EE 3002 L1 (Junior Design Studio - Robotics)

Spring 2025 - LAB 5

SOLUTION

Task 1: Understanding Robomaster Camera [20 MARKS]

For tasks 1.1, 1.2 , 1.3, follow steps in the manual.

For task 1.4, The python script **marker_detection.py**:

```
# -*-coding:utf-8-*-
import cv2
import robomaster
from robomaster import robot
from robomaster import vision
import math
import numpy as np



class MarkerInfo:

    def __init__(self, x, y, w, h, info):
        self._x = x
        self._y = y
        self._w = w
        self._h = h
        self._info = info

    @property
    def pt1(self):
        return int((self._x - self._w / 2) * 1280), int((self._y - self._h / 2) *
720)

    @property
    def pt2(self):
        return int((self._x + self._w / 2) * 1280), int((self._y + self._h / 2) *
720)
```

```
    @property
    def center(self):
        return int(self._x * 1280), int(self._y * 720)

    @property
    def text(self):
        return self._info


wr = 0.15 # real width of marker [meters]
markers = []

def calculate_distance(wp, wr, fc, x):
    distance = (wr * fc) / (wp*1280)
    # print("Distance from marker: ", distance)

    # calculating the angle from robot
    FOV = 120
    image_pw = 1280 # pixel width of the image
    angle_per_pixel = FOV / image_pw
    center_x = image_pw / 2
    angle = ((x * image_pw) - center_x) * angle_per_pixel
    angle = max(-60, min(60, angle))
    # print("Angle to the marker: ", angle)

    # next, we find the x and y cordinates of the marker wrt robot frame
    marker_x = distance * math.cos(math.radians(angle))
    marker_y = distance * math.sin(math.radians(angle))

    return marker_x, marker_y

def detect_largest_contour(img):
    # Convert BGR to HSV
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    # Define the color range for the mask
    lower_green = np.array([36, 0, 0])
    upper_green = np.array([86, 255, 255])
```

```
    # Create the mask
    mask = cv2.inRange(hsv, lower_green, upper_green)

    # Find contours in the masked image
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    # Find the largest contour
    largest_contour = max(contours, key=cv2.contourArea)

    # Get the x, y coordinates, width, and height of the largest contour
    x, y, w, h = cv2.boundingRect(largest_contour)

    return x, y, w, h

def on_detect_marker(marker_info):
    number = len(marker_info)
    markers.clear()
    for i in range(0, number):
        x, y, w, h, info = marker_info[i]
        markers.append(MarkerInfo(x, y, w, h, info))
        print("marker:{0} x:{1}, y:{2}, w:{3}, h:{4}".format(info, x, y, w, h))

        x_cordinate, y_cordinate = calculate_distance(w, wr, 630, x)
        print("Cordinates of Marker in robot frame: ", x_cordinate, y_cordinate)
        # distance = calculate_distance(w, wr, 630)
        # print("Distance from marker: ", distance)

        # # calculating the angle from robot
        # FOV = 120
        # image_pw = 1280 # pixel width of the image
        # angle_per_pixel = FOV / image_pw
        # center_x = image_pw / 2
        # angle = ((x * image_pw) - center_x) * angle_per_pixel
        # print("Angle to the marker: ", angle)


        # angle = max(-60, min(60, angle))

        # # next, we find the x and y cordinates of the marker wrt robot frame
```

```
        # marker_x = distance * math.cos(math.radians(angle))
        # marker_y = distance * math.sin(math.radians(angle))

        # print("Cordinates of Marker in robot frame: ", marker_x, marker_y)

if __name__ == '__main__':
    ep_robot = robot.Robot()
    ep_robot.initialize(conn_type="ap")

    ep_vision = ep_robot.vision
    ep_camera = ep_robot.camera

    ep_camera.start_video_stream(display=False)
    result = ep_vision.sub_detect_info(name="marker", callback=on_detect_marker)

    for i in range(0, 10000):
        img = ep_camera.read_cv2_image(strategy="newest", timeout=0.5)
        for j in range(0, len(markers)):

            cv2.rectangle(img, markers[j].pt1, markers[j].pt2, (255, 255, 255))
            cv2.putText(img, markers[j].text, markers[j].center,
cv2.FONT_HERSHEY_SIMPLEX, 1.5, (255, 255, 255), 3)
        cv2.imshow("Markers", img)
        cv2.waitKey(1)
    cv2.destroyAllWindows()

    result = ep_vision.unsub_detect_info(name="marker")
    cv2.destroyAllWindows()
    ep_camera.stop_video_stream()
    ep_robot.close()
```

Task 2: Detecting and Gripping an Object [15 MARKS]

The python script **grip_obj.py**:

```
# -*-coding:utf-8-*-
# Copyright (c) 2020 DJI.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License in the file LICENSE.txt or at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

import numpy as np
import time
import robomaster
from robomaster import robot
import cv2
from math import sin,cos


def getlargestcountour(image):

    # Convert BGR to HSV
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # define range of green color in HSV
    lower_green = np.array([40,50,50])
    upper_green = np.array([86,255,255])

    # Creating a mask. Thresholding the HSV image to get only green colors
    mask = cv2.inRange(hsv, lower_green, upper_green)

    # Find contours
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
```

```
    # Pick the largest contour
    largest_contour = max(contours, key=cv2.contourArea)

    # Get the bounding box coordinates, width, and height of the largest contour
    x, y, w, h = cv2.boundingRect(largest_contour)

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

    # display the mask and masked image
    cv2.imshow('Mask',mask)
    cv2.waitKey(0)
    cv2.imshow('Masked Image',result)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    # Print the coordinates, width, and height of the bounding box
    print("X coordinate:", x)
    print("Y coordinate:", y)
    print("Width:", w)
    print("Height:", h)

    return getxy(x,h)


def getxy(x,h):
    distance = (0.135 * 630) / h
    Angle = (x*(120/1280)) - 60
    X_coordinate = distance*sin((Angle))
    Y_coordinate = distance*cos((Angle))
    print("Distance .. Angle .. x_coord .. y_coord ..
",[distance,Angle,X_coordinate,Y_coordinate])
    return [distance,Angle,X_coordinate,Y_coordinate]


# def sub_position_handler(position_info):
#     global x
#     global y
#     x, y, _ = position_info
#     print("chassis position: x:{0}, y:{1}".format(x, y))
```

```
if __name__ == '__main__':
    ep_robot = robot.Robot()
    ep_robot.initialize(conn_type="ap")

    ep_camera = ep_robot.camera
    ep_camera.start_video_stream(display=False)

    ep_chassis = ep_robot.chassis
    # ep_chassis.sub_position(freq=10, callback=sub_position_handler)

    while True:
        img = ep_camera.read_cv2_image(strategy="newest")
        # cv2.imshow("Robot", img)
        coords = getlargestcountour(img)
        print(coords)

        ep_chassis.move(x=(coords[0]-0.2), y=0, z=0,
xy_speed=0.2).wait_for_completed()
        # ep_chassis.move(x=0.1, y=0, z=0, xy_speed=0.2).wait_for_completed()
        # ep_chassis.move(x=0, y=(coords[3]), z=0,
xy_speed=0).wait_for_completed()

        # if coords[0] <= 0.01:
        ep_gripper = ep_robot.gripper
        ep_gripper.open(power=50)
        time.sleep(1)
        ep_gripper.pause()
        ep_gripper.close(power=50)
        time.sleep(1)
        ep_gripper.pause()

        ep_chassis.move(x=-1, y=0, z=0, xy_speed=0.5).wait_for_completed()

        cv2.waitKey(1)
        time.sleep(1)
        break

    cv2.destroyAllWindows()
    ep_camera.stop_video_stream()
```

```
    ep_robot.close()


# Greyscale image is 2d array (8 bit encoded from 0 to 255)
# RGB image is 3d array (3 2d arrays stacked on top of each other)

#Image masking done to change background of images (setting some pixels to 0)

#edges are points where pixel values are rapidly changing (i.e black to white)
#Canny-edge detection detects edges (and hence complex shapes)
#Contour detection detects irregular shapes as well quite easily
# first it makes a color mask, i.e. for a red ball, set all other pixels to 0
# largest contour is ur obj, other contours is noise
# uniform color obj , calculate its width in img, and use its width in real world
to calculate distance from camera, and grab it

#Hough transform  to identify complex shapes within an image
```

Task 3: Visual Odometry [15 MARKS]

The python script **vis_odom.py**:

```
#!/usr/bin/env python3
import rospy
import cv2
import time
from robomaster import robot
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError

class ImagePublisher(object):
    def __init__(self):
        self.node_rate = 10  # Set the rate to 10 Hz
        self.image_pub = rospy.Publisher("image", Image, queue_size = 10)
        self.bridge = CvBridge()

    def publish_image(self, img):
```

```
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        cropped = gray[:-150, :]
        try:
            ros_img = self.bridge.cv2_to_imgmsg(cropped)
        except CvBridgeError as e:
            print(e)
        self.image_pub.publish(ros_img)

    def run(self):
        loop = rospy.Rate(self.node_rate)
        while not rospy.is_shutdown():
            self.publish_image()
            loop.sleep()

if __name__ == '__main__':
    rospy.init_node('image_publisher', anonymous=True)
    ep_robot = robot.Robot()
    ep_robot.initialize(conn_type = "rndis")
    ep_camera = ep_robot.camera
    ep_camera.start_video_stream(display = False)

    publisher = ImagePublisher()

    while not rospy.is_shutdown():
        img = ep_camera.read_cv2_image(strategy = "newest", timeout = 2)
        publisher.publish_image(img)

    ep_camera.stop_video_stream()
    ep_robot.close()
```