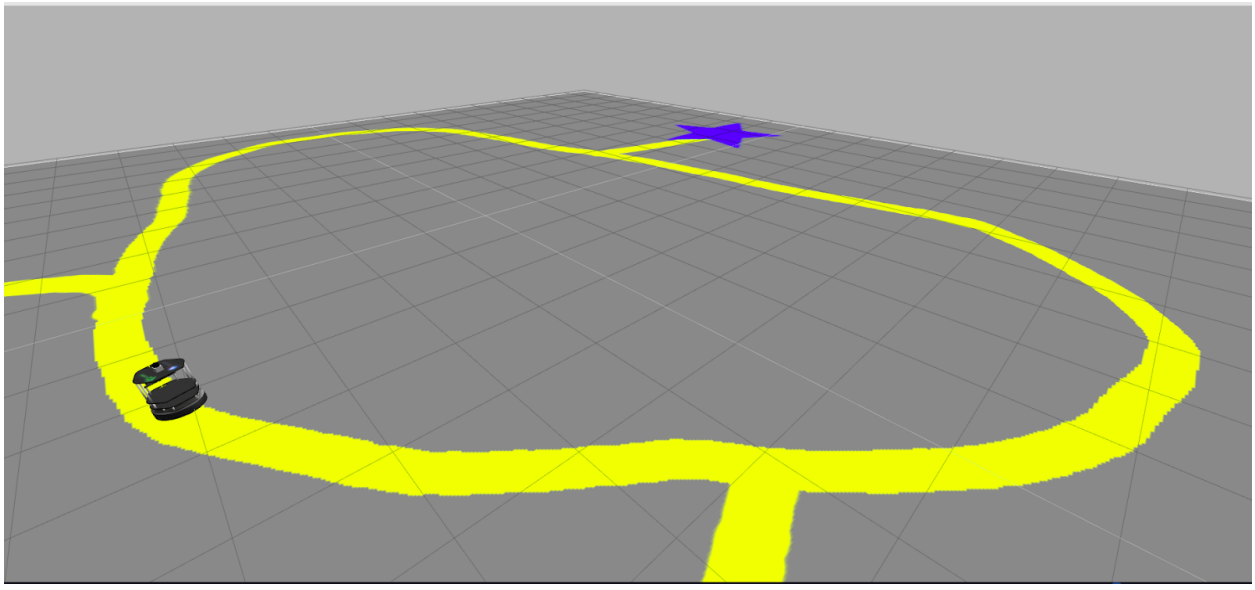
Computer Vision & Image Processing

OpenCV-ROS : A Color Based Path Following Platform for Ground Robots Based On Robot Operating System

Automatic Guided Vehicles



Abstract

This project is about how to make color based path following Robot, these kinds of Robots are used in warehouses in industries to transfer items from one place to another. In this project I will be using Robot Operating System and OpenCV bridge package of ROS to integrate ROS with OpenCV so that we can utilize all the features of OpenCV.

Introduction

Line Following is one of the most important aspects of robotics. A Line Following Robot is an semi-autonomous robot which is able to follow either a black or white line that is drawn on the surface consisting of a contrasting color. It is designed to move automatically and follow the made plot line. The robot uses several sensors to identify the line thus assisting the robot to stay on the track. The array of four sensor makes its movement precise and flexible. The robot is driven by DC gear motors to control the movement of the wheels.The Robot Operating System base interface is used to control the speed of the motors, steering the robot to travel along the line smoothly. This project aims to show how to make line following Robot and tune it's parameters in simulated environment. OpenCV library, OpenCV_bridge ROS package and turtlebot is used to demonstrate this in Gazebo simulation environment.

Implementation

How to use OpenCV in ROS :

As you might have guessed, OpenCV is not a ROS library, but it's been integrated nicely into ROS with OpenCV_bridge. This package allows the ROS imaging topics to use the OpenCV image variable format. To use OpenCV, a RGB camera for the TurtleBot robot is used.

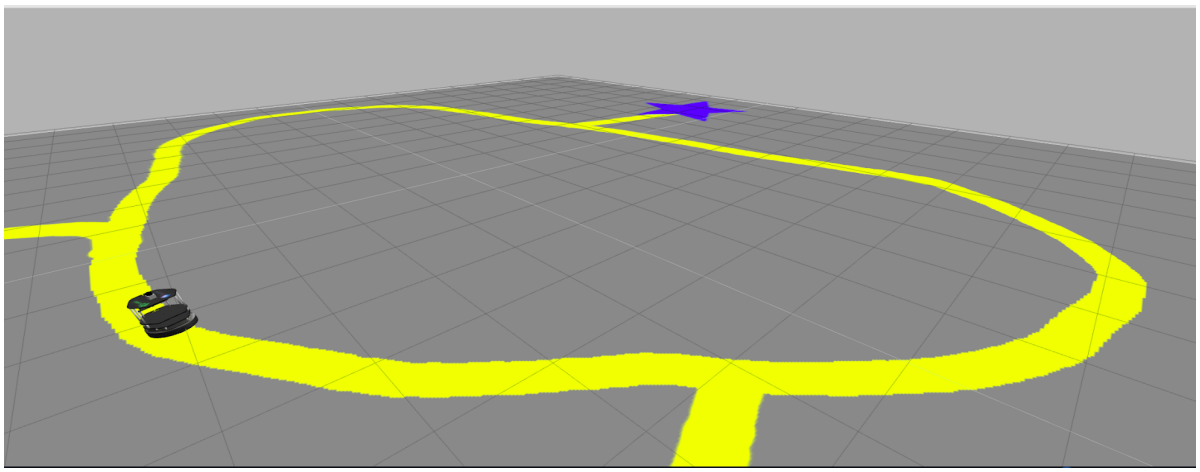


Figure: Robot environment used in this project

To move the robot in this environment, use the following steps:

1. Take images from the ROS topic and convert them into OpenCV format
2. Process the images using OpenCV libraries to obtain the data we want for the task
3. Move the robot along the yellow line, based on the data obtained

1. Take images from the ROS topic and convert them into OpenCV format :

```
$cd ~/catkin_ws/src

$catkin_create_pkg my_following_line_package rospy cv_bridge image_transport sensor_msgs

$cd ~/catkin_ws/

$catkin_make / catkin build

$source devel/setup.bash

$rospack profile

$roscd my_following_line_package

$mkdir scripts;cd scripts

$touch line_follower_basics.py

$chmod +x *.py
```

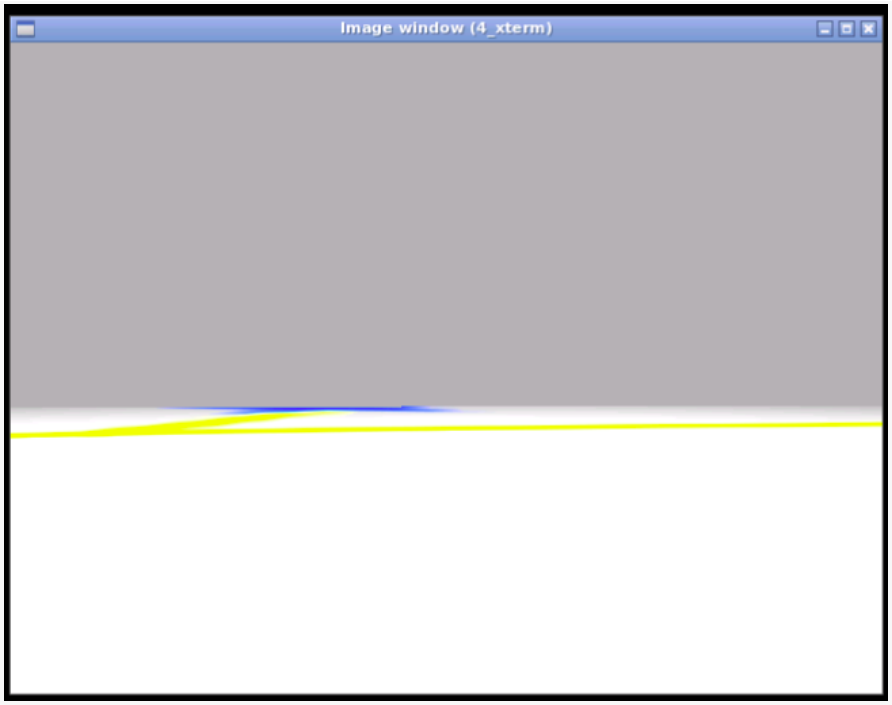
Take the images from a ROS topic, convert them into the OpenCV format, and visualize them in the Graphical Interface window.

line_follower_basics.py

```
#!/usr/bin/python3

import roslib

import sys

import rospy

import cv2

import numpy as np

from cv_bridge import CvBridge, CvBridgeError

from geometry_msgs.msg import Twist

from sensor_msgs.msg import Image

class LineFollower(object):

    def __init__(self):

        self.bridge_object = CvBridge()

        self.image_sub = rospy.Subscriber("/camera/rgb/image_raw",Image,self.camera_callback)

    def camera_callback(self,data):

        try:

            cv_image = self.bridge_object.imgmsg_to_cv2(data, desired_encoding="bgr8")

        except CvBridgeError as e:

            print(e)

        cv2.imshow("Image window", cv_image)

        cv2.waitKey(1)
```

```
def main():
    line_follower_object = LineFollower()
    rospy.init_node('line_following_node', anonymous=True)
    try:
        rospy.spin()
    except KeyboardInterrupt:
        print("Shutting down")
    cv2.destroyAllWindows()
if __name__ == '__main__':
    main()
```

Result :



2. Process the images using OpenCV libraries to obtain the data we want for the task:

Apply filters to the image:

The raw image is useless unless you filter it to only see the color you want to track. You can crop sections of the image that you do not require so that your program runs faster. We also need to extract some data from the images to move the robot to follow the line.

2.1 First Step: Get Image Info and Crop the Image

```
height, width, channels = cv_image.shape

descentre = 160

rows_to_watch = 20

crop_img = cv_image[(height)/2+descentre:(height)/2+(descentre+rows_to_watch)][1:width]
```

Why these values and not others? It depends on the task. In this case, you're interested in lines that aren't too far away from the robot, nor too near. If you concentrate on lines too far away, the robot won't follow the lines, and it will just go across the map. On the other hand, focusing on lines that are too close won't give the robot time to adapt to changes in the line.

It's also vital to optimize the region of the image as a result of cropping. If it's too big, too much data will be processed, making your program too slow. On the other hand, it has to have enough data to work with. You will have to adapt your image routinely.

2.2 Second Step: Convert from BGR to HSV

Remember that OpenCV works with BGR instead of RGB for historical reasons (some cameras, when OpenCV was created, worked with BGR).

It seems that it is not very easy to work with either RGB or BGR, to differentiate colors. That's why HSV is used. The idea behind HSV is to remove the component of color

saturation. This way, it's easier to recognize the same color in different light conditions, which is a serious issue in image recognition.

```
# Convert from RGB to HSV

hsv = cv2.cvtColor(crop_img, cv2.COLOR_BGR2HSV)

lower_yellow = np.array([20,100,100])

upper_yellow = np.array([50,255,255])
```

In the code above, you're converting the cropped_image (crop_img) into HSV. Next, you're selecting which color in HSV you want to track

Finally, select an upper and lower bound to define the yellow region of the cone base. The bigger the region, the more gradients of your chosen color will be accepted. This will depend on how your robot detects the color variations in the image and how critical it is mixing similar colors.

2.3 Third Step: Apply the Mask :

Generate a version of the cropped image in which you only see two colors: black and white. The white will be all the colors you consider yellow, and the rest will be black. It's a binary image.

Why do you need to do this? It has two functions:

- **In doing this, you don't have continuous detection. It is the color, or it's NOT; there's no in-between. This is vital for the centroid calculation that will be done later because it only works on the principle of YES or NO.**
- **Second, it will allow the generation of the resulting image afterward. You extract everything on the image except the color line, seeing only what you are interested in seeing.**

```
# Threshold the HSV image to get only yellow colors

mask = cv2.inRange(hsv, lower_yellow, upper_yellow)
```

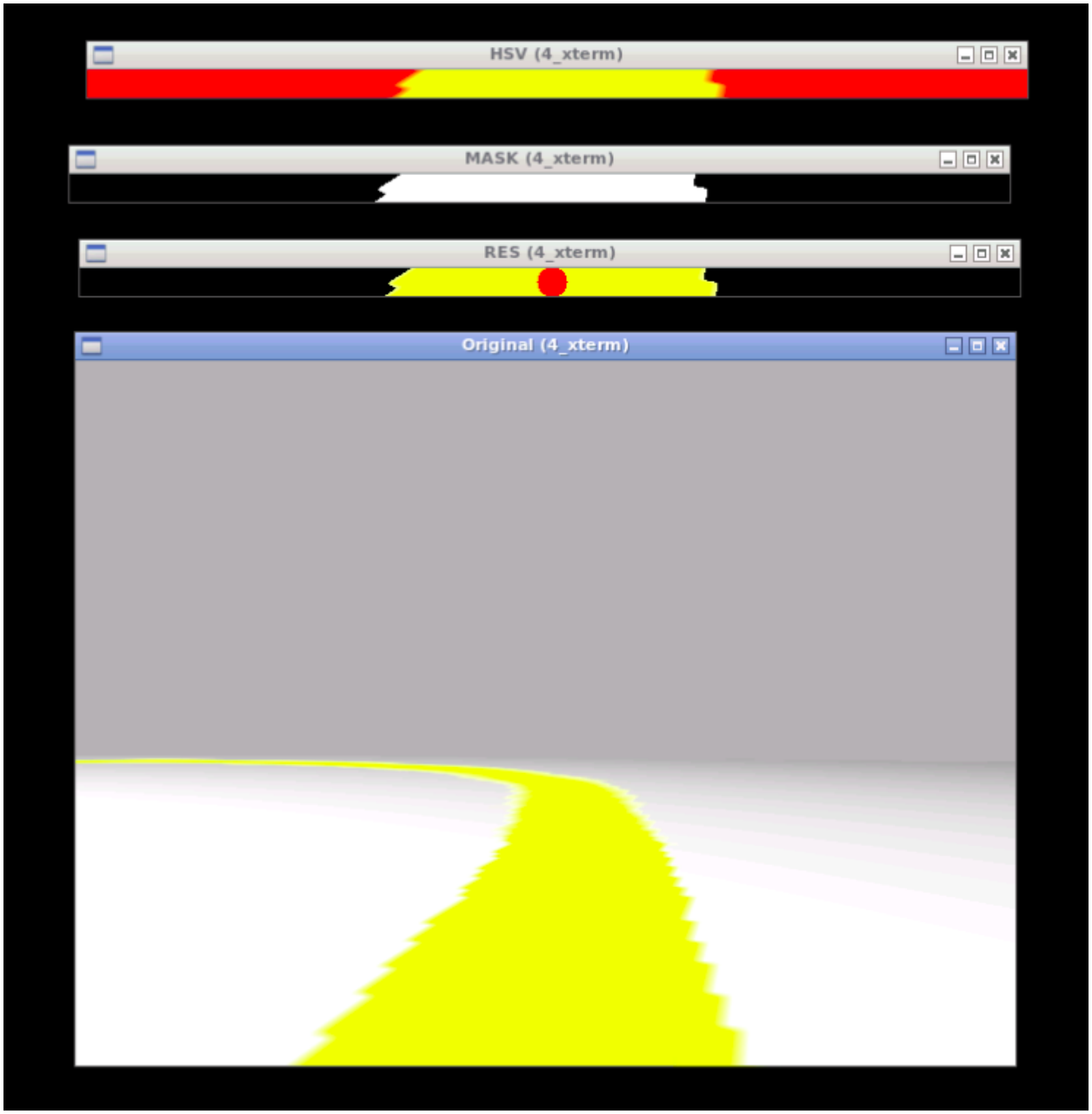
```
# Bitwise-AND mask and original image
res = cv2.bitwise_and(crop_img,crop_img, mask= mask)
```

Merge the cropped, colored image in HSV with the binary mask image. This step will color only the detections, leaving the rest black.

2.4 Fourth Step: Get the Centroids, Draw a Circle for the Centroid and show the Images:

```
# Calculate centroid of the blob of binary image using ImageMoments
m = cv2.moments(mask, False)
try:
   cx, cy = m['m10']/m['m00'], m['m01']/m['m00']
except ZeroDivisionError:
   cy, cx = height/2, width/2

cv2.circle(res,(int(cx), int(cy)), 10,(0,0,255),-1)
cv2.imshow("Original", cv_image)
cv2.imshow("HSV", hsv)
cv2.imshow("MASK", mask)
cv2.imshow("RES", res)
cv2.waitKey(1)
```

As you can see here, you will obtain the cropped image coordinates where there is detection where the centroid of the positive yellow color occurs. If nothing is detected, it will be positioned in the center of the image.


HSV (4_xterm)
MASK (4_xterm)
RES (4_xterm)
Original (4_xterm)

2.5 Move the TurtleBot Based on the Position of the Centroid

```
error_x = cx - width / 2;
twist_object = Twist();
twist_object.linear.x = 0.2;
twist_object.angular.z = -error_x / 100;
rospy.loginfo("ANGULAR VALUE SENT===>"+str(twist_object.angular.z))
self.movekobuki_object.move_robot(twist_object)
```

This movement is based on proportional control. This means that the TurtleBot will oscillate a lot and probably have an error. But, it is the simplest way to move the robot and get the job done. It gives a constant linear movement, and the angular Z velocity depends on the difference between the centroid center in X and the center of the image.

Final Script follow_line_step_hsv.py

```
#!/usr/bin/python3
import rospy
import cv2
import numpy as np
from cv_bridge import CvBridge, CvBridgeError
from geometry_msgs.msg import Twist
from sensor_msgs.msg import Image
from move_robot import MoveKobuki

class LineFollower(object):

    def __init__(self):

        self.bridge_object = CvBridge()
        self.image_sub =
rospy.Subscriber("/camera/rgb/image_raw",Image,self.camera_callback)
        self.movekobuki_object = MoveKobuki()
```

```
    def camera_callback(self,data):

        try:
            cv_image = self.bridge_object.imgmsg_to_cv2(data, desired_encoding="bgr8")
        except CvBridgeError as e:
            print(e)
        height, width, channels = cv_image.shape
        descentre = 160
        rows_to_watch = 20
        crop_img =
cv_image[(height)/2+descentre:(height)/2+(descentre+rows_to_watch)][1:width]

        # Convert from RGB to HSV
        hsv = cv2.cvtColor(crop_img, cv2.COLOR_BGR2HSV)

        lower_yellow = np.array([20,100,100])
        upper_yellow = np.array([50,255,255])

        # Threshold the HSV image to get only yellow colors
        mask = cv2.inRange(hsv, lower_yellow, upper_yellow)

        # Calculate centroid of the blob of binary image using ImageMoments
        m = cv2.moments(mask, False)
        try:
            cx, cy = m['m10']/m['m00'], m['m01']/m['m00']
        except ZeroDivisionError:
            cy, cx = height/2, width/2



        # Bitwise-AND mask and original image
        res = cv2.bitwise_and(crop_img,crop_img, mask= mask)

        # Draw the centroid in the resultut image
        # cv2.circle(img, center, radius, color[, thickness[, lineType[, shift]]])
        cv2.circle(res,(int(cx), int(cy)), 10,(0,0,255),-1)

        cv2.imshow("Original", cv_image)
        cv2.imshow("HSV", hsv)
        cv2.imshow("MASK", mask)
        cv2.imshow("RES", res)
         cv2.waitKey(1)
```

```
        error_x = cx - width / 2;
        twist_object = Twist();
        twist_object.linear.x = 0.2;
        twist_object.angular.z = -error_x / 100;
        rospy.loginfo("ANGULAR VALUE SENT===>"+str(twist_object.angular.z))
        # Make it start turning
        self.movekobuki_object.move_robot(twist_object)

    def clean_up(self):
        self.movekobuki_object.clean_class()
        cv2.destroyAllWindows()



def main():
    rospy.init_node('line_following_node', anonymous=True)
     line_follower_object = LineFollower()

    rate = rospy.Rate(5)
    ctrl_c = False
    def shutdownhook():
        line_follower_object.clean_up()
        rospy.loginfo("shutdown time!")
        ctrl_c = True

    rospy.on_shutdown(shutdownhook)

    while not ctrl_c:
        rate.sleep()



if __name__ == '__main__':
    main()
```

Move_robot.py

```
#!/usr/bin/python3
import rospy
from geometry_msgs.msg import Twist

class MoveKobuki(object):
    def __init__(self):
        self.cmd_vel_pub = rospy.Publisher('/cmd_vel', Twist, queue_size=10)
        self.last_cmdvel_command = Twist()
        self._cmdvel_pub_rate = rospy.Rate(10)
        self.shutdown_detected = False

    def move_robot(self, twist_object):
        self.cmd_vel_pub.publish(twist_object)

    def clean_class(self):
        # Stop Robot
        twist_object = Twist()
        twist_object.angular.z = 0.0
        self.move_robot(twist_object)
        self.shutdown_detected = True

def main():
    rospy.init_node('move_robot_node', anonymous=True)
    movekobuki_object = MoveKobuki()
    twist_object = Twist()
    # Make it start turning
    twist_object.angular.z = 0.5
    rate = rospy.Rate(5)

    ctrl_c = False
    def shutdownhook():
        movekobuki_object.clean_class()
        rospy.loginfo("shutdown time!")
        ctrl_c = True
    rospy.on_shutdown(shutdownhook)
    while not ctrl_c:
        movekobuki_object.move_robot(twist_object)
        rate.sleep()

    if __name__ == '__main__':
    main()
```