|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Task. No.: | 9 | Points: | 10 | Turtlebot3 Line tracking |

|  |
| --- |
| Objectives:  1. Using OpenCV. 2. Using CV\_bridge. 3. Working with images. |

|  |
| --- |
| Description: This project aims to get a message from an Image topic in ROS, convert it to an OpenCV Image to enable the robot to follow a line. To do so, we will need to see the line, define the center of this line and finally move the robot to follow the robot. Messages from the camera are published on the *sensor\_msgs/Image* topic so we will need to write a node that subscribe to the same topic. |

|  |  |
| --- | --- |
| Step | Action |
| 1 | Unzip the *line\_follower* file in your *catkin\_ws* folder. |
| 2 | Create a python file called *line\_follower.py* and copy this code in it.  #!/usr/bin/env python3 import rospy, cv2, cv_bridge, numpy  from sensormsgs.msg import Image  from geometrymsgs.msg import Twist  from move_robot import MoveKobuki class Follower: def_init_ (self): # set up ROS / OpenCV bridge  self.bridge = cv_bridge.CvBridge()  self.movekobukiobject = MoveKobuki() # initalize the debugging window  cv2.namedWindow("window", cv2.WINDOW_NORMAL) # subscribe to the robot's RGB camera data stream self.image_sub = rospy.Subscribed'camera/rgb/imageraw', Image, self.image_callback) def image_callback(self, msg): # converts the incoming ROS message to OpenCV format and HSV (hue, saturation, value)  image = self.bridge.imgmsg_to_cv2(msg,desired_encoding=,bgr8') hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV) # TODO: define the upper and lower bounds for what should be considered ’yellow’  lower_yellow = numpy.array([10, 50, 100]) #TODO upper_yellow = numpy.array([45, 255, 255])  #TODO mask = cv2.inRange(hsv, lower_yellow, upper_yellow)  # this erases all pixels that aren’t yellow  h, w, d = image.shape searchtop = int(3*h/4)  searchbot = int(3*h/4 + 20)  mask[0:searchtop, 0:w] = 0  mask[search_bot:h, 0:w] = 0 # using momentsQ function, the center of the yellow pixels is determined  M = cv2.moments(mask) # if there are any yellow pixels found  if M[’m00’] >0: # center of the yellow pixels in the image  cx = int(M[’ml0']/M['m00']) cy = int(M['m01*]/M[*m00']) # a red circle is visualized in the debugging window to indicate # the center point of the yellow pixels  cv2.circle(image, (cx, cy), 20, (0,0,255), -1) # TODO: based on the location of the line (approximated # by the center of the yellow pixels), implement # proportional control to have the robot follow # the yellow line # shows the debugging window  cv2.imshow("window", image)  cv2.waitKey(3)  error_x = ex - w / 2 twistobject = Twist()  twist_object.linear.x = 0.2  twistobject.angular.z = -errorx / 1000 rospy.loginfo("ANGULAR VALUE SENT===>"+str(twist_object.angular.z))  # Make it start turning self.movekobuki_object.move_robot(twist_object) def run(self): rospy.spin() if _name == '_main_' : rospy.init_node('linefollower')  follower = Follower()  follower.run() |
| 3 | Create a python file called *move\_robot.py* and copy this code in it.  #!/usr/bin/env 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, twistobject): self.cmd_vel_pub.publish(twist_object) def clean_class(self): # Stop Robot  twist_object = Twist()  twistobject.angular.z = 0.0  self.move_robot(twist_object)  self.shutdowndetected = True def main(): rospy.init_node('move_robot_node’, anonymous=True) movekobuki_object = MoveKobukiQ twistobject = Twist() # Make it start turning  twist_object.angular.z =0.5  rate = rospy.Rate(5) ctrl_c = False  def shutdownhook() : # works better than the rospy.is_shut_down()  movekobukiobject.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() |
| 4 | To see the environment run this code.  smoz@MRL:/~/catkin_ws$ roslaunch line_follower robot_and_line_sim.launch  path of travel modelled out |
| 4 | Use the Turtlebot3 Burger model. It has two sensors/actuators:  •Laser: The Turtlebot3 robot is equipped with a 360º LiDAR laser. The data of this laser is published into the */scan* topic.  •Wheels: The Turtlebot3 robot receives velocity commands in order to move. This data has to be published into the */cmd\_vel* topic. |
| 5 | Use the keyboard\_teleop.launch launch to move the robot. |
| 6 | Create a map of the environment, using slam\_gmapping node provided by the ROS Navigation package.  To visualize the mapping process, use RViz tool.  Save my\_map.pgm and my\_map.yaml and show their content. |
| 7 | Use OpenCV for line tracking based on color filtering. Run the following code to open Gazebo.  smoz@MRL:/~$ roslaunch line_follower robot_and_line_sim.launch |
| 8 | Run the following code to see the line from the robot point of view.  smoz@MRL:/~/catkin_ws$ rosrun line_follower line_follower.py  red dot following a yellow path |
| 9 | Change lower and upper thresholds to select which color in HSV you want to track. The red point shows the center of the line.  turtle following a yellow path |

 This is shared under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Public License