ECE276C Assignment 3: Visual Servoing

General Scenario: A robot manipulator, the Franka Emika Panda 7 degree of freedom arm has a camera attached to its end effector. An object is in its vicinity that it wishes to track in the center of the camera. Find the controller that will keep the eye-in-hand robot's camera centered on the object. The computer vision code is given to you so that it will know the pixel location of the object (a flat red block).

An image is generated at each timestep. This image comes in pairs, one is a 3-channel RGB image of size 512x512 pixels, and the other is a single channel depth image that measures a positive distance from camera's origin to the environment, in units of meters, and at 512x512 resolution. The alignment means that depth[u,v] and RGB[u,v] refer to the same location. Note that cartesian space is also measured in units of meters.

Scenario 1: Without Robot (i.e., assume a floating camera)

Q1. Derive the Image Jacobian analytically.

The robot end-effector (EE) frame points with z axis out from its gripper, and x-axis in a reasonable "right direction" way. This follows DH convention. The camera, for OpenGL, is required to be pointed towards the user, i.e., z for the camera's coordinate frame points into the gripper, and camera x aligns with end-effector x. You can verify this using the `draw_coordinate_frame` function.

Given a signed distance u_{NDC}, v_{NDC} [pixels] measured form the center of an image describing the location of the object, and a measurement β describing the positive distance of the object to the camera origin, as measured by a depth sensor. Derive the 2×6 image Jacobian for the current coordinate frame setup, using the following equation to start (w:world, c:camera, loc: cartesian position).

$$^{w}p_{loc} = {}^{w}p_{c} + {}^{w}_{c}R\ {}^{c}p_{loc}$$

The image Jacobian should be defined as the mapping from Cartesian world frame to pixel space, i.e., ${}^{w}[\dot{x}_c, \dot{y}_c, \dot{c}_c, \omega_{c,x}, \omega_{c,y}, \omega_{c,z}] \rightarrow [\dot{u}_{NDC}, \dot{v}_{NDC}]$.

Show an independent sequence of +ve $\dot{x}_c$, +ve $\dot{y}_c$, … $\omega_{c,z}$ so that you are moving the camera in only one axis, the displacement of the object moves correctly. You should reset the camera pose each time to prevent the object from leaving the screen. Save the sequence as an animation. A suggestion would be to construct the Jacobian 1 column at a time and verify that $\dot{u}$ and $\dot{v}$ makes sense when given a delta on that axis, before moving on.

Produce 1 gif of translation and angular displacement of camera

Q2. Given the position of an object found in pixel coordinates (u,v), and a spatially aligned depth channel where depth(u,v) is the positive distance measured from the camera's origin to the scene, find a controller on a floating camera

***Note:** Images in OpenCV start with x=0 and y=0 from the top left corner, and y is increasing as one scans *downwards* on an image. Pinhole camera models assume that (x = 0,y = 0) is at the center of the image, with x increasing the right direction, and y increasing in the *upward* direction. Therefore, you need to make a conversion before you use them in the image Jacobian.

- show that you can center object with only translation.
  Hint: set your image Jacobian last 3 columns to zero.

- Show that you can center an object in the image with only rotation.
  Hint: set your image Jacobian first 3 columns to zero.
- Show that you can center an object with both translation and orientation.

(Recommended settings: $K_\omega = 0.1, K_\omega = 0.02$)

Produce 3 gifs of camera motion tracking stationary object

Scenario 2: With Camera Attached to the Robot

Q3. Show that you can center an object with both translation and orientation when the camera is placed on the end effector of a robot arm, without robot singularities.

a) Track the object when it is stationary, i.e.,
`object_center = [0.3, 1, 0.01].`

b) Track the object when it is moving, i.e.,
```
object_center [0.5+0.2*np.sin(np.pi/2+ITER/10),
0.5+0.2*np.sin(ITER/20), 0.01]
```

Uncomment the following definition to reset the box position in the simulation loop at very timestep:
```
p.resetBasePositionAndOrientation(boxId,object_center,p.getQuaternionF
romEuler(object_orientation))
```

Produce 2 gifs of camera motion tracking stationary object

c) **Add in a term to make the roll of the camera as close to zero as possible w.r.t. the ground plane. (i.e., keep the camera level and only allow pitch and yaw).** Hint, define a metric to minimize, take the gradient w.r.t. joint angles and apply that in a way that does not affect the primary tracking objective.

Produce 1 gif of camera motion tracking stationary object

Deliverable: Besides GIFs, upload a modified Jupyter Notebook that can run to verify the solutions. As long as the code is complete and it is clear which line-items to (un)comment to achieve different outputs of Q1-Q4, you do not have to give us separate files and it can be 1 script.

Notes: Because PyBullet uses OpenGl renderer and has a more computer graphics flavor, we have provided helper functions and a robot class to wrap around pybullet functions and provide the same functionality but in the correct format / frames of reference. Here are some examples (function names should be self-explanatory, but its probably still worth reading the code). Note that anything with "_" in front of the function or variable is a private member variable/function and you should avoid calling these functions (they may not do what you expect. What you should call as they are written to make life easier for you is:

- `get_camera_img_float` #gets the image and depth for a camera, given its position and orientation matrix in world frame
- `get_jacobian_at_current_position` #get the robot Jacobian
- `get_ee_position` #get the end effector position and orientation
- `get_current_joint_angles` #gets the current joint angle of the robot
- `draw_coordinate_frame` #draws a coordinate frame given a position and orientation matrix in the world coordinate frame (red = x, green = y, blue = z)

Having trouble? Hints:

Evaluate one column of the Jacobian at a time. Zero out the other axes of motion. Apply a motion in one direction. See if the your camera or robot moves as you would expect

When debugging, set the floating camera looking down with x-axes aligned with the world axes. This should help you debug. Then, when movement seems reasonable, set the camera at an angle; does movement still seem reasonable?

Camera frame x aligns with image x, camera frame z points towards user. Depth values should be made negated if presenting them in the camera's frame of reference.