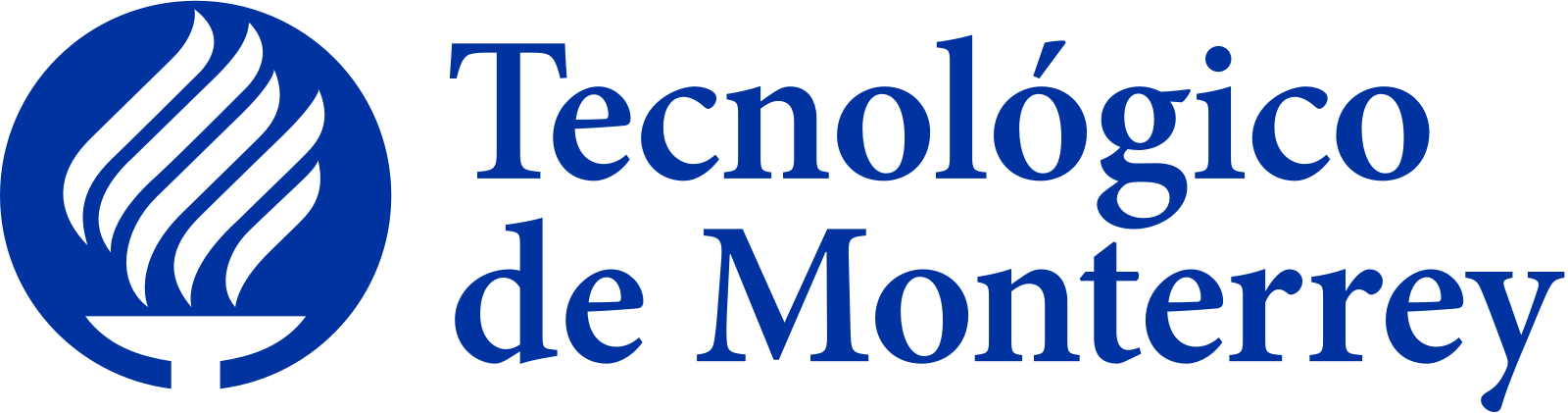
****

**Tecnológico de Monterrey**

**Campus Puebla**

**Visión para Robots**

**Febrero - Junio 20’**

**Sun-Founder Vehículo Autónomo**

**Sistema de Visión por Computadora**

**Con OpenCV**

Adrián Viveros Luques

A01551920

**Profesor**

Cesar Torres Huitzil

**Abstract**

**El siguiente documento presentan un vehículo autónomo capaz de completar una trayectoria sin la manipulación de un usuario. Este sistema embebido cuenta con una Raspberry Pi 3 como tarjeta de procesamiento en la cual se ejecuta el sistema de reconocimiento que guía al vehículo. En este reporte se presentan los puntos principales durante el desarrollo de sistema de visión por computadora del vehículo autónomo.**

**Introducción**

En la actualidad existe un concurso de vehículos llamado "The F1/tenth Autonomous Grande Prix" en el cual diferentes equipos presentan su desarrollo de un vehículo autónomo para completar un circuito en el menor tiempo posible; este concurso es una adaptación de la original Formula 1 pero a menor escala.



*Logo de la competencia f1/tenth*

Este modelo de competencia puede ser un método de impulso para la educación y desarrollo de tecnología automotriz, sin embargo, uno de los principales obstáculos es el alto costo de los materiales y recursos que componen al sistema, lo cual dificulta el acceso a desarrolladores con presupuesto limitado.

Por tal motivo, se planea la implementación de un sistema similar al de la competencia mencionada anteriormente, pero con hardware más asequible. Para lograr la meta, es necesario desarrollar un algoritmo de visión por computadora que entregue fluidez a pesar de las limitaciones técnicas.

**Objetivo**

La idea de este proyecto es desarrollar un vehículo autónomo funcional con material asequible. De esta forma se pueden sentar las bases para un proyecto a mayor escala que abarque redes neuronales, *deep learning* e inteligencia artificial.

**Marco Teórico**

En el campo de la visión por computadora existen una gran cantidad de métodos distintos para reconocer objetos, patrones o formas dentro de una imagen. Se puede decir que muchos de estos métodos tienen el mismo propósito, pero la forma en que lo realizan suele variar bastante entre uno de otro.

La segmentación de imagen es una técnica de visión por computadora que tiene diversas aplicaciones, entre la que destaca el rubro de la conducción autónoma, incluso algunos grandes fabricantes de la industria usan este algoritmo para sus productos finales. Esta técnica es de gran utilidad ya que a grandes rasgos se puede definir como el reconocimiento de objetos dentro de una o más regiones de intereses definidas por el desarrollador.

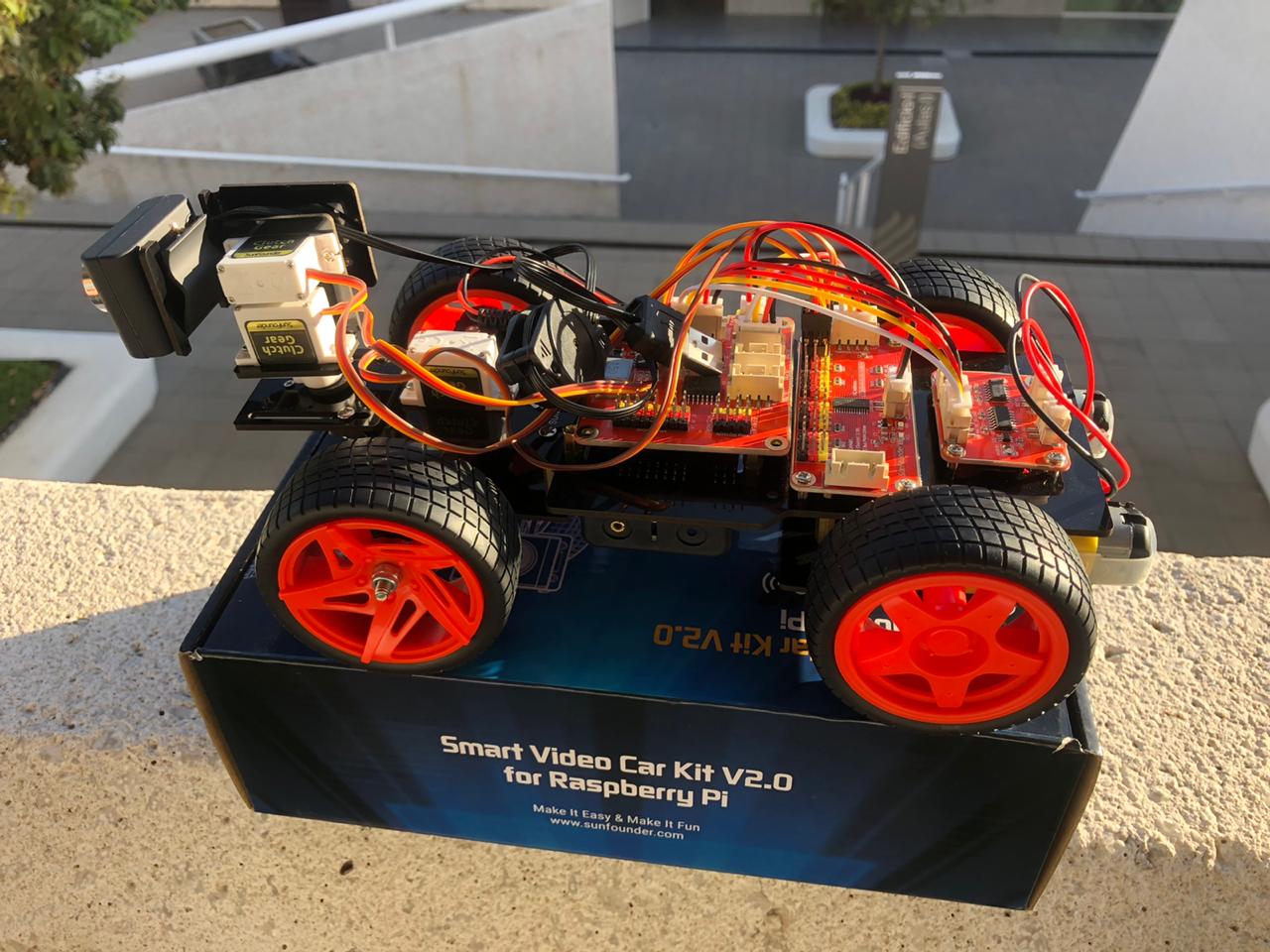


*Ejemplo ilustrativo segmentación de imagen*

**Metodología**

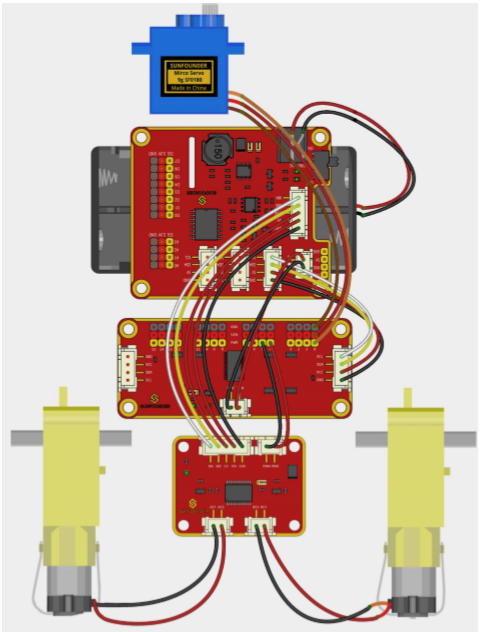
Después de buscar en internet se encontró la opción de un kit para construir un robot móvil terrestre, el cual contiene una cámara USB para ser controlado vía remota mediante una aplicación móvil. En este proyecto se usará el material de kit para adaptarlo a conducción autónoma. El kit utilizado para este proyecto se llama SunFounder y usa una *Raspberry Pi 3* con *Raspbyan* como sistema operativo.

Para el procesamiento de imágenes como primer paso en la visión por computadora fue necesario instalar *Python,* en su versión 3.7 así como la librería *OpenCV* mediante la cual será desarrollado el sistema de visión. En la siguiente figura se muestra el vehículo SunFounder ensamblado completamente.



*Smart Video Car Kit armado e implementado.*

A continuación, se incluye los diagramas de conexiones del sistema.



*Diagrama de Conexiones*

Al ser un kit con un propósito especifico es de esperarse que todo lo relacionado a hardware este practicante cubierto, es decir, todas las piezas que conforman al vehículo ya son incluidas con un instructivo para su armado, por lo tanto, no se ahondara en detalles técnicos acerca del armado ni cuestiones mecánicas, pues son ajenas al interés de este documento.

**Visión por Computadora**

Al inicio del proyecto se tenía la propuesta de hacer el reconocimiento del entorno mediante la detección de figuras de diversos colores, de esta forma, se aseguraba que el vehículo llevara acabo cierta acción dependiendo el tipo de figura detectada. Sin embargo, se demostró que la cámara incluida en el sistema no cuenta con la resolución suficiente para detectar figuras que se encuentren alejadas en la imagen por lo que a cierta distancia suele confundir las formas geométricas.

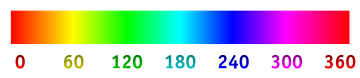
Así sin mas preámbulo, fue como se encontró el método de detección de carriles usado actualmente en muchos vehículos semi autónomos. Este método se basa en detectar las líneas que conforman un carril para direccionar el vehículo en medio de este dónde se encuentra el vehículo. Este método usa principios de robótica relacionados con el seguimiento de trayectorias, siendo mas preciso, utilizando los conceptos del seguidor de línea. De otra forma, se puede describir la detección de carriles como convertir un video del camino en las coordenadas que guían a nuestro vehículo mediante carriles identificables.

El primer paso en este sistema de visión es definir el color de los carriles y aislarlos de la imagen obtenida por la cámara.

En las siguientes lineas de código, se realiza todo el procesamiento de imagen para obtener las lineas de los carriles. En este proyecto se trabajara con el espacio de color HSV; dado que OpenCV lee las imágenes por defaul en RGB es necesario hacer la conversión entre espacio de color.

hsv = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)

Lo siguientes es definir los rangos de los valores para la mascara que detectara el color azul en nuestra imagen



*Espacio de color HSV*

Como se puede observar, el rango de colores azul abarca desde 120 – 300 en la escala de 0 – 360, sin embargo OpenCV usa un rango de 0 -180 por lo que los valores de la mascara quedarían 60 – 150, por lo tanto se crean dos arreglos con valores minimo y maximo respectivamente. En estos arreglos tambien se incluye el valor de la saturación y brillo.

lower\_blue = np.array([60, 40, 0])

upper\_blue = np.array([150, 255, 255])

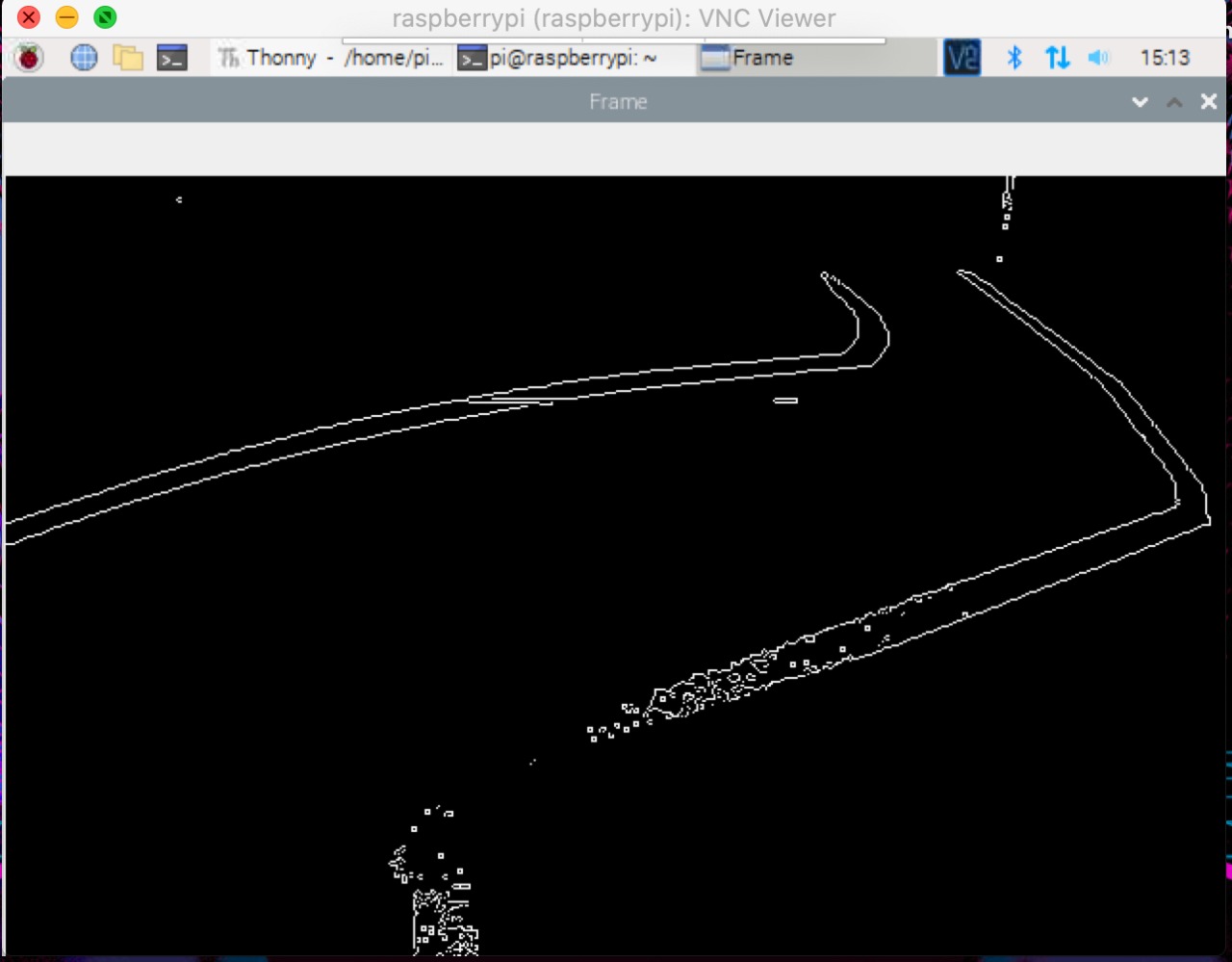
La siguiente linea de código junta ambos limites en una mascara para detectar el color azul en la imagen.

mask = cv2.inRange(hsv, lower\_blue, upper\_blue)

El resultado de este proceso nos da como resutlado una imagen donde solo son visibles las lineas del carril. Posteriormente, se aplica la función *Canny*, la cual forma parte de la librería de OpenCV y entrega por resultado los bordes detectados en la imagen. El primer parametro que recibe esta función es la mascara que definimos previamente, mientras lo otros dos son los rangos minimos y maximos para la deteción de bordes, donde la documentación de OpenCv recomienta que sea 200 y 400.

edges = cv2.Canny(mask, 200, 400)

El resultado de todos los pasos anteriores queda demostrado con la siguiente figura:



*Imagen Resultado de la función Canny*

En la imagen se puede observar como se detallan los bordes de las lineas que conforman el carril a lo largo de toda la pantalla, sin embargo, con el fin de mejorar el procesamiento y performance del sistema de vision, es necesario definir una región de interes y así descartar lo que este fuera de ella. En este caso se dividio verticalmente la imagen en dos partes, tomando la porción de abajo como la región de interes. Para esta tarea se implento la siguiente función:

def region\_of\_interest(canny):

height, width = canny.shape

mask = np.zeros\_like(canny)

polygon = np.array([[

(0, height \* 1 / 2),

(width, height \* 1 / 2),

(width, height),

(0, height),

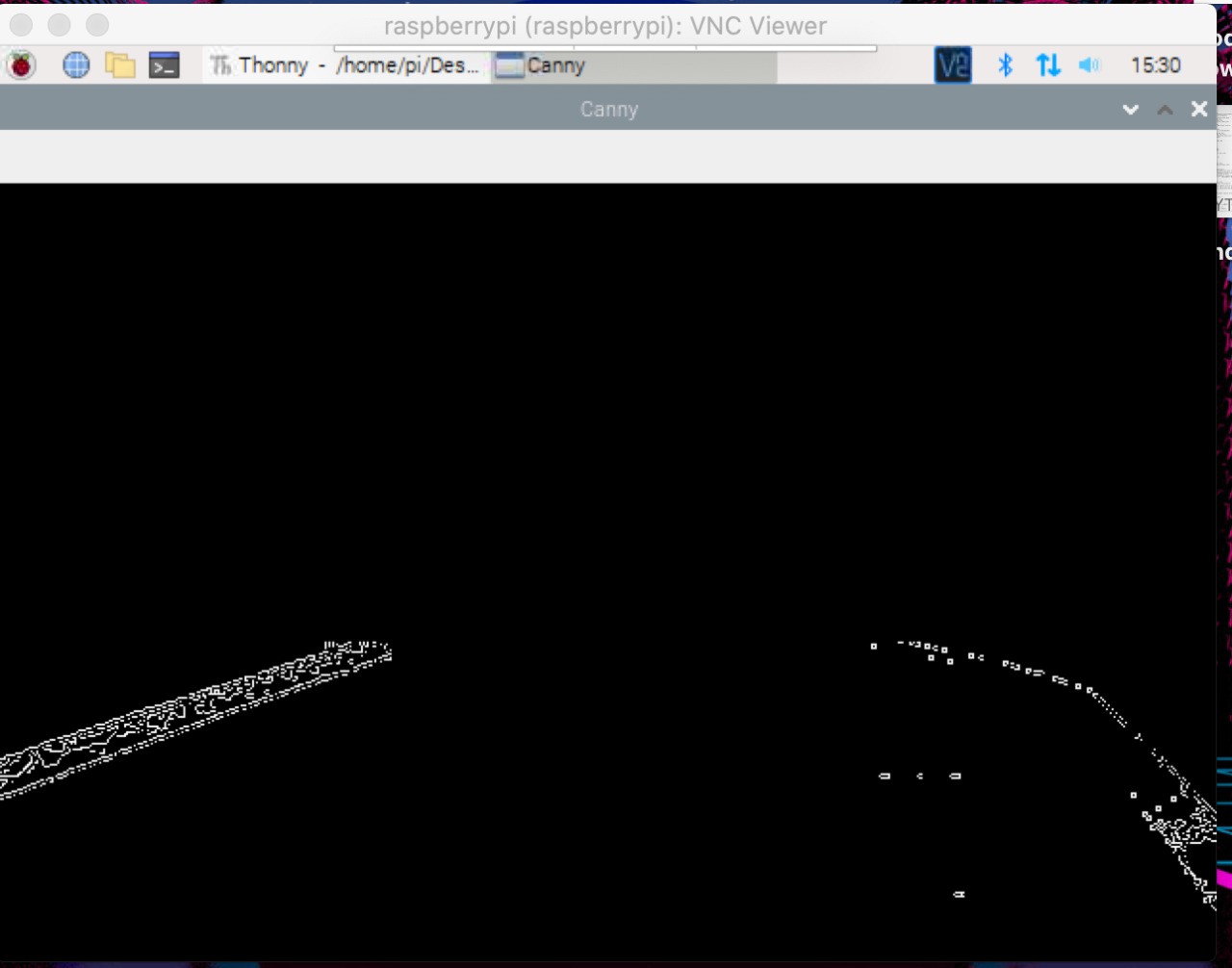
]], np.int32)

cv2.fillPoly(mask, polygon, 255)

masked\_image = cv2.bitwise\_and(canny, mask)

return masked\_image

Dentro de esta función se crea una mascara para la parte baja de la pantalla que despues se mezclara con el resultado de la función canny. El resultado de esta función se guarda en la variable *masked\_image* donde el resultado luce de la siguiente manera:



*Resultado de Region de Interes*

En esta imagen, se puede observar como la detección de bordes ha sido limitada a solo la parte inferior de la imagen gracias a la region de interes. Como humano puede parecer bastante obvio y facil de notar que se forman dos lineas en la pantalla, sin embargo, como sistema no es tan sencillo ya que solo se tienen un monton de pixeles muy juntos uno del otro. Afortunadamente OpenCV incluye una funcion llama “La transformada de Hough” la cual es una tecnica utilzada para detectar circulos, lineas y elipses en una imagen. Esta función escencialmente intenta acomodar lineas en todos los pixeles blancos y retornar la que mas se ajuste a la figura predefinida.

Esta parte del procesamiento de imagen en este sistema de visión puede ser llamado como la detección de segmentos de lineas o carriles. Las siguientes lineas de código son la implementación de este paso en el sistema.

rho = 1

angle = np.pi / 180

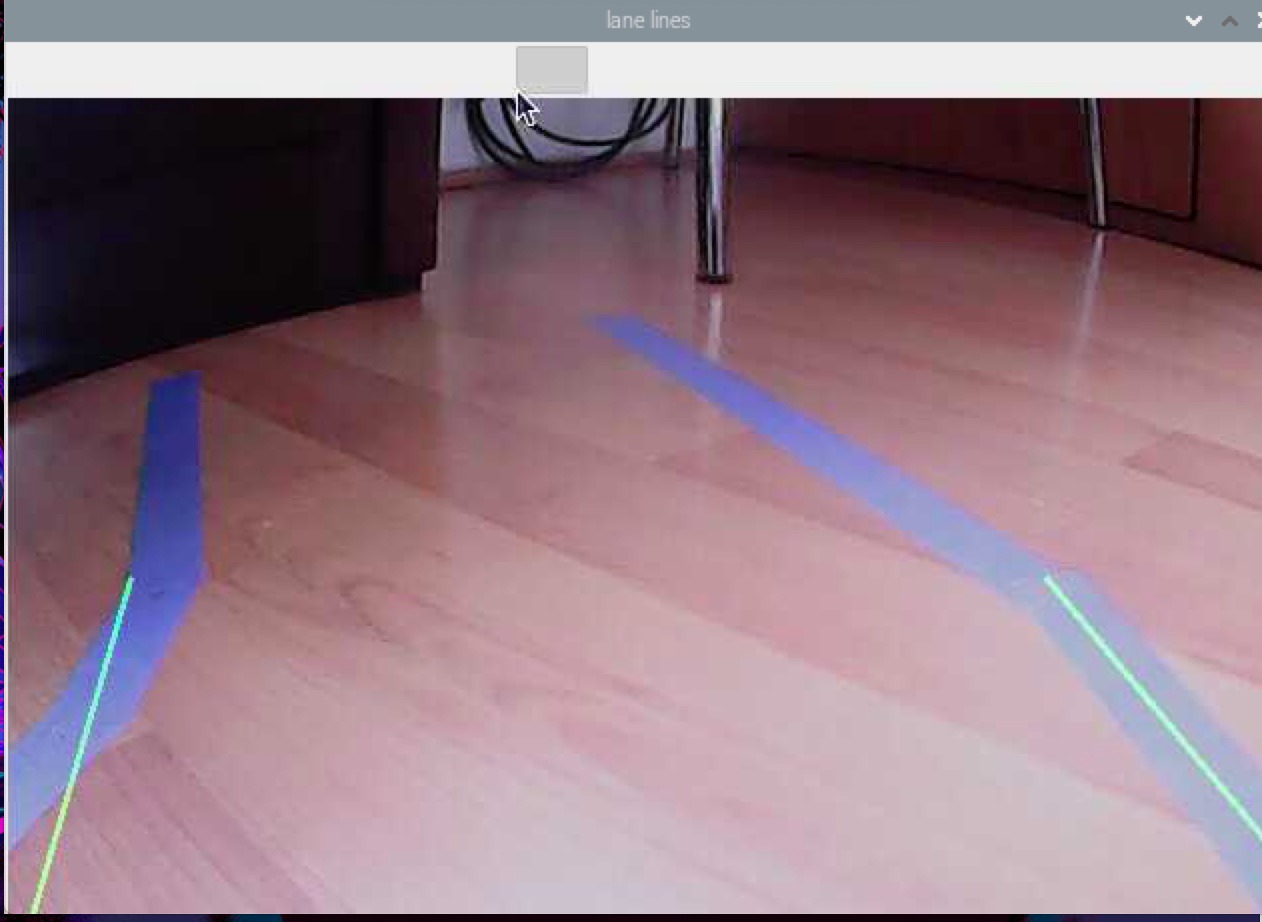
min\_threshold = 10

line\_segments = cv2.HoughLinesP(masked\_image, rho, angle, min\_threshold, np.array([]), minLineLength=8, maxLineGap=4)

Internamente la función HoughLinesP trabaja los ángulos en coordenados polares, entre los parámetros que recibe esta función son:

* *Rho* es la distancia de presión en pixeles, para este caso solo se usará la unidad mínima.
* *Angle* es la precisión angular en radianes.
* *min\_threshold* es el numero mínimo de puntos para considera un segmento de línea.
* *minLineLenght* es la distancia mínima de un segmento en pixeles, esto quiere decir que la transformada no regresara ninguna línea mas corta de este valor.
* *maxLineGap* es el numero de pixeles que dos líneas pueden estar separadas y seguir considerándose la misma, esto debido a que por propiedades de iluminación y el sensor de la cámara, en ciertos casos no detecta siempre todos los tonos del color azul.

El resultado es el siguiente:



*Imagen resultante de la transformación de Hough*

Una vez que se tiene ambos pequeños segmentos de líneas, se puede diferenciar una de la otra por medio de su ubicación, derecha e izquierda, gracias a su pendiente, es decir, las líneas izquierdas siempre tendrán una pendiente hacia arriba, mientras que la derecha tendrá lo contrario, tal como se aprecia en la imagen de arriba. Para obtener la pendiente de cada línea primero se debe calcular los puntos finales de ambos.

* **Direccionamiento del vehículo**

Una vez que existen las coordenadas de ambos carriles como referencia, se debe establecer una línea guía que se usará para calcular el ángulo de giro del vehículo. El objetivo es tratar de mantener el carro lo más centralizado posible, sin que se salga del camino. En el caso ideal cuando ambas líneas están presentes, se puede calcular el ángulo de dirección simplemente gracias al punto medio que existe entre los dos puntos finales de cada segmento de línea. Sin embargo, existe un caso especial debido a que la cámara de vehículo se encuentra fija en una misma posición, esto provoca que ciertos momentos en el accionar del sistema solo detecte una línea, siendo mas específicos, este caso se presenta al momento de dar una vuelta.

Cuando solo una línea es detectada se puede inferir que el sentido de giro debe ser contrario a la ubicación del segmento, es decir, si la línea izquierda es la única presente, el carro debe girar hacia la derecha. Esto se puede solucionar de una manera sencilla haciendo que la línea de dirección tenga la misma pendiente que la línea detectada. Ambos casos se demostrarán mas adelante con ejemplos.

* **Steering Angle**

Por otra parte, se tiene que calcular el ángulo de dirección para el vehículo; como se menciono al principio de este documento, para este proceso se usara la planificación de trayectoria en base a el seguimiento de línea, donde se obtiene la siguiente formula:



Donde theta representa el ángulo de dirección, mientras a y b son los puntos finales calculados de cada segmento. En el algoritmo del sistema de visión de computadora, fue implementado de la siguiente manera:

angle\_to\_mid\_radian = math.atan(x\_offset / y\_offset)

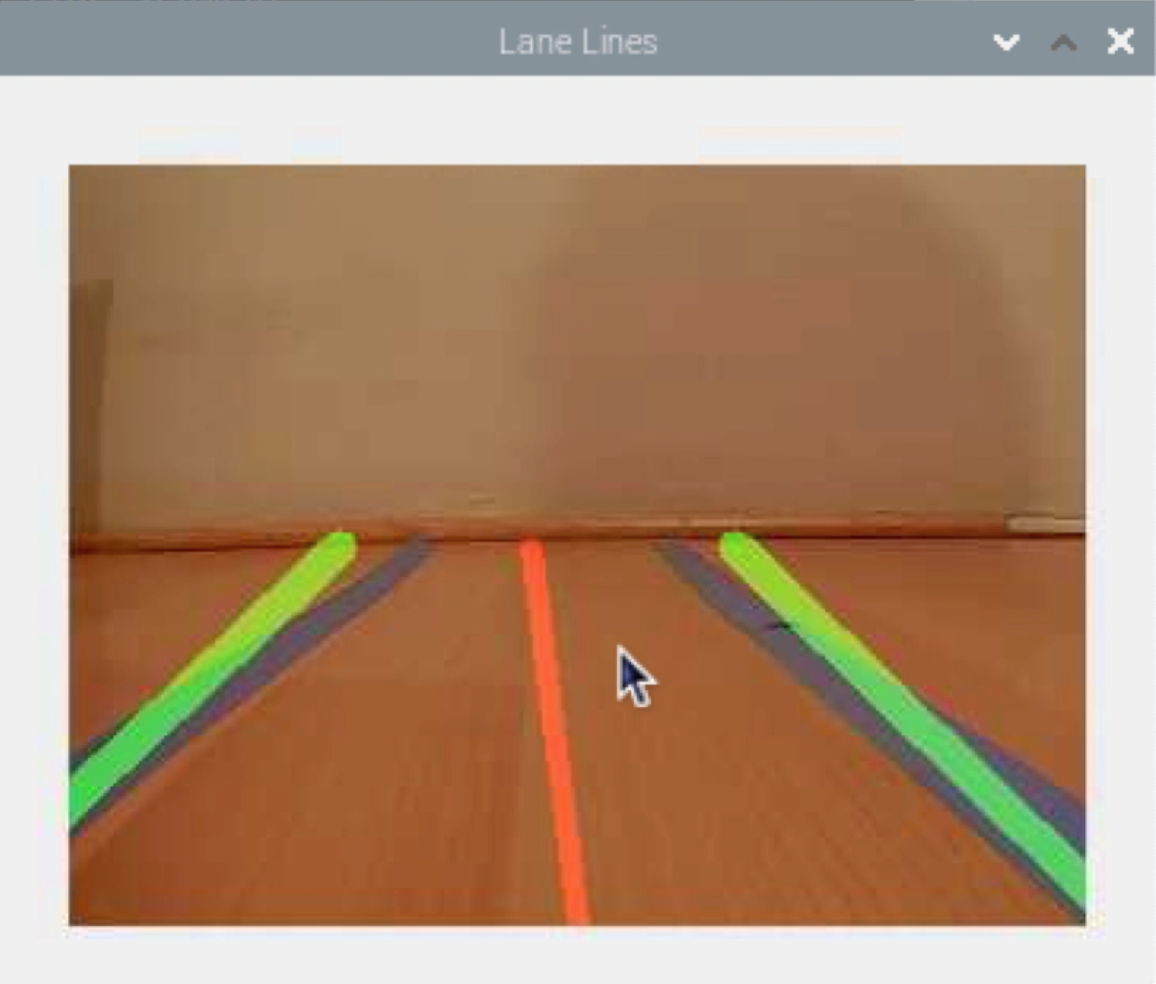
angle\_to\_mid\_deg = int(angle\_to\_mid\_radian \* 180.0 / math.pi)

steering\_angle = angle\_to\_mid\_deg + 70

En la primera linea, se aplica la formula del seguidor de linea usando la librería math para encontrar la distancia arco tangencial. Este angulo esta en radianes, por lo que el siguiene paso es transformalo a grados, ya que la librería PiCar que incluye las funciones para controlar los motores del vehículo esperan recibir un valor en esta escala para controlar la posición del servo motor de la dirección. Por ultimo, se debe sumar la constante de setenta, pues es la posición del servo motor donde el vehículo avanza de forma recta y es la referencia de nuestro direccionamiento.

Con el valor del angulo de dirección es posible trazar la linea guía con fines ilustrativos, esta linea siempre tendra su origen fijo justo a la mitad de la pantalla, mientras que el punto final es el angulo de direccionamiento.

El resultado del sistema de visión despues de implementar estos conceptos se muestra en la siguiente figura:



*Sistema de visión implementado*

Como se observa en la imagen anterior, se dibujan dos lineas verdes encima de las lineas fisicas del carril (en color azul) así como se dibuja la linea de dirección en color rojo.

**Resultados y análisis**

Con el sistema de visión implementado, se procedio a crear una ruta en la cual hacer la validadicón del sistema así como observar su comportamiento. Debido a factores como el espacio, la señal inalambrica de internet entre otros, se diseño un ruta apropiada, quedando de la siguiente manera:

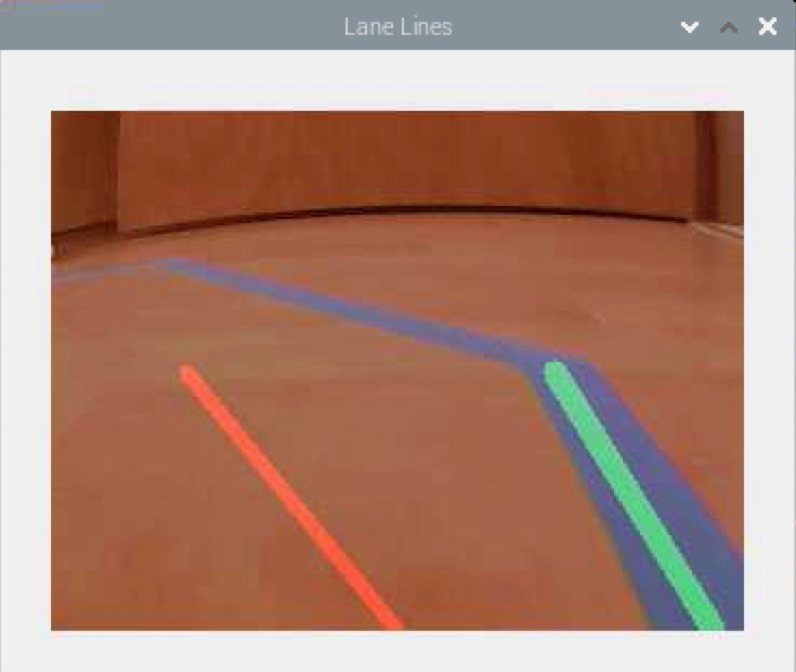


*Ruta experimental para el vehículo autonomo*

Esta ruta esta compuesta tanto de tramos rectos como de vueltas a la izquierda o derecha con el fin de poner a prueba el sistema en todos los escenarios posibles.

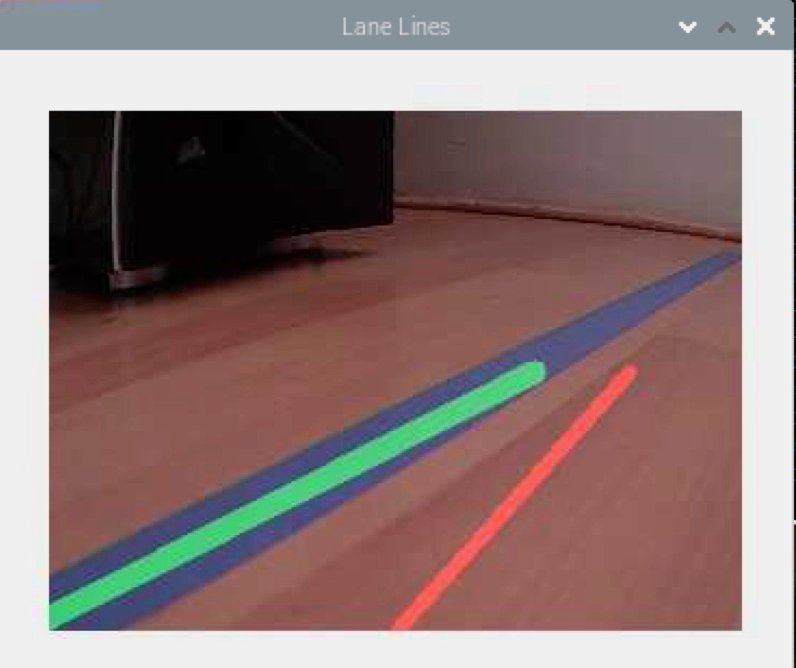
Al realizar las pruebas se logro comprobar el funcionamiento del algoritmo de visión, para esta tarea se utilizo el servicio de VNC con el fin de obtener retroalimentación en tiempo real de lo que la camara esta captando. En las siguientes figuras se demuestra el sistema bajo ciertos escenarios.

En la siguiente imagen se muestra el caso cuando el vehículo encuentra una vuelta hacia la izquierda, en ella se demusetra lo mencionado anteriormente: solo se detecta una linea, por lo que la linea de dirección tendra una pendiente similar al segmento de linea detectado.



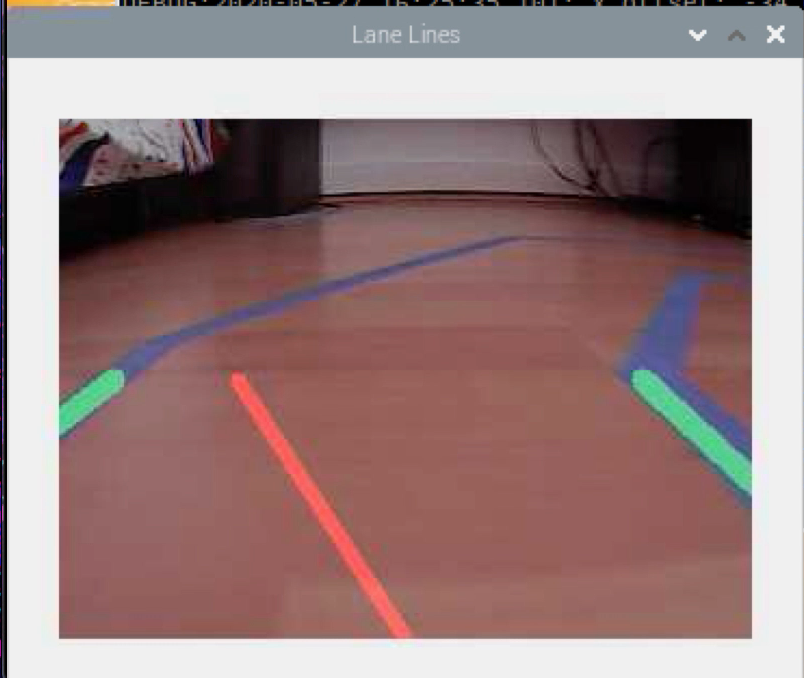
*Vuelta a la izquierda.*

El siguiente escenario muestra el mismo principio que el descrito anteriomente, con la diferencia que el giro se hace hacia el sentido contrario.



*Vuelta para la derecha*

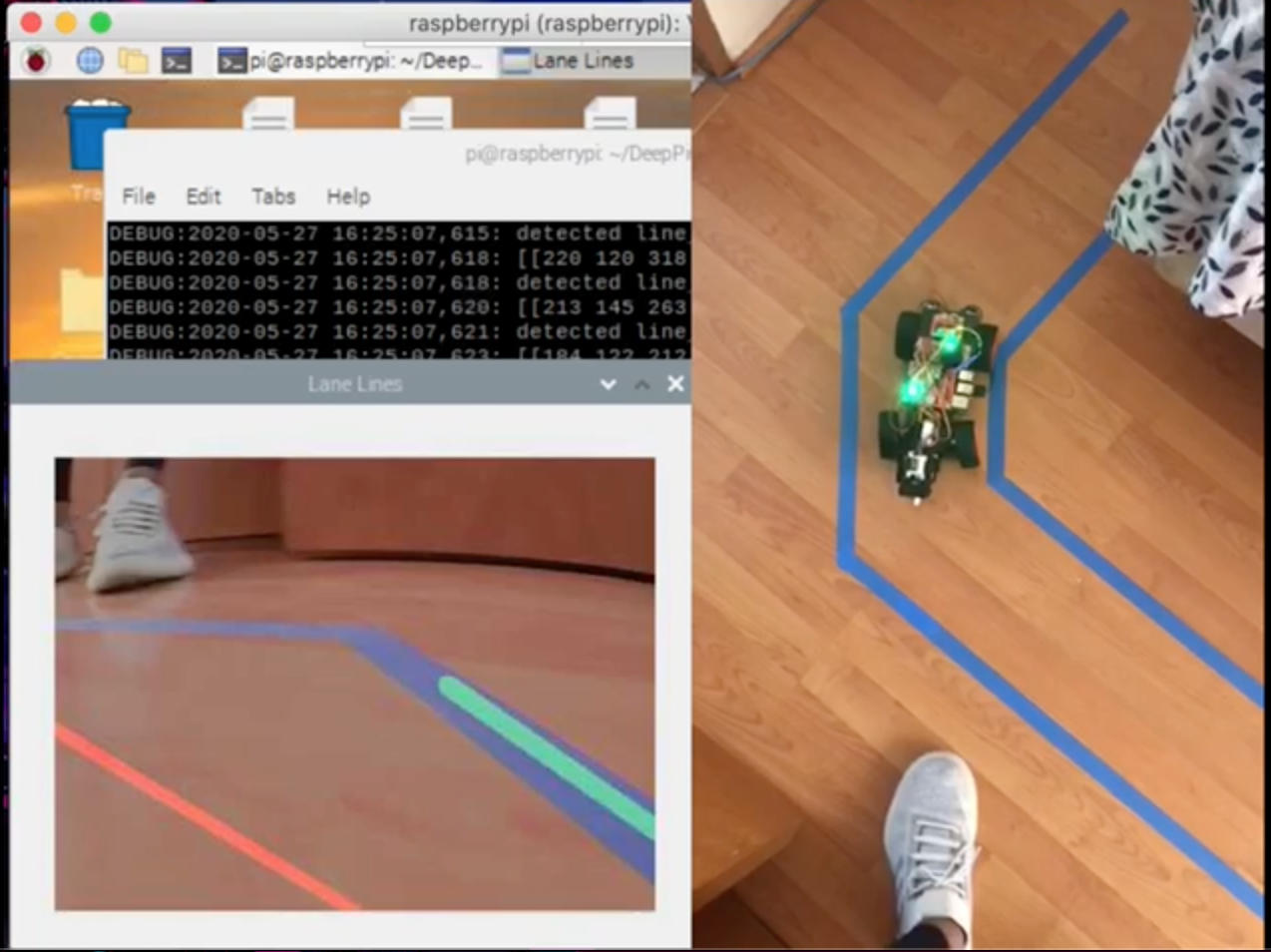
Finalmente, se demuestra un ultimo escenario, cuando ambas lineas son captadas y avanza hacia adelante.



*Caso de avance recto*

En la pasada imagen el angulo de dirección tiene una ligera desviación hacia la izquierda debido a que la linea del mismo lado fue invisible durante un breve lapso, pero en este momento ya es detectada por lo que el angulo esta regresando lentamente a su posición central.

Por ultimo se anexa el video de demostración del vehículo autonomo, en el cual se puede apreciar como por momentos el vehículo pareciera perder la trajectoria pero logra volver al camino.



*Vista previa video final de demostración*

<https://drive.google.com/file/d/1yeZYG_JIDaJCK-nG8EJzxkQo66VcjAF6/view?usp=sharing>

**Conclusiones**

Gracias a este proyecto se demuestra como la visón por computadora es una herramienta muy versátil y potente que da solución a tareas complejas que con otros recursos seria muy difícil llevarse acabo, además que posiblemente significarían un costo elevado económicamente. El vehículo autónomo implementado durante el proyecto es capaz de cumplir con su tarea, pero a bajas velocidades debido a la capacidad de procesamiento nativa de la tarjeta Raspberry Pi 3, la cual se considera una restricción intrínseca del hardware.

Por otro lado, el campo de visión por computadora será el pilar sobre el cual se desarrollen tecnología con aplicaciones al futuro. Por ejemplo, durante la actual crisis sanitaria originada por el COVID-19 un sistema de visión que arroje muestras de la temperatura de personas en cierto lugar sería una gran herramienta para combatir el contagio masivo.

**Referencias**

Bradski, G., Kaehler, A. (2008). Learning OpenCV: Computer vision with the OpenCV library. " O'Reilly Media, Inc.".

Howse, J. (2013). OpenCV computer vision with python. Packt Publishing Ltd.

Pulli, K., Baksheev, A., Kornyakov, K., Eruhimov, V. (2012). Real-time computer vision with OpenCV. Communications of the ACM, 55(6), 61-69.

Sharma, P. (2019). Computer Vision Tutorial: A Step-by-Step Introduction to Image Segmentation Techniques (Part 1). junio 2020, de Analytics Vidhya Sitio web: https://www.analyticsvidhya.com/blog/2019/04/introduction-image-segmentation-techniques-python/

Bechtel, M. G., McEllhiney, E., Kim, M., Yun, H. (2018). DeepPicar: A Low-cost Deep Neural Network-based Autonomous Car. junio 2020, de University of Kansas; Indiana University Sitio web: https://arxiv.org/pdf/1712.08644.pdf