Introduction to Vision and Robotics 2021-2022

Assignment

The assignment is due by 2021-11-26 16:00

This assignment will be extending a lot of what was previously done in the labs to a non-planar (3D) robot with 4 degrees of freedom. You will be working in groups of two. There will be, however, no extra credit for completing the assignment alone. The assignment is going to be broken down into 2 sections that cover robot vision and control.

1 Overview of the Simulator

1.1 Installation of Lab Library

1. Use UBUNTU as instructed in labs.
2. 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
3. The package for the lab is in a repository (repo) on Github which you can find through the following link: https://github.com/AQQQQY/ivr_assignment.git You can download it, unzip, and put in a folder named **ivr_assignment** inside the **src** folder you just created. **Important:** Make sure the folder that includes the downloaded files is named **ivr_assignment** and is inside the **src** folder.
4. 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/melodic/setup.bash
   source ~/catkinws/devel/setup.bash
5. Once the installation is completed you can navigate to the **~/catkinws/src/ivr_assignment/src** folder and run the following command to make the python files executable:
   chmod +x image1.py image2.py
   Note: If you create any python files in this folder you should make them executable using **chmod** command.

Once the installation is completed you can navigate to the ∼/**catkin_ws**folder and run the robot simulator using:

roslaunch ivr_assignment spawn.launch

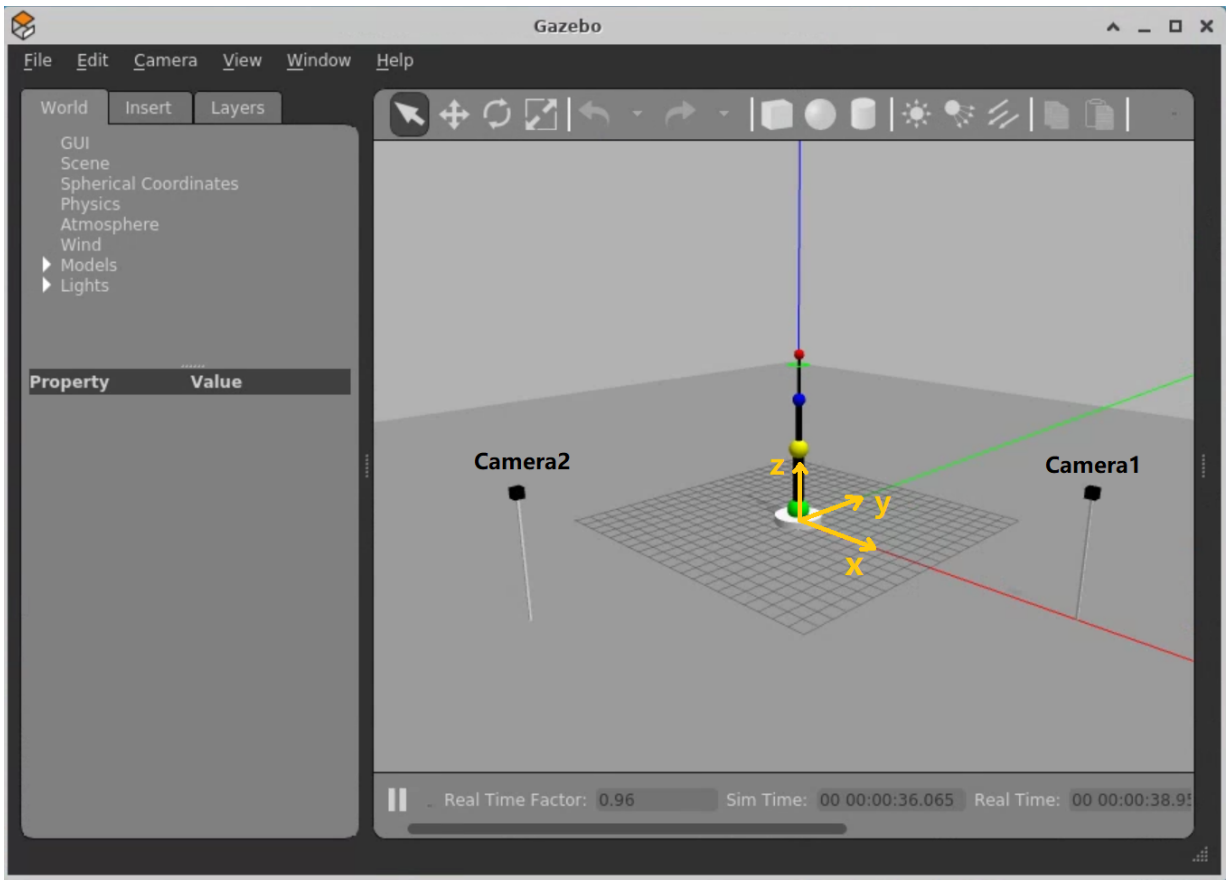

Figure 1: The robot in Gazebo simulation environment.

This opens the simulated robot in ROS simulation environment. The robot has 4 degrees of freedom and moves in a 3D world. The joints of the robot are shown by green (joint 1), yellow (joint 2 & 3), and blue (joint4) spheres. The red sphere is the robot end effector. Joint 1 is a revolute joint rotating around its z axis. Joint 2 and 3 are located at the yellow circle and are revolute joints moving around the drawn y and x axes, respectively. Joint 4 is a revolute joint rotating around the y axis. The table below gives a description for each link:

Link	Length	Rotation axis	Joint location
1	4[m]	z	Green
2	0[m]	y	Yellow
3	3.2[m]	x	Yellow
4	2.8[m]	y	Blue

The robot is in position control mode. You can move the robot's joints using the following command in anew 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. Joint 1 can be moved between $-\pi$ and π and the rest of the joint can be moved between $-\pi/2$ and $\pi/2$.

The simulator is also designed to return two RGB arrays from two cameras placed around the robot. The images from the cameras are what you will be performing computer vision algorithms on.

The main codes which you shall use in the labs is inside ∼**/catkin_ws/src/ivr_assignment/src** and named **image1.py** and **image2.py**.

This codes receive the images from the cameras, process it and publish the results. To run the codes, open up a new terminal and use these commands:

rosrun ivr_assignment image1.py

It should show the image received from camera 1.

rosrun ivr_assignment image2.py should show the image received from camera 2.

Try to move the robot and check if the robot moves in the camera views. You should modify these codes as you did in the labs.

2 Robot Vision (30 marks)

2.1 Joint state estimation - I (20 marks)

2.1.1 Problem statement

As part of the new simulation you are given two orthogonal views on the zy plane and the xz plane. Similar to the labs you will need to calculate the joint angles for the robot to use. For this part of the assignment assume joint 1 is fixed and robot can only move in a 3D world with 3 degree of freedom (DoF).

Next, create a node to move the other three joints using the following sinusoidal signals:

joint2= $\frac{\pi}{2} sin(\frac{\pi}{15} * t)$

joint3= $\frac{\pi}{2} sin(\frac{\pi}{20} * t)$

joint4= $\frac{\pi}{2} sin(\frac{\pi}{18} * t)$

where t denotes the time in seconds.

This can be done by publishing to topic "/robot/joint{x}_position_controller/command", where "{x}" is [2, 3, 4].

Create a python file named "vision_1.py" in **src** folder, use it to start a node that estimates the value of the 3 joints using computer vision. You may use whichever technique you want. Note that some of the spheres might not be clearly visible in one camera when the robot moves in 3D. Your algorithm should be able to handle these scenarios.

In your report, describe your algorithm (5 marks) and provide 3 plots (each 4 marks). In each plot, you should compare the estimated value of one joint with the sinusoidal signal that you used to move the joint. The plot should show the estimated joint values for at least 15 seconds. When comparing your estimated values with the sinusoidal signal, please describle what you observe using serveral sentences and analysis the reason of the errors, especially relatively big errors in some time periods (3 marks) .

2.1.2 Expected outcome

By the end your system should be able to accurately acquire the joint states of the robot (average error less than 0.15 radian error).

To plot the results you can publish your estimated joints on a topic and use **rqt_plot** command. We have used several codes in the labs to create a publisher and publish data. Check out the lab solutions if you don't know how to do it. To plot two topics vs each other you can use

rqt_plt [TOPIC1_NAME] [TOPIC2_NAME]/data[num]

use TOPIC1_NAME if the topic is only one number and [TOPIC2_NAME]/data[num] if the topic is an array.

To change the limits of rqt plot click on the arrow shaped button. Select a reasonable length to show the results, e.g. 15 seconds. You can save your plot by clicking on the save button. Additionally, you can record

your data using **rosbag record** command and plot them in Python. See the following link for more information http://wiki.ros.org/rosbag/Code%20API#py_api

2.2 Joint state estimation - II (10 marks)

Repeat the process described in section 2.1. This time do not fix joint 1, instead fix joint 2. Joint 1, 3 and 4 are free to move.

Similar to Section 2.1, create a node to move the joints using the following sinusoidal signals:

joint1=$\pi sin(\frac{\pi}{28} * t)$

joint3=$\frac{\pi}{2} sin(\frac{\pi}{20} * t)$

joint4=$\frac{\pi}{2} sin(\frac{\pi}{18} * t)$

where t denotes the time in seconds.

Create a python file named "vision_2.py" in **src** folder, use it to start a node that estimates the value of the 3 joint angles using computer vision. You may use whichever technique you want. In your report, describe your algorithm (4 marks) and provide 3 plots (each 2 marks). In each plot, you should compare the estimated value of one joint with the sinusoidal signal that you used to move the joint. The plot should show the estimated joint values for at least 10 seconds.

3 Robot Control (20 marks)

Joint 2 will be frozen throughout this section. Joint 1, 3 and 4 are free to move.

3.1 Forward Kinematics (10 marks)

In this section we will be using the joint angle estimations you have obtained from Section 2.2, with joint 2 frozen. Calculate the equations for robot Forward Kinematics (FK). We are only interested in controlling the x,y, and z position of the robot end-effector. Ignore the end-effector's orientation. The FK should look like this:

$$K(q) = \begin{bmatrix} x_e \\ y_e \\ z_e \end{bmatrix} \tag{1}$$

where x_e, y_e, z_e are the end-effector position and $q = (\theta_1, \theta_3, \theta_4)$.

Implement the FK in a new python file, "control.py". It should subscribe to topics published by "vision_2.py".

In your report, present the results of your Forward kinematics calculation (5 marks). To verify your FK using your code implementation, move the robot to 10 different points across it's workspace using rostopic pub command. Do not keep any of the 3 joints zero while selecting these 10 points. Compare the estimated end-effector position via the images to the estimated end-effector position via FK in a table and comment on accuracy (5 marks).

3.2 Inverse Kinematics (10 marks)

From the previous tasks you can acquire an estimation of the joint states. Calculate the Jacobian of the robot.

Our topic named /target_control/target_pos will publish a target position for the robot end-effector at a frequency of 50Hz. Your task is to calculate the necessary rotation of the each joint, so that the end-effector reach the target position. In the "control.py" you created in Section 3.1, develop a Controller similar to lab3 to follow the target position.

By the end, your robot should be moving to the target position. Not only this you should be able to report quantitatively the accuracy of the arm movement. Present the results of your inverse kinematics calculation (4 marks). Present three plots (2 marks for each) comparing the x,y, and z position of the robot end-effector with the x,y, and z position of the published target. The plot should show the positions for at least 20 seconds. The starting several seconds of your robot can be a random jiggly motion, you can ignore the motion and start your evaluation only after the robot converges to the target position.

4 Submission

Your submission on learn should be a compressed package containing your report PDF file and required 3 Python codes. File name of the package should be studentNo1_studentNo2.zip. Each group should only submit one time.

4.1 Report

At the beginning of the report, explain who has worked on what part of the project. Wildly imbalanced workload will be investigated through your GitHub commit history.

The main content of the report should include plots and short answers to the questions asked in the assignment. Your figures should be clear, readable, with legend, and all axis labeled properly. The report must be maximum of 4 pages, minimum of 1 cm margin on all sides, single spacing, and font of 11. Reports that don't follow these guidelines will not be marked. Exceeding pages will not be marked.

4.2 Code

Submit these 3 code files in your zip file:

1. vision_1.py: solution for Section 2.1. This file should initiate a node that subscribes to image topics:

   "/camera1/robot/image_raw"

   "/camera2/robot/image_raw"

   and publishes the 3 joint state topics:

   "joint_angle_2"

   "joint_angle_3"

   "joint_angle_4"

2. vision_2.py: solution for Section 2.2. This file should initiate a node that subscribes to image topics:

   "/camera1/robot/image_raw"

   "/camera2/robot/image_raw"

   and publishes the 3 joint state topics:

   "joint_angle_1"

   "joint_angle_3"

   "joint_angle_4"

3. control.py: solution for Section 3.2. This file should initiate a node that subscribes to the joint angles published by vision_2.py:

   "joint_angle_1"

   "joint_angle_3"

"joint_angle_4"

and publishes the topics that controls the robot joints:

"/robot/joint1_position_controller/command"

"/robot/joint3_position_controller/command"

"/robot/joint4_position_controller/command"

Make sure your code can run smoothly in the Ubuntu environment. For example, if your code is created on Windows, the file will raise errors because Windows and Unix use different line breaks. Errors like this will cause heavy point deductions.

Notice: The joint states you use in Section 3.1 and Section 3.2 can only be estimations from Section 2.2. You'll find a "/robot/joint_states" topic in your ROS topics. This topic contains true joint states from the simulation. You can use this topic to see how far off your joint state estimations are. But in no circumstances should you ever use this topic in your solution. Some other topics can serve the same purpose: for example, "/gazebo/link_states" and "/gazebo/model_states". Subscribing to these topics within your coursework submission codes will be considered cheating.

4.3 GitHub Repository

In your report, give us a link to your GitHub repository for the final solution. Your repository should be public. We should be able to download your library and run your executable files and get the results of Section 2 and Section 3 of the project without errors.

Your GitHub account should clearly show the history of development of your code. Do not copy and paste the final library in GitHub. Gradually update it (commit and push) during the course of four weeks as you are writing your code. Contribution disputes without a clear GitHub commit history will be ignored.