Introduction to Vision and Robotics 2020-21

Lab 1

Installation and Blob Detection

October 2020

Over these demonstrations you shall be learning to control a 3-link 2D robot using computer vision. You will be programming in the python language using ROS and OpenCV. In the first practical, you will employ computer vision to detect a robot's joints.

1 Installation

This installation guide has been tested on Ubuntu 20.04 with ROS distribution Noetic. You can use VirtualBox to install Ubuntu, which will allow you to have a virtual machine of Ubuntu on your laptop. If you do not have access to a computer or laptop please contact your PT. The University has laptops available for long term loan over the year for new and continuing undergraduate students. Please follow the guide below for installation of lab material.

1.1 Installation

1. Download and install Ubuntu 20.04 on your PC/laptop. You can follow the instructions provided on ROS Video Tutorials on Learn.
2. After you have installed Ubuntu, follow the instruction at `http://wiki.ros.org/Installation/Ubuntu` to install a Full-Desktop package of ROS.
3. Install OpenCV-python. You can follow the instructions given here:
   `https://pypi.org/project/opencv-python/`
4. (optional) Install Python IDE of your choice (e.g., PyCharm).

1.2 Installation of Lab Library

1. Create a folder in your home directory, here we named the folder **catkin_ws**, enter the folder and create another folder named **src**. You can do it via the terminal:
   mkdir catkin_ws
   cd catkin_ws
   mkdir src

2. The package for the lab is in a repository (repo) on Github which you can find through the following link: `https://github.com/mohsenkhadem1/ivr_lab.git`. You can download it, unzip, and put in a folder named **ivr_lab** inside the **src** folder you just created. **Important:** Make sure the folder that includes the downloaded files is named **ivr_lab** and is inside the **src** folder.

3. Get back to your workspace folder **cd ~/catkin_ws**. Use the following command to install the package you just downloaded:
   catkin_make
   source devel/setup.bash
   Important: The setup.bash file will have to be sourced as above in every new terminal. You can also add this to the end of your .bashrc file to automatically do it whenever a new terminal is opened. To do so you can open **/.bashrc** using **gedit /.bashrc**. Add the following two lines at the end of the opened file and save:
   source /opt/ros/noetic/setup.bash
   source ~/catkin_ws/devel/setup.bash

2 Overview of the Simulator

Once the installation is completed you can navigate to the **~/catkin_ws** folder and run the robot simulator using:
roslaunch ivr_lab spawn.launch

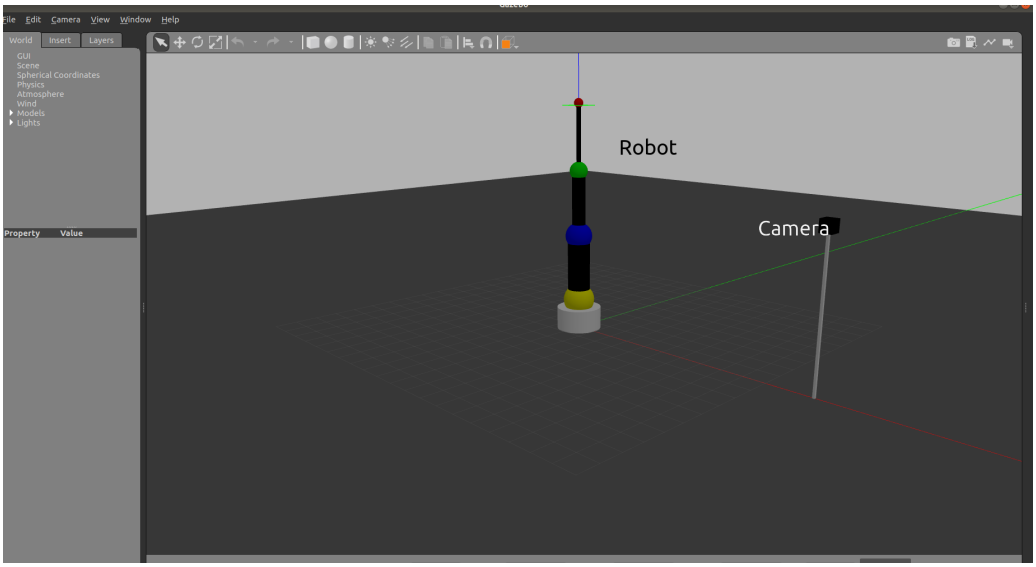

Figure 1: The robot in Gazebo simulation environment.

This opens the simulated robot in ROS simulation environment called **Gazebo** that shall be used for the lab sessions. The robot has three links and moves in a 2D world, (x and y dimensions). The joints of the robot are shown by yellow (joint 1), blue (joint 2), and green (joint3) spheres. The red sphere is the robot end effector. The three spheres are located at the end of each link so that the joint angles angles can be calculated.

The robot is in position control mode. You can move the robot's joints using the following command in a new terminal:

rostopic pub -1 /robot/joint1_position_controller/command std_msgs/Float64 "data: 1.0"

This line of code will move the joint 1 by 1 Radian. You can start to get familiar with the simulator by changing the joint angles. If you want to move another joint change the joint number in the above code. Each joint can be moved between $-\pi/2$ and $\pi/2$.

The simulator is also designed to return an RGB array from the camera placed in front of the robot. This image is what you will be performing computer vision algorithms on. The main code which you shall use in the labs is inside **~/catkin_ws/src/ivr_lab/src** and named **image_processing.py**. You can navigate to that folder and open it using the following command lines:

cd ~/catkin_ws/src/ivr_lab/src

gedit image_processing.py

```
  # Defines publisher and subscriber
  def __init__(self):
    # initialize the node named image_processing
    rospy.init_node('image_processing', anonymous=True)
    # initialize a publisher to send messages to a topic named image_topic
    self.image_pub = rospy.Publisher("image_topic",Image, queue_size = 1)
    # initialize a publisher to send joints' angular position to a topic called joints_pos
    self.joints_pub = rospy.Publisher("joints_pos",Float64MultiArray, queue_size=10)
    # initialize the bridge between openCV and ROS
    self.bridge = CvBridge()
    # initialize a subscriber to recieve messages rom a topic named /robot/camera1/image_raw and use callback function to recieve data
    self.image_sub = rospy.Subscriber("/robot/camera1/image_raw",Image,self.callback)

  # Recieve data, process it, and publish
  def callback(self,data):
    # Recieve the image
    try:
      cv_image = self.bridge.imgmsg_to_cv2(data, "bgr8")
    except CvBridgeError as e:
      print(e)
```

Your code goes here

```
    # Perform image processing task (your code goes here)
    # The image is loaded as cv_img

    # Uncomment if you want to save the image
    #cv2.imwrite('image_copy.png', cv_image)

    cv2.imshow('window', cv_image)
    cv2.waitKey(3)

    # change te value of self.joint.data to your estimated value from thew images once you have finalized the code
    self.joints = Float64MultiArray()
    self.joints.data = np.array([0, 0, 0])

    # Publish the results - the images are published under a topic named "image_topic" and calculated joints angles are published under a topic named "joints_pos"
    try:
      self.image_pub.publish(self.bridge.cv2_to_imgmsg(cv_image, "bgr8"))
      self.joints_pub.publish(self.joints)
    except CvBridgeError as e:
      print(e)

# call the class
def main(args):
  ic = image_converter()
  try:
    rospy.spin()
  except KeyboardInterrupt:
    print("Shutting down")
  cv2.destroyAllWindows()

# run the code if the node is called
if __name__ == '__main__':
    main(sys.argv)
```

Python ▾ Tab

Figure 2: Image processing code.

This code receives the image from the camera, processes it and publishes the results. The image processing code goes inside the **callback** function. Currently it saves the image under the name **cv_image** and shows it using **imshow** command. To run this code, first you must make it executable. Open up a new terminal and use these commands:

cd ~/catkin_ws/src/ivr_lab/src

chmod +x image_processing.py

Now to run the code you can type:

rosrun ivr_lab image_processing.py

It should show the image received from the camera.

Try to move the robot and check if the robot moves in the camera view. You can modify the code shown in

black block in Fig. 2 to save the image file. Use the **cv2.imwrite** command to save the image. It should be saved in your **catkin_ws** folder.

3 The Labs' Overall Goal

Over these sets of labs you will be working on how to control the 3 link 2D robot using image feedback. To control robots of this type we normally have several steps to perform:

1. Obtain the current joint state of the robot: This is required to control the robot as we need a sense of where the robot is in the world and what the robot is doing. Primarily this includes the current joint angles of the robot. Using this information we can control the robot through various types of controllers. This will be the first source of information obtained using vision in this lab and the next.

2. Determine the desired positions: The next step normally is to decide on where the robot is to go next. This can be written either in the joint space of the robot in terms of the joint angles of each link, or it can be written in the world coordinate system as you will see in lab 3. For the purpose of this lab you can set some joint angles as have been shown in previous section.

3. Calculate the error between the joint states and the desired positions This step involves determining which way and to what magnitude the robot should move in, as such it generates a velocity. This is important to control the robot.

4. Feed this into the robot: This is the final step which involves feeding the error and the joint state into a controller for the robot to make it move. In this case the robot implements a simple PD controller which you shall learn later on in the lectures. For this lab though the only information it requires is the current and desired joint angles and the joint velocities of the robot.

4 Detecting the joint states - Blob Detection

4.1 Purpose:

For the robot to act in the world it must first know where it is in the world. The task is to get the robot to know where it is from the joint states of its links from using the RGB array obtained from the camera. From here the robot can then determine in future labs how to reach the target.

4.2 Method:

Now that installation is done the rest of this lab session will be focused on segmenting the three coloured joints and detecting the state of each. The joint state describes the robot in joint angles. This will be broken down into a series of tasks: 1. Isolate the different joints in the image 2. Detect the center of each circle on the joints 3. Use trigonometry to calculate the joint angles You can use the OpenCV library for this course, however, do not use the SimpleBlobDetector implemented in OpenCV. You can achieve the same effect by writing your own blob detector.

Isolate the different joints in the image This will involve isolating the different colours in the methods, detect yellow, detect green, detect blue, detect red. This is achieved by getting three binary images with each

circle being highlighted in one image each. This can be achieved by using a thresholding technique on the RGB array. There are python code examples on the Learn page of doing this.

Detect the center of each circle on the joints Once the detect colour methods are successful in detection, you will need to expand this function so that it returns the centre point of the binary images. This should be returned as an (x,y) coordinate in metres with the respect to the center of the image (remember to use coordinate conert). It is advised to look at the documentation for OpenCV moments for this: `https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_contours/py_contour_features/py_contour_features.html#contour-features`. To turn the pixel values to meters you can use the information from the robot. For instance, in this case we know the length of each link (distance between two centers) is 3 meters.

Use trigonometry to calculate the joint angles You will have to fill in the detect joint angles method. To do this first get the centre points of the joints (the coloured circles) by calling the detect colour methods within this method. When you have the desired centre points you can then use trigonometry to find the joint angle for that specific link. Remember to consider which frame (i.e. what counts as 0 to that link) you are working in when calculating the angles. Once you have calculated the angles you can publish it under the variable name **joints.data**. In the code it is currently set to zeros.

5 Outcome

The outcome of this section of the lab should be that the arm should move towards a desired joint configuration and you should be able to measure those joints values.

Once you have finished your code run it as explained in Section 2 (**rosrun ivr_lab image_processing.py**). You can see the values you measured by typing the following command in a new terminal: **rostopic echo /joints_pos**

Move the robot to a certain position using joint moving commands described in section 2 (**rostopic pub -1 /robot/joint1_position_controller/command std_msgs/Float64 "data: 1.0"**). The result you get from image processing should be roughly the same as the command you gave the robot. Try to figure out sources of error.

6 Hints

Initially do play around with the simulator. This will help you to understand how to interact with the robot and how the links operate with respect to their working frame. To perform the blob detection of each joint you may want to look at the methods **inRange**, **erode**, **dilate**, and **moments** in OpenCV python.

If your computer is slow, you can close the Gazebo window. You can close it by clicking on the red cross button on top of Gazebo window, do not close it via the terminal.

The joint angles of each joint will be between $-\pi/2$ and $\pi/2$. Also, try not to move the robot into positions which might hit the ground.

If you are not comfortable modifying the **image_processing.py** code, save one image of the robot using **imwrite** command. Load it in Pycharm, write your code in there, after you have ensured it works correctly, paste it inside the **image_processing.py** file.

The ROS commands used in the lab and ROS library might have been confusing. In future class sessions we will explain the structure of ROS and these commands. You can also check ROS and OpenCV-Python tutorials here:

`http://wiki.ros.org/ROS/Tutorials`

`https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html`.