Jair [1]

[1]Facultad de Ingeniería, UNAM

Introducción

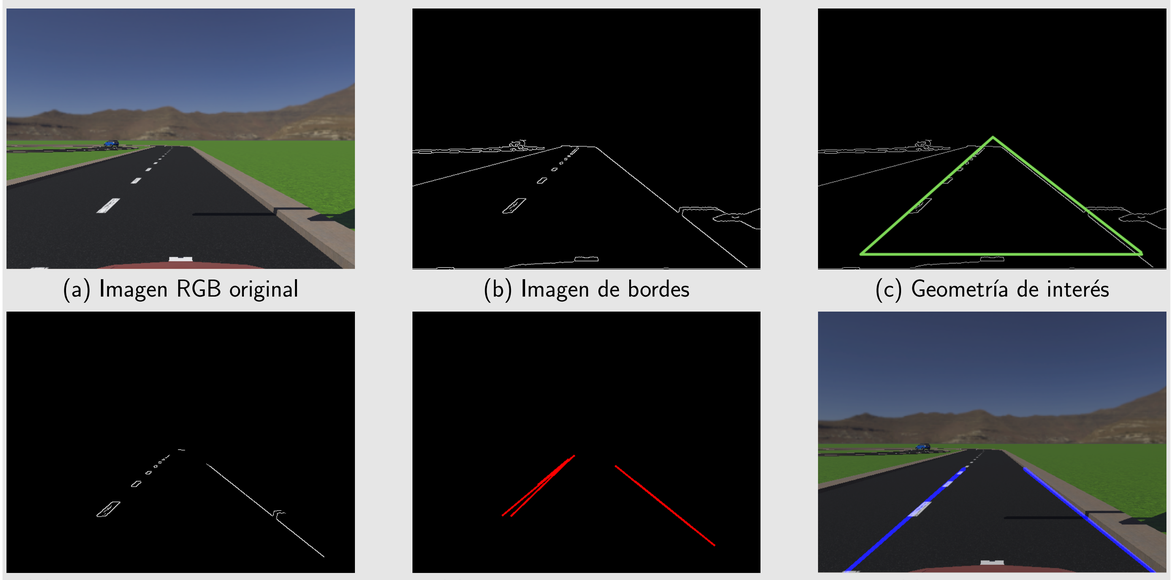
Los accidentes viales en su mayoría son provocados por errores humanos, los vehículos autónomos cuentan con el potencial de reducirlos en base a toma de decisiones bien planeadas y acciones supervisadas constantemente. Este póster contiene algunas de las múltiples herramientas teórico matemáticas necesarias para la elaboración de un sistema de navegación autónoma para vehículos sin conductor, en este caso utilizando un modelo escala 1:10, **ROS2**[Open Roboyics n.d.] y **Ubuntu Server for Raspberry Pi 22.04**[Canonical Ltd. n.d.] Con el propósito de crear este sistema se desarrollaron los siguientes módulos para tareas específicas.

- **Detección de Carril.**
- **Diseño de leyes de control.**
- **Seguimiento de Carril.**
- **Detección de Obstáculos.**
- **Evasión de Obstáculos.**
- **Detección de señales de tránsito.**

Detección de Carril

El objetivo de la detección del carril, es que el robot conozca en todo momento su posición relativa respecto al ancho del carril, lo que le servirá como parámetro de entrada para la ley de control. Proceso de **Detección de Carril**

1. Subscribirse al tópico de imagen RGB en tiempo real.
2. Recortar al área de interés (mitad inferior de la imagen).
3. Aplicar el filtro de paso banda de color en el espectro HSL.[Lorena García n.d.][W3Schools n.d.]
4. Aplicar el algoritmo Canny[OpenCV n.d.] para la detección de bordes.
5. Aplicar la transformación Probabilistic Houhg Lines[Huamán n.d.] para obtener las líneas representadas por dos puntos enformato en formato $(x_1, y_1), (x_2, y_2)$.
6. Proyectar la línea representada a una línea en formato $[\rho, \theta]$.
7. Comparar la distancia al origen (ρ) y el ángulo (θ) para saber si es una línea izquierda, derecha o ruido.
8. Promediar los valores de ρ y θ obtenidos para recalibrar el filtro.
9. Publicar los valores promedio encontrados en los tópicos $/lines/left$ y $/lines/right$ en formato $[\rho, \theta]$



(a) Imagen RGB original (b) Imagen de bordes (c) Geometría de interés

(d) Imagen de bordes delimitada (e) Bordes de carril detectados (f) Líneas promedio

Figure 1. Líneas del carril detectadas por medio del Detector de bordes de Canny y Transformada Hough

Seguimiento de Carril

[a]La velocidad se mantiene constante mientras se tengan líneas, en caso de no contar con líneas, el vehículo apaga el motor.

Los pasos para realizar el seguimiento de carril, son escencialmente los siguientes.

1. Subscribirse a la línea izquierda y derecha en formato $[\rho, \theta]$.
2. Evaluar si se cuenta con ámbas líneas, para promediarlas, o para considerar solamente la línea que se tiene.
3. Calcular un error de posicionamiento relativo al carril, utilizando la ecuación

$$steering = (k_\rho * e_\rho) + (k_\theta * e\theta)$$

Donde

$$e_\rho = \rho_{goal} - \rho_{actual}; e_\rho = \theta_{goal} - \theta_{actual}$$

Y los parámetros k_ρ y k_θ son los pesos del controlador.

4. Publicar al tópico $/steering$ [a] el valor normalizado para el ángulo.

Detección de obstáculos

La detección de obstáculos se realiza utilizando la nube de puntos obtenida por una *Intel Realsense*[Intel n.d.(a)] y técnicas tradicionales de visión computacional.

1. Se inicia la cámara utilizando la biblioteca proporcionada por *Intel*[Intel n.d.(b)].
2. Se obtiene la imagen en el tópico $/depth$.
3. Se recorta la imagen a la zona de interés (donde habitualmente se esperaría un coche).
4. Si el valor promedio pasa de cierto umbral, se publica en el tópico $/found$ un verdadero, indicando que se debe tomar una desición sobre el rebase del obstáculo.

Evasión de obstáculos

La evasión de obstáculos se realiza en idk partes, las cuales son las siguientes.

1. Pausa activa del tópico $/found$.
2. Una vez se detecta la señal activadora del tópico, se toma el control del robot.
3. El vehículo disminuye velocidad para evitar colisiones con el obstáculo, y se enciende la intermitente izquierda durante 5 ciclos, indicando que el vehículo cambiará de carril al carril izquierdo.
4. El vehículo cambia de carril al carril izquierdo.
5. El vehículo mantiene la velocidad crucero mientras hace las correcciones necesarias para poder aumentar la velocidad de forma segura.
6. El vehículo aumenta la velocidad al límite establecido para poder efectuar el rebase.
7. Una vez que el vehículo sensa que el obstáculo ha sido rebasado, enciende la intermitente derecha durante 5 ciclos y emite un sonido para advertir al vehículo que está siendo rebasado.
8. El vehículo regresa al carril derecho, y continúa a velocidad crucero.

Estacionamiento Autónomo

La forma de abordar el estacionamiento, consiste en que el vehículo considera que habrá el espacio necesario para estacionarse en batería, utilizando el siguiente algoritmo.

1. El vehículo avanza hasta encontrar un espacio libre.
2. El vehículo pone la dirección en el límite contrario a la dirección donde se encuentra el espacio.
3. El vehículo avanza hasta quedar en diagonal al lugar que encontró.
4. El vehículo pone la dirección completamente al lado contrario, y avanza en reversa hasta calcular que está bien estacionado.

Hardware

El hardware utilizado es el siguiente.

- Vehículo a escala 1:10 modelo *Redcat Lightning EPX Drift* como base.
- Mini computadora *Raspberry Pi 4* en su versión de 8GB de RAM.
- Controladora *Roboclaw* de dos canales con capacidad de 30A por canal.
- Intel Realsense.
- Regulador de voltaje *Pololu* de 5V a 12A.
- Cámara web USB marca *Logitech*.
- Baterías de *Li-Po* de dos celdas, a 5200 mAh.

Software

References

Canonical Ltd. (n.d.). *Ubuntu Server for Raspberry Pi*. Ed. by Canonical Ltd. The quickest route to everything that open source and Raspberry Pi has to offer. URL: https://ubuntu.com/raspberry-pi/server.

Huamán, A. (n.d.). *Hough Line Transform*. Ed. by OpenCV. Se utiliza la tranformación probabilística. URL: https://docs.opencv.org/4.8.0/d9/db0/tutorial_hough_lines.html.

Intel (n.d.[a]). *Tecnología Intel® RealSense™*. Ed. by Intel. URL: https://www.intel.la/content/www/xl/es/architecture-and-technology/realsense-overview.html.

Intel (n.d.[b]). *Using depth camera with Raspberry Pi 3*. Ed. by Intel. URL: https://dev.intelrealsense.com/docs/using-depth-camera-with-raspberry-pi-3.

Lorena García, R. S. (n.d.). *Thresholding Operations using inRange*. Ed. by OpenCV. URL: https://docs.opencv.org/4.8.0/da/d97/tutorial_threshold_inRange.html.

Open Roboyics (n.d.). *The Robot Operating System (ROS)*. Ed. by Open Roboyics. ROS is a set of software libraries and tools for building robot applications. URL: https://docs.ros.org/en/iron/index.html.

OpenCV (n.d.). *Canny Edge Detection*. Ed. by OpenCV. URL: https://docs.opencv.org/4.8.0/da/d22/tutorial_py_canny.html.

W3Schools (n.d.). *Colors HSL and HSLA*. Ed. by W3Schools. OpenCv reconoce el esquema de color como HLS, y no HSL. URL: https://www.w3schools.com/colors/colors_hsl.asp.

Este proyecto fue apoyado por la UNAM-DGAPA a través del proyecto PAPIIT IT102424