Tarea N°2:

Aprendiendo a detectar líneas y eliminar distorsión

Cristóbal Tomás David Alberto Lagos Valtierra
Escuela de Ingeniería, Universidad de O'Higgins
13 de Mayo del 2024

***Abstract*—Este informe presenta un estudio sobre la detección de líneas en imágenes mediante técnicas como el algoritmo de Canny y la transformada de Hough. Además, se aborda el proceso de calibración de cámaras para mejorar la precisión de la detección. También se analiza la aplicación de la detección de vías en el entorno de simulación Gym-Duckietown, explorando su utilidad en la navegación autónoma y la comprensión de escenarios urbanos. Este paper proporciona una visión de las técnicas y aplicaciones clave en el campo de la visión por computadora y la robótica.**

I. Introducción

La detección de líneas en imágenes es un componente esencial en numerosas aplicaciones de visión por computadora y robótica móvil. Hoy en día, vemos aplicaciones como la navegación autónoma, el análisis de imágenes médicas e incluso en la Realidad Aumentada. La capacidad de identificar y seguir líneas en entornos visuales complejos ha sido un área de investigación activa durante décadas, y resulta realmente emocionante que podamos ver las distintas aplicaciones que han surgido con los años. En este informe, nos enfocaremos en dos técnicas fundamentales para esta área: el algoritmo de Canny y la transformada de Hough.

El algoritmo de Canny es ampliamente reconocido por su capacidad para detectar bordes en imágenes con alta precisión y bajo nivel de ruido. Por otro lado, la transformada de Hough se utiliza para identificar formas geométricas, como líneas rectas, mediante el análisis de las características de la imagen. Exploraremos cómo estas técnicas pueden combinarse para lograr una detección de líneas en una variedad de contextos.

Además, abordaremos otro procesos que involucra la detección de linea, la Calibración de Cámaras. Una correcta calibración es clave para garantizar la precisión de la detección y la correspondencia entre las coordenadas de imagen y el mundo real.

Finalmente, examinaremos una aplicación práctica de la detección de líneas en el entorno de simulación Gym-Duckietown. Exploraremos cómo la detección de vías puede ser utilizada en este contexto para facilitar la navegación precisa y segura de vehículos autónomos.

II. Marco Teórico

A. Detector de lineas

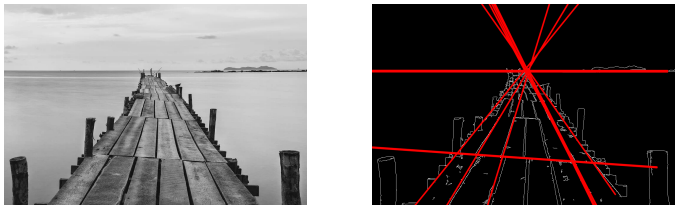
Un detector de líneas es un algoritmo diseñado para identificar líneas rectas o curvas en una imagen digital. Este proceso se lleva a cabo mediante el análisis de los cambios abruptos de intensidad en los píxeles de la imagen, ya que las líneas se caracterizan por tener una fuerte variación en la intensidad de los píxeles a lo largo de su longitud.

Uno de los métodos más comunes para la detección de líneas es el algoritmo de Canny, que opera en varias etapas. Primero, se aplica un filtro de suavizado para reducir el ruido en la imagen. Luego, se calcula el gradiente de intensidad de la imagen para identificar los bordes potenciales. Después, se aplica un umbral para seleccionar los bordes más relevantes. Finalmente, se utiliza la técnica de seguimiento de bordes para conectar píxeles adyacentes y formar líneas continuas.

(a) Imagen sin filtro Canny (b) Imagen con filtro Canny

Fig. 1: Filtro Canny

Otro enfoque ampliamente utilizado es la transformada de Hough, que busca identificar líneas en una imagen mediante la detección de puntos que forman parte de esas líneas. Esta técnica transforma los puntos de la imagen en parámetros de una representación espacial, donde las líneas se convierten en puntos en este espacio. Luego, se realiza un proceso de acumulación para identificar los puntos que corresponden a líneas en la imagen original.



(a) Imagen sin Transformada Hough (b) Imagen con Transformada de Hough

Fig. 2: Transformada de Hough

En adición, existe una variante para la Transformada de Hough llamada Transformada de Hough probabilística. La diferencia principalmente radica es que en esta ultima se examina todos los posibles pares de puntos para identificar líneas selecciona aleatoriamente solo un subconjunto de puntos y realiza un muestreo probabilístico. Este enfoque reduce

significativamente el tiempo de cómputo y mejora la eficiencia del algoritmo, haciéndolo más adecuado para aplicaciones en tiempo real. Además, la transformada de Hough probabilística puede manejar eficazmente imágenes con alto ruido o con líneas superpuestas, lo que la hace especialmente útil en entornos complejos.

B. Calibración de Cámara

La calibración de cámaras es un proceso esencial en el campo de la visión por computadora que se utiliza para determinar los parámetros intrínsecos y extrínsecos de una cámara. Los parámetros intrínsecos incluyen características como la distancia focal, la distorsión radial y tangencial, y el centro óptico de la cámara, mientras que los parámetros extrínsecos describen la posición y orientación relativas de la cámara respecto a un sistema de coordenadas del mundo real.

El objetivo principal de la calibración de cámaras es establecer una relación geométrica precisa entre los puntos en un mundo tridimensional y sus proyecciones en una imagen bidimensional capturada por la cámara. Este proceso es fundamental para garantizar que las mediciones realizadas en la imagen se correspondan de manera precisa con las dimensiones y ubicaciones reales en el mundo físico.

El proceso de calibración generalmente implica la captura de imágenes de un patrón conocido, como una cuadrícula de puntos o un tablero de ajedrez, desde diferentes ángulos y posiciones. Estas imágenes se utilizan luego para estimar los parámetros de la cámara utilizando algoritmos.

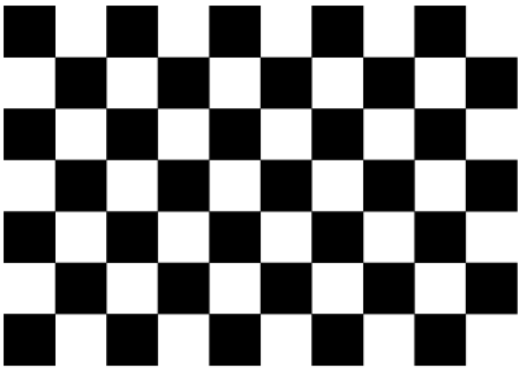


Fig. 3: Patrón de tablero de ajedrez usualmente utilizado para la calibración de cámaras.

Una vez que se han determinado los parámetros de la cámara, se pueden corregir las distorsiones geométricas presentes en las imágenes capturadas y proyectar puntos en el mundo real en la imagen con una alta precisión.

C. Gym-duckietown

Gym-Duckietown es un entorno de simulación desarrollado como parte del proyecto Duckietown, este tiene como objetivo proporcionar un entorno de aprendizaje para la visión computacional, robótica autónoma y la inteligencia artificial.

Se puede simular y desarrollar algoritmos de control para vehículos autónomos llamados ”Duckiebots” que navegan por un entorno urbano virtual. El entorno de simulación corresponde a una ciudad con carriles, intersecciones, señales de tráfico y los patitos ”duckies” que se utilizan como referencia visual y objetivos en los desafíos de navegación.

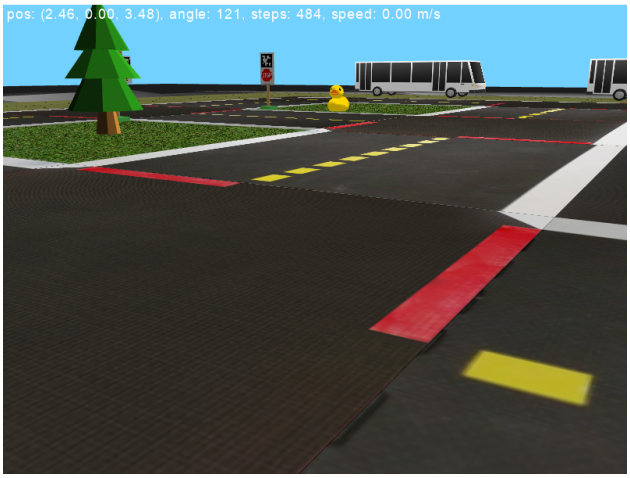


Fig. 4: Gym-duckietown.

III. Metodología

Primero partiremos viendo la aplicación del filtro Canny. En primer lugar leemos la imagen y luego la pasamos a escala de Grises ya que es necesario para que Canny pueda funcionar correctamente. Una vez pasada la imagen al espacio de color requerido, fijamos el lower threshold y upper threshold en una razón 1:3 y con estos parametros, aplicamos el filtro de Canny a nuestra imagen.

```
    # Convertir la imagen a escala de grises
    gray_image = cv2.cvtColor(imagen_cv2, cv2.
     COLOR_BGR2GRAY)

    # Aplicar el detector de Canny
    lower_threshold = 160
    upper_threshold = 480
    edges = cv2.Canny(gray_image, lower_threshold,
     upper_threshold)

    # Mostrar la imagen resultante con los bordes
     detectados
    cv2_imshow(gray_image)
    cv2_imshow(edges)
```

Ahora aplicamos la transformada de Hough, primero calculamos estas llamando a la función *cv2.Houghlines*, teniendo estas las dibujamos las lineas resultantes en la imagen de la cual se calcularon.

```
# Aplicar la transformada de Hough para detectar lineas
     rectas
lines = cv2.HoughLines(edges, 1, np.pi/180, threshold=100)

# Dibujar las lineas detectadas en la imagen original
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))
        cv2.line(imagen_cv2, (x1, y1), (x2, y2), (0, 0,
     255), 2)
```

Ahora procedemos a la calibración de una cámara. Para esto, tomamos como referencia un total de 7 imagenes y a partir de estas se va a proceder a la calibración de la cámara. Partimos definiendo las coordenadas del mundo real de los puntos 3D del tablero de ajedrez ya que estas se utilizarán como referencia para calibrar la cámara. Luego por cada imagen cargada la pasamos a escala de grises, se lee y utilizando la función *cv2.findChessboardCorners()*, encontramos las esquinas del tablero de ajedrez en la imagen. En caso de que las esquinas se encuentren correctamente, se agregan los puntos 3D y 2D correspondientes a las listas *objpoints* y *imgpoints*. Una vez que se han procesado todas las imágenes y se han recopilado los puntos 3D y 2D, se utiliza la función cv2.calibrateCamera() para calcular los parámetros intrínsecos y extrínsecos de la cámara. Por último, se imprimen los resultados de la calibración, siendo estos la matriz de la cámara (mtx), los coeficientes de distorsión (dist), los vectores de rotación (rvecs) y los vectores de traslación (tvecs).

```
import glob
criteria = (cv2.TERM_CRITERIA_EPS + cv2.
    TERM_CRITERIA_MAX_ITER, 30, 0.001)

# Definir las coordenadas del mundo real de los puntos 3D
objp = np.zeros((6*9, 3), np.float32)
objp[:, :2] = np.mgrid[0:9, 0:6].T.reshape(-1, 2)

# Arrays para almacenar los puntos 3D y 2D de todas las
    imagenes
objpoints = [] # puntos 3D en el espacio del mundo
imgpoints = [] # puntos 2D en el plano de la imagen

# Cargar las imagenes
images = glob.glob('image_*.jpg')

for fname in images:
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Encontrar las esquinas del tablero de ajedrez
    ret, corners = cv2.findChessboardCorners(gray, (9, 6),
     None)

    # Si se encuentran las esquinas, agregar puntos 3D y 2D
    if ret == True:
        objpoints.append(objp)
        corners2 = cv2.cornerSubPix(gray, corners, (11, 11)
    , (-1, -1), criteria)
        imgpoints.append(corners2)

# Calibrar la camara
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(
    objpoints, imgpoints, gray.shape[::-1], None, None)

# Mostrar los resultados
print("Matriz de la camara:\n", mtx)
print("Coeficientes de distorsion:\n", dist)
print("Vectores de rotacion:\n", rvecs)
print("Vectores de traslacion:\n", tvecs)
```

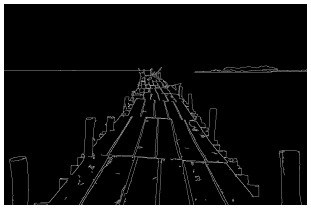
Ahora aplicaremos lo aprendido respecto al filtro Canny y Transformada de Hough en el entorno gym-duckietown. Lo que haremos será realizar una detección de lineas blancas de los caminos aplicando el filtro Canny, en el caso de las lineas amarillas aplicaremos la transformada de Hough. Lo anterior lo realizamos estableciendo valores hsv tanto para detectar el color amarillo como el color blanco. Una vez encontrado estos valores aplicamos 2 mascaras para quedarnos con lo que nos interesa y poder asignarlo a 2 variables distintas. Aplicamos transformaciones morfológicas erosionando y dilatando lo encontrado en ambas máscaras. Realizamos un filtro Canny para las lineas blancas y con el objetivo que se vea mejor la detección de estas, estas misma lineas las dilatamos para que se vean mas anchas en el resultado final. Para las lineas amarillas partimos realizando un filtro Canny y luego, aplicamos una transformada de Hough probabilística, las lineas resultantes las dibujamos en nuestro resultado final que será en la variable obs, así como lo encontrado con el filtro Canny en el caso de las lineas blancas. La implementación de lo anterior se muestra a continuación:

```
    # Seteamos los valores para mascara blanca
    lower_blanco = np.array([0, 0, 145])
    upper_blanco = np.array([160, 42, 209])

    # Seteamos los valores para mascara amarilla
    lower_amarillo = np.array([85, 50, 158])
    upper_amarillo = np.array([95, 238, 255])

    hsv = cv2.cvtColor(obs, cv2.COLOR_RGB2HSV)
    mask_blanco = cv2.inRange(hsv, lower_blanco,
     upper_blanco)
    mask_amarillo = cv2.inRange(hsv, lower_amarillo,
     upper_amarillo)

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

    # Aplicamos operaciones morfologicas a blanco
    op_morf_blanco = cv2.erode(mask_blanco,kernel,
     iterations = 1)
    op_morf_blanco = cv2.dilate(op_morf_blanco,kernel,
     iterations = 2)

    # Aplicamos operaciones morfologicas a amarillo
    op_morf_amarillo = cv2.erode(mask_amarillo,kernel,
     iterations = 2)
    op_morf_amarillo = cv2.dilate(op_morf_amarillo,kernel,
     iterations = 2)

    # Filtro Canny en lineas blancas
    bordes_canny_blanco = cv2.Canny(op_morf_blanco, 180,
     540)

    bordes_canny_blanco = cv2.cvtColor(bordes_canny_blanco,
      cv2.COLOR_GRAY2BGR)
    # Definir el kernel para la dilatacion
    kernel_dilatacion = np.ones((3, 3), np.uint8)

    # Aplicar la dilatacion a las lineas blancas
    bordes_canny_blanco_dilatadas = cv2.dilate(
     bordes_canny_blanco, kernel_dilatacion, iterations=1)

    # Filtro Canny en lineas amarillas
    bordes_canny_amarillo = cv2.Canny(op_morf_amarillo,
     180, 540)

    # Transformada de Hough probabilistica
    linesP = cv2.HoughLinesP(bordes_canny_amarillo, 1, np.
     pi / 180, 5, None, 50, 40)
    if linesP is not None:
        for i in range(0, len(linesP)):
            l = linesP[i][0]
            cv2.line(obs, (l[0], l[1]), (l[2], l[3]),
     (0,255,255), 3, cv2.LINE_AA)

    # Unir los resultados en obs
    obs = cv2.bitwise_or(obs,bordes_canny_blanco_dilatadas)

    # Mostramos las imagenes generadas
    cv2.imshow('Detector de lineas',obs)
    cv2.waitKey(1)

    env.render()
```

IV. Resultados

A continuación se muestra el resultado del filtro de Canny implementado:

(a) Imagen sin filtro Canny

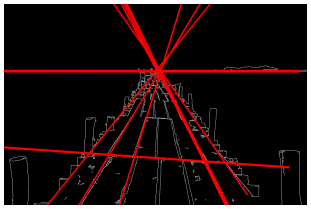


(b) Imagen con filtro Canny

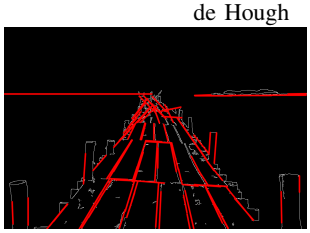
Fig. 5: Filtro Canny

A la misma imagen, también realizamos la transformada de Hough:

(a) Imagen sin Transformada Hough



(b) Imagen con Transformada de Hough



(c) Imagen con Transformada de Hough probabilística

Ahora, veremos los parámetros resultantes de la calibración de la Cámara:

1) Matriz de la camara:

```
Matriz de la camara:
 [[503.79080737   0.         313.80573857]
 [  0.         503.49847433 243.30923266]
 [  0.           0.           1.        ]]
```

2) Coeficientes de distorsión:

```
[[ 2.08365581e-01
-4.69764652e-01
4.68869442e-04
-1.85580049e-03
2.40400543e-01]]
```

3) Vectores de rotación:

```
(array([[0.2420272 ],
[0.54454108],
[3.02683017]]), array([[-1.44764591],
[-0.41678462],
[ 2.56185534]]), array([[-1.08333243],
[-0.30258099],
[ 2.80151941]]), array([[-0.01741583],
[ 0.45665827],
[ 3.10949622]]), array([[0.06814018],
[0.06533316],
[3.13231322]]), array([[0.86142692],
[0.21765203],
[2.88427839]]), array([[ 0.11151155],
[-0.4797971 ],
[-3.08461618]]), array([[0.04590726],
[0.04908939],
[3.13758249]]), array([[0.16238616],
[0.58855646],
[3.03791376]]), array([[-1.23758905],
[ 0.11541846],
[ 2.85936117]]), array([[ 1.01803658],
[-0.63564746],
[ 2.88555805]]), array([[0.7008223 ],
[0.40659524],
[2.9167126 ]]), array([[0.83041618],
[0.29081022],
[2.80959092]]), array([[0.07075553],
[0.10165738],
[3.11093461]]), array([[0.03546714],
[0.0375473 ],
[3.13719129]]), array([[ 0.78297824],
[ 0.1299104 ],
[-2.97496742]]), array([[0.10289138],
[0.53004937],
[3.07399717]]), array([[-0.10548119],
[ 0.20861923],
[ 3.08538381]]), array([[-1.25497135],
[ 0.24205658],
[-2.80864839]]), array([[0.13761815],
[0.38758047],
[3.07584061]]), array([[-0.34857002],
[-0.56601436],
[ 3.00819923]]), array([[-0.17746687],
[-0.5613588 ],
[-3.08323711]]), array([[-0.46045145],
[ 0.34077159],
[ 3.06892786]]), array([[-0.277364  ],
[ 0.18547962],
[ 3.08169962]]), array([[ 0.69227345],
[-0.52584249],
[ 2.98416726]]), array([[ 0.15739428],
[-0.45640187],
[-3.08392025]]), array([[0.95566419],
[0.08616128],
[2.86370907]]), array([[-1.12792057],
[-0.33309789],
[ 2.76377184]]), array([[1.05826031],
[0.2448358 ],
[2.79168936]]), array([[ 0.7466414 ],
[-0.13693699],
[-3.04715229]]), array([[-1.35492933],
[ 0.57614516],
[-2.71831252]]))
```

4) Vectores de traslación:

```
(array([[ 3.63551227],
[ 2.65716363],
[12.26190425]]), array([[ 0.98680855],
[ 0.26198907],
[15.76362047]]), array([[ 3.65998485],
[ 1.53534461],
[14.07769719]]), array([[ 2.75176613],
[ 3.52663565],
[11.26376686]]), array([[4.23207102],
[2.5434374 ],
[9.13477405]]), array([[3.16344565],
[2.19861228],
[8.6072909 ]]), array([[ 5.07599793],
[ 3.90843393],
[11.85196164]]), array([[3.96866435],
[2.32238253],
[8.42086826]]), array([[3.69437611],
[2.39029192],
[8.65405572]]), array([[ 2.41048519],
[ 2.29178697],
[14.2553933 ]]), array([[1.81519182],
[2.4815682 ],
[9.49614501]]), array([[1.91517812],
[2.33676615],
[7.68093071]]), array([[2.56964887],
[1.64265826],
[7.66835687]]), array([[3.85064209],
[2.40460168],
[8.16166835]]), array([[3.75914372],
[2.08748981],
[8.10089115]]), array([[ 4.68122438],
[ 2.34563126],
[15.19834463]]), array([[3.19565016],
[2.18348677],
[9.49873111]]), array([[ 3.25700998],
[ 2.5669968 ],
[18.58872642]]), array([[1.59543539],
```

```
       [2.52543273],
       [8.98988106]]), array([[3.60002551],
       [2.58725842],
       [8.5968029 ]]), array([[ 4.91560813],
       [ 2.1004736 ],
       [11.81231151]]), array([[4.48179066],
       [1.79320424],
       [9.51646054]]), array([[ 2.46767795],
       [ 2.55087566],
       [13.81499491]]), array([[ 5.99265572],
       [ 2.25119701],
       [13.45726271]]), array([[3.37510679],
       [2.64966657],
       [8.77701529]]), array([[ 5.45437463],
       [ 3.77474143],
       [11.33298086]]), array([[3.54095848],
       [2.47275117],
       [9.00860169]]), array([[ 3.14199463],
       [ 1.0163129 ],
       [14.36271313]]), array([[1.07418758],
       [2.39583123],
       [9.53980188]]), array([[ 5.52860291],
       [ 2.56417688],
       [14.01726448]]), array([[2.7560501 ],
       [2.59034602],
       [9.14410681]]))
```

Por último, mostramos lo resultante de la aplicación del filtro Canny y Transformada de Hough en el entorno de Gym-duckietown:

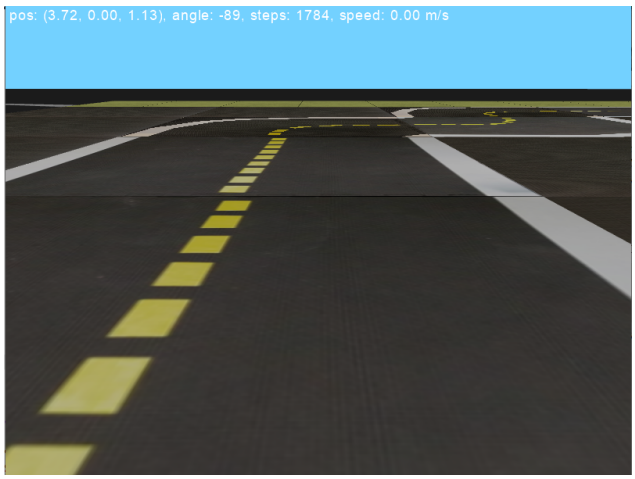


Fig. 7: Gym-duckietown sin filtros.

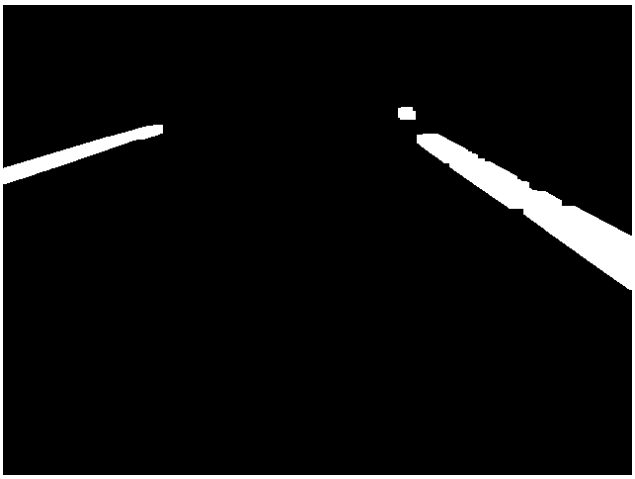


Fig. 8: Operaciones morfológicas lineas blancas.

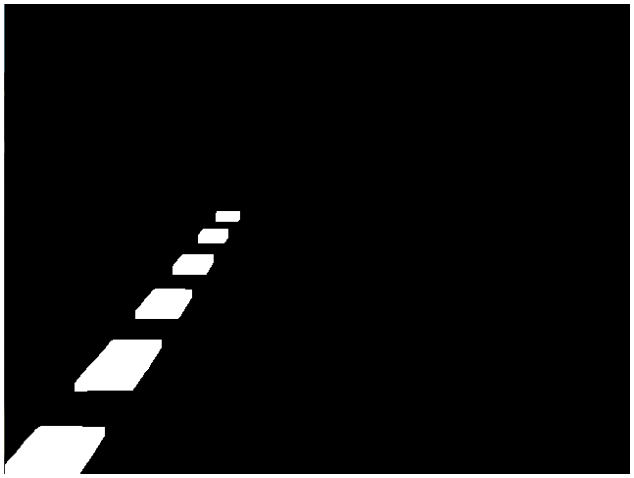


Fig. 9: Operaciones morfológicas lineas amarillas.

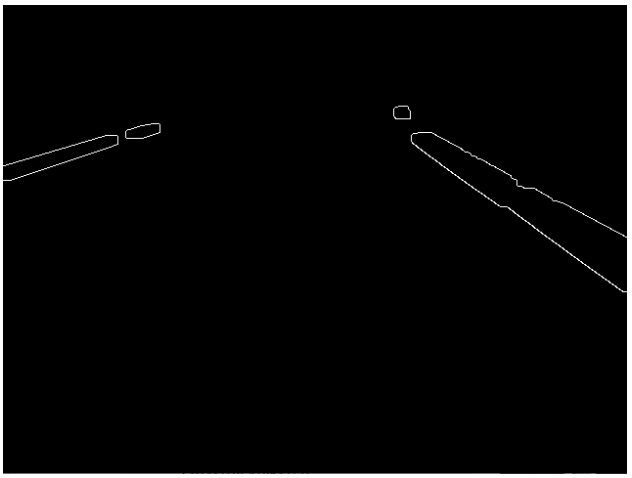


Fig. 10: Filtro Canny lineas blancas.

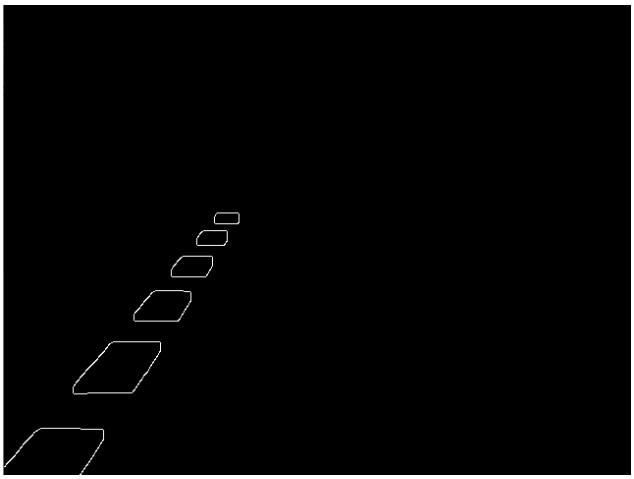


Fig. 11: Filtro Canny lineas amarillas.

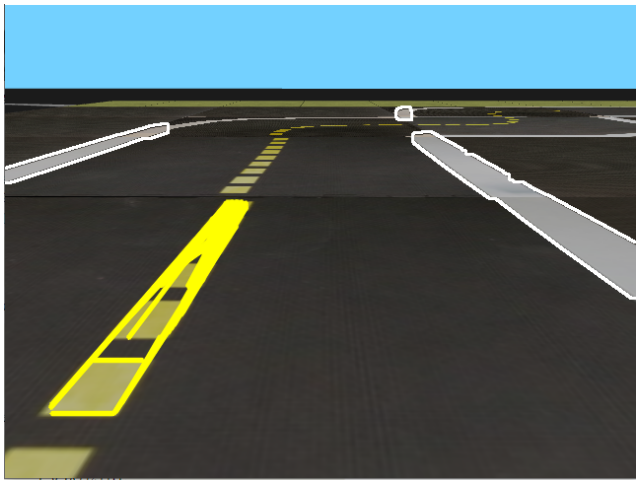


Fig. 12: Gym-duckietown con filtros aplicados.

V. Análisis

Para una correcta aplicación del filtro Canny en imágenes resultó crucial una correcta elección del lower y upper threshold, antes de llegar a los valores óptimos de estos, las imagenes resultantes poseian un exceso de contornos que resultaban en una detección poco precisa de los bordes.

La combinación este filtro y la transformada de Hough resultan ser herramientas primordiales para la detección precisa de líneas en imágenes. Se evidencia la importancia de utilizar el filtro Canny como paso inicial para resaltar los bordes. Esta etapa resulta crucial para preparar la imagen antes de aplicar la transformada de Hough ya que una vez se ha aplicado el filtro Canny, la transformada de Hough entra en juego para identificar las líneas en la imagen.

La aplicación conjunta del filtro Canny y la transformada de Hough posee ventajas. En primer lugar, esta combinación permite una detección de líneas en presencia de ruido y variaciones en la iluminación, lo que la hace adecuada para una amplia gama de aplicaciones en distintos entornos. En adición, la transformada de Hough es capaz de detectar líneas con diferentes orientaciones y longitudes, lo que la convierte en una herramienta realmente versátil para la detección de formas en imágenes complejas.

Gracias a lo anterior, es que se pudo llevar a cabo la detección de las lineas en el entorno de Gym-duckietown, evidenciando la clase de aplicaciones que se pueden lograr haciendo uso de estas técnicas de procesamiento de imágenes.

Notar que sin la correcta aplicación y utilización de máscaras con sus posteriores transformaciones morfológicas no podríamos haber llevado a cabo la aplicación de los anteriores filtros.

VI. Conclusiones

La combinación del filtro Canny y la transformada de Hough se reveló como una estrategia fundamental para la detección precisa de líneas en imágenes. El uso del filtro Canny como paso inicial para resaltar los bordes prepara la imagen de manera óptima para la posterior aplicación de la transformada de Hough, que identifica las líneas de manera eficiente.

Esta aplicación conjunta proporciona ventajas significativas, ya que permite la detección de líneas incluso en condiciones de ruido y variaciones en la iluminación, lo que la hace adecuada para una amplia gama de entornos y aplicaciones.

La detección exitosa de líneas en el entorno de Gym-Duckietown demuestra el potencial de estas técnicas de procesamiento de imágenes en aplicaciones prácticas. Siendo estas fundamentales en la utilización de aplicaciones relacionadas con la conducción autónoma.

En este trabajo se evidencia la importancia de comprender y aplicar de manera adecuada las técnicas de procesamiento de imágenes para lograr resultados precisos y efectivos en la detección de líneas, abriendo las puertas a una amplia variedad de aplicaciones no solamente en la conducción autónoma sino que en un gran diversidad de campos.

VII. Bibliografia

1) https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html