Tarea N° 2

Aprendiendo a detectar líneas y eliminar distorsión

Cristian Herrera Badilla
Escuela de Ingeniería, Universidad de O'Higgins
Viernes 10, Mayo, 2024

***Abstract*—El documento presenta una metodología para la detección de carriles en entornos simulados de conducción autónoma, utilizando técnicas de procesamiento de imágenes. Comienza con un marco teórico que abarca conceptos clave como la Transformada de Hough, la rectificación de cámara y la detección de bordes mediante el algoritmo de Canny. Luego, detalla una metodología paso a paso que incluye filtros de color, suavizado de imágenes, detección de bordes y transformada de Hough. Los resultados muestran una comparación entre la transformada de Hough estándar y probabilística, destacando la efectividad de esta última. El análisis subraya la importancia de un enfoque riguroso en la selección y ajuste de técnicas de procesamiento de imágenes para aplicaciones de navegación autónoma.**

I. Introducción

El presente documento aborda una serie de técnicas fundamentales en el procesamiento de imágenes aplicadas a la detección de carriles en entornos simulados de conducción autónoma. Comenzando con un sólido marco teórico que abarca conceptos como la Transformada de Hough, la rectificación de cámara, la detección de bordes mediante el algoritmo de Canny, la convolución y el uso de kernels, el documento establece una base sólida para comprender las metodologías y resultados que se presentan posteriormente.

A través de una metodología detallada, que incluye la aplicación de filtros de color, suavizado de imágenes, detección de bordes y transformada de Hough, se desarrolla un proceso paso a paso para la identificación precisa de carriles en imágenes de simuladores de conducción. La calibración de la cámara también se aborda con claridad, proporcionando una comprensión completa de cómo se ajustan y aplican los parámetros intrínsecos y extrínsecos de la cámara para obtener mediciones precisas.

Los resultados obtenidos se presentan de manera exhaustiva, incluyendo imágenes de entrada y salida de cada etapa del proceso, lo que permite una evaluación detallada de la efectividad de las técnicas empleadas. Se destaca especialmente la comparación entre la transformada de Hough estándar y probabilística, lo que proporciona información valiosa sobre cuál de las dos variantes es más adecuada para la detección de carriles en entornos simulados.

Por ultimo, se realiza un análisis detallado de los resultados, resaltando los desafíos encontrados y las conclusiones alcanzadas. Se subraya la importancia de un enfoque riguroso en la selección y ajuste de técnicas de procesamiento de imágenes, así como la calibración adecuada de los parámetros del sistema, para garantizar la robustez y eficacia de los sistemas de visión utilizados en aplicaciones como la navegación autónoma.

II. Marco Teórico

A continuación el detalle y definiciones de conceptos utilizados dentro de esta tareas.

A. Transformada de Hough

El caso más simple para la transformada de Hough es la transformación lineal para detectar líneas rectas. En el espacio de la imagen, una recta se puede representar con la ecuación $y = mx + b$, y se puede graficar para cada par (x, y) de la imagen. En la transformada de Hough, la idea principal es considerar las características de una recta en términos de sus parámetros (m, b), y no como puntos (x, y) de la imagen. Basándose en lo anterior, una recta se puede representar como un punto en el espacio de parámetros.

Sin embargo, cuando se tienen rectas verticales, los parámetros de la recta se indefinen. Por esta razón, es mejor usar los parámetros que describen una recta en coordenadas polares, denotados (ρ, θ).

El parámetro ρ representa la distancia entre el origen de coordenadas y el punto más cercano de la recta, mientras que θ es el ángulo del vector director de la recta perpendicular a la recta original y que pasa por el origen de coordenadas.

Usando esta parametrización, la ecuación de una recta se puede escribir de la siguiente forma:

$$\rho = x\cos\theta + y\sin\theta$$

que se puede reescribir como:

$$y = \left(-\frac{\cos\theta}{\sin\theta}\right)x + \frac{\rho}{\sin\theta}$$

Entonces, es posible asociar a cada recta un par (ρ, θ) que es único si $\theta \neq 0°$ o $\theta \neq 180°$. El espacio (ρ, θ) se denomina espacio de Hough para el conjunto de rectas en dos dimensiones.

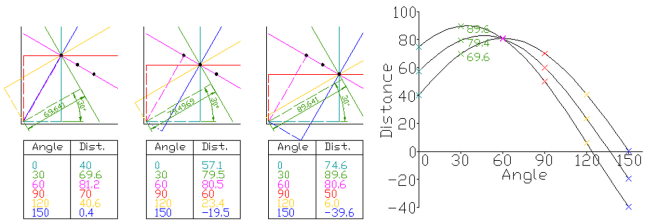

Fig. 1: Ejemplo de uso de la transformada de Hough para la detección de rectas en una imagen.

B. rectificación de cámara

Se utiliza para estimar los parámetros de una lente y el sensor de imagen de una cámara. Estos parámetros incluyen la distancia focal, el punto principal, la distorsión radial y tangencial, así como la posición y orientación relativa de la cámara respecto a un sistema de coordenadas externo.

1) **Parámetros intrínsecos**: Estos incluyen la distancia focal, el centro óptico y los coeficientes de distorsión de la lente. La matriz de cámara (una matriz 3x4) mapea los puntos del mundo 3D a la imagen 2D.
2) **Parámetros extrínsecos** Representan la posición y orientación de la cámara en la escena 3D.
3) **Distorsión de la lente** Incluye la distorsión radial y tangencial. La distorsión radial proviene de la forma de la lente, mientras que la tangencial se debe a desalineaciones en la lente y el sensor.

La calibración de la cámara se realiza típicamente utilizando un patrón conocido, como un tablero de ajedrez o una cuadrícula de puntos, que se coloca en diferentes posiciones y orientaciones en el campo de visión de la cámara. Al capturar imágenes del patrón desde diferentes ángulos y distancias, se pueden estimar los parámetros de la cámara mediante técnicas de procesamiento de imágenes y optimización.

Una vez que se han determinado con precisión los parámetros de la cámara, se pueden corregir las distorsiones presentes en las imágenes capturadas, lo que es crucial para aplicaciones de visión por computadora y robótica que requieren mediciones precisas y reconstrucciones tridimensionales del mundo real.

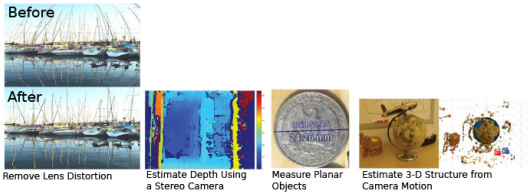

Fig. 2: Calibración de cámara

C. Canny detección de bordes

1) **Teoría de la Detección de Bordes de Canny**: La Detección de Bordes de Canny es un algoritmo de detección de bordes popular. Fue desarrollado por John F. Canny. Es un algoritmo de múltiples etapas y pasaremos por cada una de ellas.
2) **Reducción de Ruido**: Dado que la detección de bordes es susceptible al ruido en la imagen, el primer paso es eliminar el ruido en la imagen con un filtro Gaussiano de 5x5.
3) **Encontrando el Gradiente de Intensidad de la Imagen**: La imagen suavizada se filtra luego con un kernel de Sobel en ambas direcciones horizontal y vertical para obtener la primera derivada en la dirección horizontal G_x y en la dirección vertical G_y. A partir de estas dos imágenes, podemos encontrar el gradiente de borde y la dirección para cada píxel de la siguiente manera:

$$\text{Gradiente\_de\_Borde}(G) = \sqrt{G_x^2 + G_y^2}$$

$$\text{Ángulo}(\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

La dirección del gradiente siempre es perpendicular a los bordes. Se redondea a uno de los cuatro ángulos que representan las direcciones vertical, horizontal y dos diagonales[1,2].
4) **Supresión de Máximos No Máximos**: Después de obtener la magnitud del gradiente y la dirección, se realiza un escaneo completo de la imagen para eliminar cualquier píxel no deseado que pueda no constituir el borde. Para esto, en cada píxel, se verifica si es un máximo local en su vecindario en la dirección del gradiente[2].

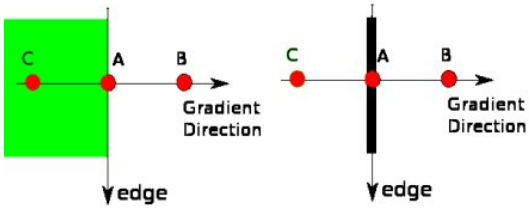

Fig. 3: Non-maximum Suppression

5) **Umbralización por Histéresis**: Esta etapa decide cuáles son realmente bordes y cuáles no. Para esto, necesitamos dos valores de umbral, minVal y maxVal. Cualquier borde con un gradiente de intensidad mayor que maxVal está seguro de ser un borde y aquellos por debajo de minVal están seguros de no serlo, por lo que se descartan. Aquellos que se encuentran entre estos dos umbrales se clasifican como bordes o no bordes basados en su conectividad. Si están conectados a píxeles de "borde seguro", se consideran parte de los bordes. De lo contrario, también se descartan.

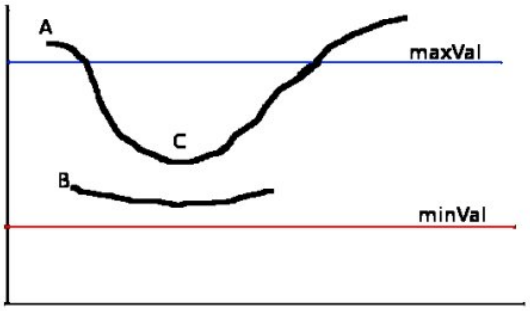

Fig. 4: Hysteresis Thresholding

D. Convolución

Función fundamental en el procesamiento de imágenes digitales, la cual realiza el filtrado de los valores de píxel de una imagen, lo que se puede utilizar para aumentar su nitidez, difuminarla, detectar sus ejes u otros realces basados en el kernel. En Python se realiza mediante bibliotecas como NumPy y OpenCV.

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) \cdot g(t - \tau)\, d\tau$$

Esta fórmula representa la convolución de dos funciones f y g, donde $*$ es el operador de convolución y $(\cdot)$ denota la multiplicación de funciones. El resultado de la convolución se obtiene integrando el producto de las dos funciones trasladadas una respecto a la otra a lo largo del eje τ.

En el contexto del procesamiento de imágenes discretas, esta fórmula se simplifica a una sumatoria sobre los índices de las matrices que representan las imágenes. La convolución discreta se realiza multiplicando elemento por elemento la imagen de entrada con el kernel y sumando los productos correspondientes.

E. kernel

Matriz cuadrada de números que actúa como una especie de "filtro" o "máscara" que se aplica a cada píxel de la imagen de entrada durante la convolución. Esta matriz puede tener diferentes tamaños, como 3x3, 5x5 o incluso más grande, y sus valores determinan cómo se realizará la transformación en la imagen.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Fig. 5: Sharpen

III. Metodología

Para identificar los carriles en el simulador, primero aplicamos mascaras que nos permitan filtrar los elementos de interés en la imagen. En este contexto, los elementos de interés son las líneas amarillas y blancas que delimitan las calles en el simulador. Para lograr esto, empleamos dos mascaras que seleccionan los colores mencionados.

Código 1: Filtro de color

```python
def filter_color(image, color):
    # Amarillo
    lower_yellow = np.array([89, 64, 138])
    upper_yellow= np.array([96, 240, 255])

    # Blanco
    lower_white = np.array([0, 0, 145])
    upper_white = np.array([179, 52, 255])

    # Create HSV Image and threshold into a range.
    yellow = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
    red = yellow.copy()
    white = yellow.copy()

    mask_yellow = cv2.inRange(yellow, lower_yellow,
     upper_yellow)
    mask_white = cv2.inRange(white, lower_white,
     upper_white)

    kernel = np.ones((5,5),np.uint8)

    if color == "yellow":
        op_morf2 = cv2.erode(mask_yellow,kernel,iterations
     = 1)
    elif color == "white":
        op_morf2 = cv2.erode(mask_white,kernel,iterations =
      1)
    else:
        op_morf2 = cv2.erode(image,kernel,iterations = 1)

    return cv2.dilate(op_morf2,kernel,iterations = 2)
```

Una vez aplicadas las mascaras, utilizamos un filtro gaussiano adicional para suavizar la imagen y eliminar posibles imperfecciones. Este paso es crucial para garantizar una detección precisa de los bordes en la siguiente etapa del proceso.

El siguiente paso implica la detección de bordes en la imagen filtrada. Para lograr esto, empleamos la función Canny, que es una técnica ampliamente utilizada para detectar bordes en imágenes. Los parámetros de la función Canny se ajustan de acuerdo con las características específicas de las imágenes y el entorno del simulador. El resultado de esta operación es una imagen resaltando los bordes, lo que facilita el procesamiento de los objetos de interés, en este caso, las líneas del carril.

Código 2: Filtro gaussiano y canny

```python
def detect_edges(image, umbral_low, pr):
  umbral_high = umbral_low*pr
  image_gauss = cv.GaussianBlur(image,(5,5),1)
  edges = cv.Canny(image_gauss, umbral_low, umbral_high,
     apertureSize = 3)
  return edges
```

Finalmente, utilizamos la transformada de Hough para identificar y extraer las líneas rectas de los elementos de interés en la imagen. Esta transformada nos permite representar las líneas en forma paramétrica, lo que facilita su identificación y análisis. Posteriormente, utilizando los colores correspondientes, trazamos las líneas resultantes sobre la imagen original, lo que nos proporciona una representación visual de los carriles captados por el Duckiebot.

Código 3: Transformada de Hough estandar

```python
def hough(edges, image):
  image_hough = cv.cvtColor(image, cv.COLOR_GRAY2RGB)
  lines = cv.HoughLines(edges, 1, np.pi/180, 140, None)
  if lines is not None:
      for rho, theta in lines[:, 0]:
          a = np.cos(theta)
          b = np.sin(theta)
          x0 = a * rho
          y0 = b * rho
          x1 = int(x0 + 1000 * (-b))
```

```
            y1 = int(y0 + 1000 * (a))
            x2 = int(x0 - 1000 * (-b))
            y2 = int(y0 - 1000 * (a))
            cv.line(image_hough, (x1, y1), (x2, y2), (0, 255, 0), 3)
    return image_hough
```

Es importante destacar que durante el proceso se evaluaron tanto la versión estándar como la probabilística de la transformada de Hough con el fin de comparar los resultados obtenidos y seleccionar la opción más adecuada para el propósito específico de la detección de carriles en el simulador. Esta comparación nos permite garantizar la precisión y robustez del sistema de detección de carriles implementado.

Código 4: Transformada de Hough probabilística

```
def hough_pro(edges, image):
    image_hough_P = cv.cvtColor(image, cv.COLOR_GRAY2RGB)
    linesP = cv.HoughLinesP(edges, 1, np.pi/180, 70, None, 50, 10)
    if linesP is not None:
        for i in range(0, len(linesP)):
            l = linesP[i][0]
            cv.line(image_hough_P, (l[0], l[1]), (l[2], l[3]), (0,0,255), 3, cv.LINE_AA)
    return image_hough_P
```

Para la calibración de la cámara se utiliza un patrón conocido del tablero de ajedrez, se generan las coordenadas iniciales 3d de la cámara, las cuales seran usadas para encontrar las esquinas en la imagen con la función "cv2.findChessboardCorners", los puntos encontrados son guardados en dos arreglos, uno del objeto de interés y otro de la imagen. Como paso final se utiliza la funcion "cv2.calibrateCamera".

Código 5: Calibración de la cámara

```
def camera_global(image):
    pattern_size = (9, 6)
    square_size = 0.025
    object_points = np.zeros((pattern_size[0] * pattern_size[1], 3), np.float32)
    object_points[:, :2] = np.mgrid[0:pattern_size[0], 0:pattern_size[1]].T.reshape(-1, 2) * square_size

    object_points_list = []
    image_points_list = []

    image_files = glob.glob(image)
    for image_file in image_files:
        image = cv.imread(image_file)
        gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
        found, corners = cv.findChessboardCorners(gray, pattern_size, None)

        if found:
            object_points_list.append(object_points)
            image_points_list.append(corners)

    ret, camera_matrix, distortion_coeffs, rvecs, tvecs = cv.calibrateCamera(
        object_points_list, image_points_list, gray.shape[::-1], None, None)
```

IV. Resultados

Transformada de Hough

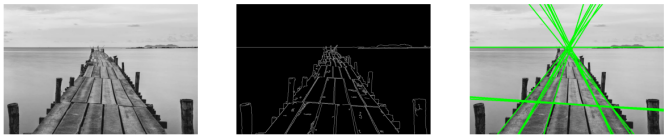

Fig. 6: Transformada de Hough estándar imagen: puente

Fig. 7: Transformada de Hough estándar imagen: vías de tren

Fig. 8: Transformada de Hough probabilística imagen: puente

Fig. 9: Transformada de Hough probabilística imagen: vías de tren

Calibración de cámara

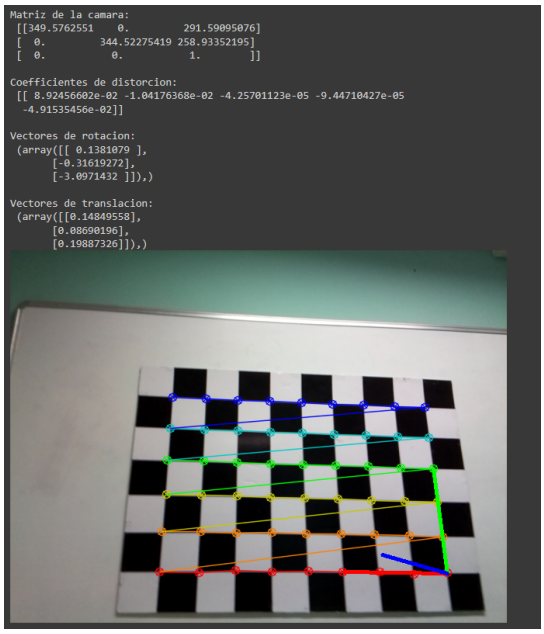

Fig. 10: Calibración de cámara: imagen 10

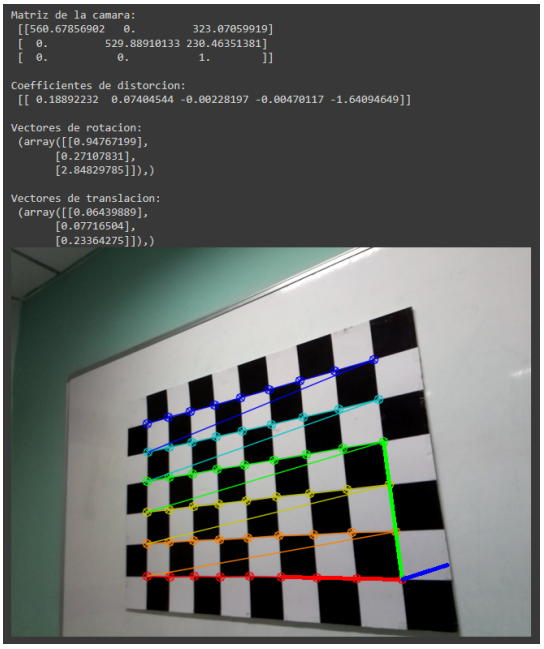

Fig. 11: Calibración de cámara: imagen 11

Gym-Duckietown

Fig. 12: Gym-Duckietown imagen 1

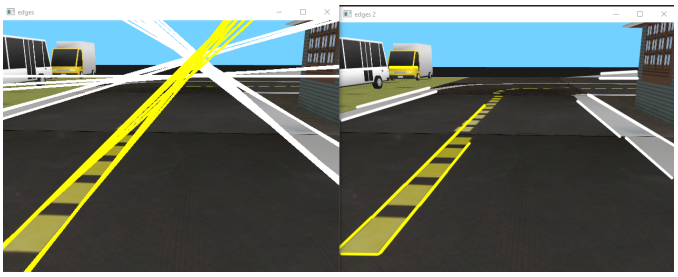

Fig. 13: Gym-Duckietown imagen 2

V. Análisis

Los resultados obtenidos tras la implementación y evaluación de las técnicas de procesamiento de imágenes revelan una serie de hallazgos significativos y desafíos inherentes al proceso de detección de carriles en entornos simulados de conducción autónoma.

Uno de los aspectos más destacados es la comparación entre la transformada de Hough estándar y la probabilística. Si bien ambas variantes ofrecen resultados satisfactorios, la transformada probabilística se destaca por su capacidad para evitar la saturación de la imagen con líneas detectadas. Esto se traduce en una detección más precisa y menos propensa a errores, incluso en entornos con alta variabilidad y presencia de ruido. Este hallazgo resalta la importancia de elegir cuidadosamente la técnica más adecuada para cada aplicación específica, considerando factores como la complejidad del entorno y la presencia de posibles fuentes de interferencia.

Además, el análisis revela la influencia crucial de los parámetros ajustables en cada etapa del proceso de detección de carriles. Desde la selección de umbrales en la detección de bordes hasta los intervalos de votación en la transformada de Hough, cada ajuste tiene un impacto directo en la precisión y robustez del sistema. Esto destaca la necesidad de un enfoque iterativo y experimental para encontrar la combinación óptima de parámetros que maximice el rendimiento del sistema en diferentes condiciones y entornos.

Por otro lado, la calibración precisa de la cámara emerge como un componente crítico en el proceso de detección de carriles. La capacidad de la cámara para capturar imágenes con una perspectiva y distorsión consistentes es fundamental para garantizar mediciones precisas y una detección confiable de los carriles en el entorno simulado. Los resultados de la calibración proporcionan una base sólida para la transformación de coordenadas y la corrección de distorsiones, lo que contribuye significativamente a la precisión y estabilidad del sistema.

VI. Conclusiones Generales

Basándonos en los resultados obtenidos al comparar las variantes estándar y probabilística de la transformada de Hough, se llega a la conclusión de que la utilización de la versión probabilística ofrece ventajas significativas. Esta variante demuestra ser más efectiva al evitar la saturación de la imagen con líneas detectadas, manteniendo un alto nivel de precisión. Este hallazgo abre la puerta a futuras exploraciones y refinamientos en la aplicación de esta técnica, con el objetivo de mejorar aún más la detección de líneas en entornos con alta variabilidad y presencia de ruido.

En términos generales, los experimentos realizados han arrojado resultados satisfactorios, permitiendo una detección precisa de líneas en el simulador. Este éxito subraya la importancia de un enfoque riguroso en la selección y ajuste de técnicas de procesamiento de imágenes, así como la calibración adecuada de los parámetros del sistema. Estos pasos son fundamentales para garantizar la robustez y eficacia de los sistemas de visión utilizados en aplicaciones como la navegación autónoma.

References

[1] P. Corke, *Robotics, Vision and Control*. Springer, 2011.
[2] G. Dudek and M. Jenkin, *Computational Principles of Mobile Robotics*. Springer, 2011.
[3] A. I. A. Mahmoud Hassaballah, *Deep Learning in Computer Vision: Principles and Applications*. Editorial, 2019/2020.
[4] R. R. Murphy, *Introduction to AI Robotics*. MIT Press, 2019.
[5] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, 2005.