Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Puebla

Implementación de Robótica Inteligente TE3002B GRUPO(502)

Alumnos

Diego García Rueda
A01735217

Jhonatan Yael Martínez Vargas
A01734193

Jonathan Josafat Vázquez Suárez
A01734225

Profesores

Dr. Alfredo García Suárez

Dr. Rigoberto Cerino Jiménez

Dr. Juan Manuel Ahuactzin Larios

Dr. Cesar Torres Rios

14 Mayo 2023, Puebla, México.

Resumen

En este reporte se mostrarán los resultados del equipo para este reto semanal, se muestran las mejoras aplicadas al resultado del reto anterior siendo la principal la implementación de un nodo que permite al Puzzlebot utilizar la camara para asi poder detectar círculos de colores, esto en representación de un semáforo. Se detalla la implementación de este nodo al igual que las conexiones mediante tópicos al nodo "Controller".

A diferencia del reto anterior, los nodos generados de ROS ya se encontraran en la *Jetson Nano* a diferencia de los anteriores avances en los que se utilizaban mediante SSH, esto debido a que la velocidad de procesamiento de las imágenes debe ser mucho más rápida para así poder hacer un procesamiento en vivo.

Este reporte tiene como objetivo mostrar las mejoras respecto al entregable anterior principalmente en la implementación del nodo de procesamiento de imágenes.

Introducción

En aplicaciones como la robótica móvil, la implementación de un sistema de visión es una parte fundamental del funcionamiento de un vehículo autónomo, independientemente de la existencia de otros métodos de reconocimiento del entorno como lo sería los LiDAR o sensores de proximidad. La visión por cámara o por cámara de profundidad ofrece múltiples ventajas para la autonomía del robot. Algunas de sus aplicaciones pueden ser el seguimiento de trayectorias, reconocimiento de objetos, detector de obstáculos,clasificación de objetos etc.

Para el desarrollo de esta visión existen múltiples librerías como *OpenCV* o *Scikit Learn* que nos permiten realizar operaciones de procesamiento de una manera bastante sencilla, sin embargo es necesario tener en cuenta que una gran cantidad de instrucciones generaron un procesamiento mucho más complejo y por lo tanto más tardado, por lo tanto, es indispensable encontrar un punto de equilibrio en el que el procesado sea robusto y que al mismo tiempo no ralentice la captura de video en vivo.

Este reto consta del desarrollo de un procedimiento en el que a partir de la obtención de video mediante la cámara del *Puzzlebot* se detectara el color del semaforo y en base a este color se definirá el comportamiento del robot, manteniendo la velocidad original proporcionada por el sistema de control, reduciendo la velocidad a la mitad o deteniéndose por completo. Finalmente se explicarán las pequeñas modificaciones realizadas al nodo *controller* para la modificación de la velocidad.

Objetivos

Desarrollar capacidades en el manejo de ROS implementando un sistema de visión por cámara que permita detectar círculos de colores a modo de representación de un semáforo y un cambio de comportamiento del robot a partir de la identificación del color.

Implementación de un procedimiento robusto de detección de colores modificando aspectos del video como la escala de colores y propiedades de la imagen para un mejor procesamiento y reconocimiento de imágenes.

Solución del problema

Para la solución de este reto se decidió utilizar la estructura proporcionada por la organización socio formadora, la estructura es la siguiente:

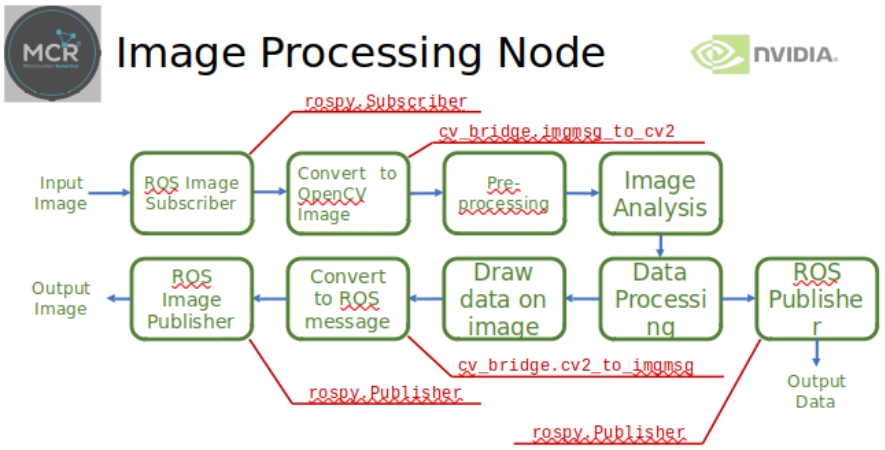

Figura 1. Estructura utilizada para el reto.

El nodo desarrollado se llama *Image Detection* , el desarrollo de este nodo y su conexión con los nodos ya existentes será explicado a continuación:

- **Image Detection**

Este nodo recibe como entrada la grabación de la cámara, La salida de este nodo son mensajes que indican el color detectado por la cámara y la imagen procesada, esta imagen se puede observar de manera que se puedan observar los resultados del procesamiento.

```
#Subscribers y Publishers
self.image_sub = rospy.Subscriber('/video_source/raw',Image, self.image_callback)

self.flag_pub = rospy.Publisher('/color_Flags', String, queue_size=10)
self.image_pub = rospy.Publisher('/proc_image', Image, queue_size=10)
```

Figura 2. Código de tópicos de Suscripción y Publicación.

La parte de procesamiento del nodo lo primero que hace es convertir la imagen a escala HSV, esta escala ofrece diversas ventajas al momento de la detección de colores principalmente en el ajuste de saturación y brillo de los colores, despues de la conversión de escalas de colroes se establecen valores maximos y minimos para los colores verde, amarillo y rojo en escala HSV, permitiendo asi elaborar mascaras, estas mascaras serán utilizadas posteriormente en la misma función:

```
#Rangos de valores HSV
self.lower_red = np.array([170, 100, 100])
self.upper_red = np.array([180, 255, 255])
self.lower_yellow = np.array([20, 100, 100])
self.upper_yellow = np.array([40, 255, 255])
self.lower_green = np.array([40, 100, 100])
self.upper_green = np.array([70, 255, 255])
```

Figura 3 Valores en HSV utilizados como mascara.

Debido a los 3 colores a detectar, se realizan 3 llamadas a la función de detección de contornos, utilizando los valores de las máscaras de cada color, en caso de qUe se detecte una figura con las características necesarias (estas características se explicarán más adelante) se publicará el mensaje con el color identificado, este mensaje será utilizado por el nodo *controller* y finalmente se publicara la imagen para que se pueda apreciar la detección de los círculos de colores.

```
def processing(self):

    frameHSV = cv2.cvtColor(self.image, cv2.COLOR_BGR2HSV)

    for i in self.circles(self.image, self.lower_red, self.upper_red):
        cv2.drawContours(self.image, [i], 0, (0,0,255),3)
        self.image_pub.publish(self.image)
        self.flag_pub.publish("R")

    for i in self.circles(self.image, self.lower_yellow, self.upper_yellow):
        cv2.drawContours(self.image, [i], 0, (0,255,255),3)
        self.image_pub.publish(self.image)
        self.flag_pub.publish("Y")


    for i in self.circles(self.image, self.lower_green, self.upper_green):
        cv2.drawContours(self.image, [i], 0, (0,255,0),3)
        self.image_pub.publish(self.image)
        self.flag_pub.publish("G")
```

Figura 3. Código función de procesamiento y publicación

La función *circles* usa como parámetros el valor máximo y mínimo para la máscara, y la imagen, lo primero que realiza es la conversión a formato HSV de BGR, se define una máscara y se detectan los contornos utilizando la función *cv2.findContours(),* esta función detectará contornos de los objetos que se encuentren dentro de los límites de la máscara definida anteriormente actuando así la máscara como primer filtro de la imagen.

En caso de que se encuentren más de 1 contorno, se utilizará únicamente el contorno de mayor tamaño, a este contorno se le aplicará una “limpieza” con la función *convex Hull().* Con este nuevo contorno la detección de la figura es mejor. Con este nuevo entorno ya podemos calcular el área y el perímetro , parámetros que son necesarios para la obtención del valor de la circularidad.

Finalmente, se establecen otros 2 filtros, el primero de ellos es que el área de la figura de la cual obtuvimos el contorno sea mayor a 300, se determinó mediante prueba y error que este

era un valor bastante bueno al momento de la identificación de los círculos de colores, también se especificó que la circularidad del contorno debía ser mayor a 1.5, este valor también fue definido mediante prueba y error, generalmente un circulo tiene una circularidad de 1 sin embargo habían contornos que a pesar de no tener figura completamente circular aún eran detectados siendo este valor bastante adecuado como filtro. El contorno resultante será enviado a la función *processing()* mencionada anteriormente para hacer la publicación de las banderas correspondientes.

```
def circles(self,image, lower_color, upper_color):
    circle = []

    # Convertir el frame a HSV
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
  # Aplicar una mascara para el color especifico
    mask = cv2.inRange(hsv, lower_color, upper_color)
  # Encontrar contornos en la mascara
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

  # Iterar sobre los contornos encontrados
    for contour in contours:
        # Calcular el area y el perimetro del contorno
        new_contour = cv2.convexHull(contour)
        area = cv2.contourArea(new_contour)
        perimeter = cv2.arcLength(new_contour, True)
        circularity = 4 * np.pi * area / max(perimeter, 0.1)

        if area > 300 and circularity > 1.5 :
                circle.append(new_contour)
    return circle
```

Figura 4. Código función detección de contornos.

- **Controller**

Este nodo se mantiene de la misma manera que en los entregables anteriores solo que se le agrega la inclusión del nuevo nodo agregando el tópico *color_Flags,* este tópico permitirá definir las acciones a seguir del robot respecto a su desplazamiento.

```
# Subscribers
rospy.Subscriber("/wr", Float32, self.wr_callback)
rospy.Subscriber("/wl", Float32, self.wl_callback)
rospy.Subscriber("/set_point", SetPoint, self.set_point_callback)
rospy.Subscriber("/color_Flags", String, self.flag_callback)
# Publishers
self.velocity_pub = rospy.Publisher("/cmd_vel", Twist, queue_size = 10)
self.status_pub = rospy.Publisher("/status", String, queue_size = 10)
```

Figura 5. Código. Publishers y Subscribers del nodo *Controller*

El tópico contiene strings que son “G”, “Y” y “R” representando cada uno de los colores, estos valores serán utilizados en una nueva función implementada llamada *color_action* que definirá si es que el robot avanza, avanza con la velocidad reducida o detendrá el robot

```
def color_action(self):
    if self.flag == "G":
        self.x = 1
        self.run()

    elif self.flag == "Y":
        self.x = 2
        self.run()

    elif self.flag == "R":
        self.stop()
```

Figura 6. Código. Nueva función implementada

El parámetro “x” es una nueva adición al nodo, este parámetro se utiliza como divisor al momento de publicar la velocidad, en caso de que se detecte el círculo verde el valor de x será de 1, esto mopstrando que la velocidad otorgada por el sistema de control será la misma, en caso de que se detecte el circulo amarillo entonces el valor de x será de 2, esto quiere decir que al momento de hacer la publicación de la velocidad, esta será reducida a la mitad y finalmente se detecta el circulo rojo entonces el *Puzzlebot* se detendrá completamente.

Resultados

A continuación se presentan los resultados, el funcionamiento del robot de manera física usando hojas de papel para la simulación del semáforo:

Conclusión del equipo

Como equipo consideramos que este challenge aumentó considerablemente el desarrollo de este proyecto de convertir al *Puzzlebot* en un vehículo autónomo gracias a la visión y el reconocimiento de imágenes, la mayor complicación del proyecto fue la configuración de la cámara ya que se presentaron diversos contratiempos al intentar conectar las cámaras, siendo un total de 3 intentos fallidos con distintas cámaras sin poder tener resultados satisfactorios, la parte del procesamiento de las imágenes fue bastante sencillo debido a los conocimientos que ya teníamos sobre procesamiento de imágenes además de que la modificación de la estructura preexistente del *Puzzlebot* permitió una incorporación sencilla en esta etapa siendo su inclusión del nuevo nodo bastante sencilla aunque a pesar de todo lo anterior siguieron existiendo complicaciones al momento de hacer la estructura en ROS.