84/100

Robotics Assignment 3.

Anupama Rajkumar(415212), Aditya Mohan (415276), David Rozenberszki (41532)

November 2019

A Control Theory

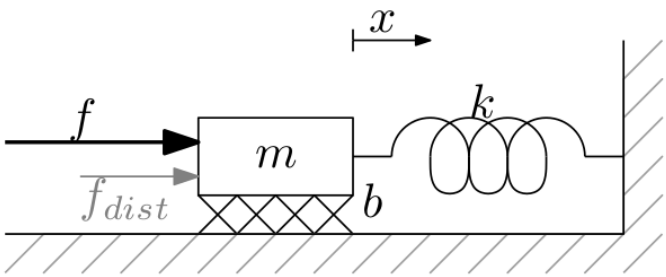

Figure 1: Damped spring-mass system with an actuator

a) Calculation of Natural Frequency and Natural Damping Ratio

Given: m = 16, k = 4, b = 16, f = 0

$$16\ddot{x} + 16\dot{x} + 4x = f$$

$$\zeta = \frac{b}{2\sqrt{km}}$$

$$\zeta = 1$$

5/5

$$\omega_n = \sqrt{\frac{k}{m}}$$

$$\omega_n = 0.5m/s$$

Since the natural damping frequency is 1, the system is critically damped

b) Design of a PD Controller for critically damped system

$$m\ddot{x} + b\dot{x} + kx = -k_p x - k_v \dot{x}$$

Given:

$$x_d = 0, m = 16, k = 4, b = 16, k_{CLS} = 16$$

Substituting the values,

$$16\ddot{x} + 16\dot{x} + 4x = -k_p x - k_v \dot{x}$$

$$16\ddot{x} + (16 + k_v)\dot{x} + (4 + k_p)x = 0$$

$$k^{'} = k_{CLS} = 4 + k_p = 16$$

Hence,

$$k_p = 12$$

For critically damped system,

$$b^{'} = 2\sqrt{mk^{'}} = b + k_v$$

Hence,

$$k_v = 16$$

$$f = -12x - 16\dot{x}$$

5/5

c) Non linear controller

$$16\ddot{x} + 30.sgn(\dot{x}) + 4x = f$$

$$f = \alpha f^{'} + \beta$$

$$f^{'} = \ddot{x_d} - k_v^{'}(\dot{x} - \dot{x_d}) - k_p^{'}(x - x_d)$$

$$f^{'} = \ddot{x}$$

$$(\ddot{x} - \ddot{x_d}) + k_v^{'}(\dot{x} - \dot{x_d}) + k_p^{'}(x - x_d) = 0$$

$$\ddot{e} + k_v^{'}\dot{e} + k_p^{'}e = 0$$

$$\alpha = 16$$

$$\beta = 30.sgn(\dot{x}) + 4x$$

$$k_p^{'} = k_{CLS} = 16$$

$$k_v^{'} = 2\sqrt{k_p}$$

$$k_v^{'} = 8$$

$$f^{'} = \ddot{x_d} - 8(\dot{x} - \dot{x_d}) - 16(x - x_d)$$

1/5

kp' is actually not kCLS. (unit mass syste

d) Calculate steady state error

$$e = x_d - x$$

$$f^{'} = \ddot{x_d} + k_v^{'}\dot{e} + k_p^{'}e$$

$$\ddot{x} = \ddot{x_d} + k_v^{'}\dot{e} + k_p^{'}e$$

$$\ddot{e} + k_v^{'}\dot{e} + k_p^{'}e = 0$$

Disturbance force,

$$f_{dist} = 8$$

So,

$$\ddot{e} + k_v^{'}\dot{e} + k_p^{'}e = f_{dist}$$

Given,

$$\ddot{e} = \dot{e} = 0$$

So,

$$f_{dist} = (k_p)e$$

This is the steady state equation and e is the steady state error

$$k_p^{'} = 16$$

So,

$$e = 0.5$$

5/5

B Visual Servoing

1. Image Features [15 Points]

There are several methods to find describing features on images. Most methods leverage of the high output values of different filters on the original images. It is easy to detect an unique feature on an image, where we know there is only one object of that type, but challenging to create the same result on diverse and general images. Also to detect the orientation and position of the image we need a minimum number of 3 different feature to create an invertable image Jacobian.

1) The relatively most basic solution to create features to run a *F* corner detector on the image. This solution is fast, uses the local ar one and keep that as a candidate to track ongoing for the later imag

2) The *SIFT* (scale-invariant feature transform) might be the mo used in computer vision. For every key point the algorithm creates a candidate pixel and store these values in the memory. For the consec vector for those pixels a well, and compare the Eucledian distance search to increase performance. As we only need a minimum number can only choose the 3 best matches with the highest confidence (lowes matches to increase robustness to noise.

3) The *ORB* (Oriented FAST and rotated BRIEF) detector is is si but with generalized image orientation changes and faster performa centroid of every patch on the image and tor from its co the orientation of the patch. Then creates ble of the patc as well to increase comparability with the image.

It is evident that you are quite familiar with con

B1: 3/15

2. Find Circle [15 Points]

(a) findCircleFeature() : In this function the cv::HoughCircles() image. This function takes-in the grayscale image and the related parameters like the Hough Gradient and the matrix window as input, detects all the circles present in the image frame and stores them in a Vec3f array. This array is then parsed to detect the circle with the largest diameter, and this is the circle that needs to be tracked, which is then assigned to the circle structure defined in the program. For observability, we also use the imshow() function to display the detected circle.

(b) transformFromOpenCVFToFF() : In this function, the co-ordinates are translated from OpenCV image frame into the feature frame. Since the origin of the Image frame is at the top left corner of the Image, the coordinates are translated as :

$$z_F = z_I$$
$$x_F = x_I - 0.5 * Width$$
$$y_F = y_T - 0.5 * Height$$

The variables in the feature frame and Image frame are expressed with subscripts F and I, respectively.

B2: 10 + 5 = 15/15

3. Image Jacobian computation [30 Points]

(a) estimateCircleDepth() : In this function the depth of the circle is calculated using the pinhole camera model.

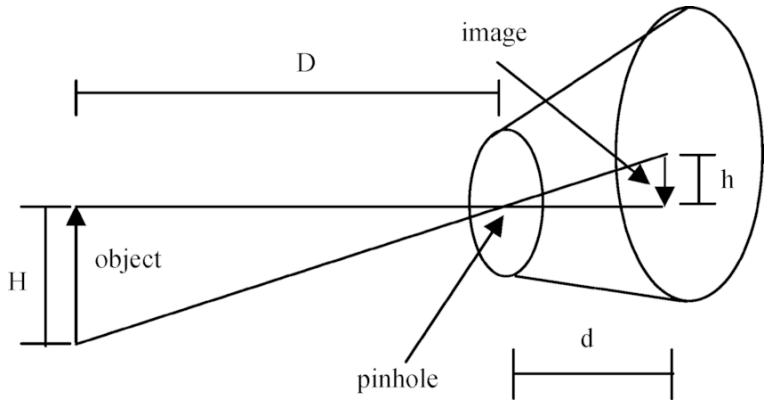

Figure 2: Pinhole Camera Model for Depth estimation

In Figure 2, the depths are represented by D and d and the heights, which in this case are the Radii of the circles, are represented by H and h. The angle made by the image and the object are vertically opposite and so, would have the same values for the tangents. Thus,

$$H/D = h/d$$

Two points need to be taken into consideration here:
(i) The image is formed on the focus of the circle, and so the depth of the image will equal the focal value of the lens. Thus,

$$d = f$$

(ii) The Ratio of the Radii is same as the Ratio of the diameters, which is easily extracted from the system as they are stored accordingly. Thus, if D is the diameter of a circle, then:

$$H/h = D_{object}/D_{image}$$

The depth can now be calculated as:

$$depth = f * (D_{object}/D_{image})$$

Here D(object) is passed as a parameter along with f and a structure that contains the circle image. Thus, all the variables on the RHS of the above equations are obtainable from the function parameters.

(b) Experiment to Determine Focal Length
This experiment can be performed in a similar manner as A, using Figure 2 as a reference. The steps are as follows:

1 Using a ruler, place the the object at a calculated short distance from the camera.

2 Note down the distance and the diameter of the circle.

3 Calculate the size of the actual image from opencv, since the image on the screen would be a magnified version.

4 Determine the focal length by equation the ratios of size and depth of the objects.

An alternative way to compute the size of the image without using opencv would be to realize that the number of pixels on the screen are constant. Hence, if the size of the aperture of the camera are known, and the dimensions of the screen of projections are known, then a magnifying factor can be calculated that essentially gives the same answer in roudabout. However, these parameters are already detected by opencv and hence, just taking the size input circuvents the problem of getting the dimensions.
(c) Calculating the Image Jacobian
Let (x,y,R) be the dimensions of the image and (S_u, S_v, r) be the dimensions of the image. Let f be the focal length

of the image, and z be the depth of the image. Using the pinhole model in figure 2 in a 3D space along all basis vectors, the following relations can eb obtained:

$$S_u = x * f/z$$
$$S_v = y * f/z$$
$$r = R * f/z$$

As we move the image, only x and y change but R always remains constant. Thus, the derivatives of the image variables can be calculated as:

$$\dot{S_u} = (\dot{x}z - x\dot{z}) * f/z^2$$
$$\dot{S_v} = (\dot{y}z - y\dot{z}) * f/z^2$$
$$\dot{r} = -\dot{z} * f/z^2$$

Using the original relationships between the image and object variables, we can write the derivatives in the following form:

$$\dot{S_u} = \dot{x} * f/z - \dot{z} * S_u/z$$
$$\dot{S_v} = \dot{y} * f/z - \dot{z} * S_v/z$$
$$\dot{r} = -\dot{z} * r/z$$

Thus the jacobian can be written as:

$$\begin{bmatrix} f/z & 0 & -S_u/z \\ 0 & f/z & -S_v/z \\ 0 & 0 & -r/z \end{bmatrix}$$

However, in our experiment the camera follows the object, which means the co-ordinates need to be reversed in sign. Hence, the jacobian to be used comes out to be:

$$J_{Im} = \begin{bmatrix} -f/z & 0 & S_u/z \\ 0 & -f/z & S_v/z \\ 0 & 0 & r/z \end{bmatrix}$$

Also as the known input for the Jacobian calculation is not the measured, but the original diameter of the real circle we use and implement this version replacing r by D (we can do this as the output of the Hough-transform is also D and not R):

$$J_{Im} = \begin{bmatrix} -f/z & 0 & S_u/z \\ 0 & -f/z & S_v/z \\ 0 & 0 & D * f/z^2 \end{bmatrix}$$

B3: 30/30

4. Velocity vector transformation [10 Points]

Es detailed in the task to transform the calculated velocities from the camera-frame to the base-frame we need to perform the following steps consecutively.

1. Calculate the rigid transformation between the camera and end effector orientation.
2. Calculate the transformation from the base-frame to the end effector-frame and apply it to the velocity vector in every control loop.

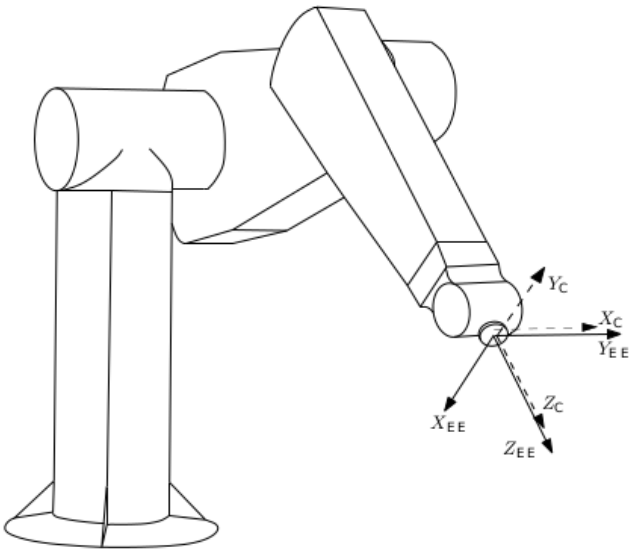

Figure 3: Camera and End effector frames relative orientation

For the first transformation as it can be seen on Figure 3, we need to rotate the camera frame to the end-effector frame by 90°. From that we can either calculate the rotation matrix – or as it is a simple rotation and to increase performance – we can directly map the corresponding velocity axes together by applying:

- $X_{EE} = -Y_C$
- $Y_{EE} = X_C$
- $Z_{EE} = Z_C$

Then to finally calculate the required velocity in the base-frame we need to understand the current pose representation given by the vector x.
It is a 7 dimension PrVector, where the first 3 elements containing the position of the end effector, while the last 4 elements the orientation in quaternion form. Here we have to opportunities. either calculate the representing the rotation matrix to multiply directly the velocity vectors by the formula given by to the code attached paper or use the implemented function of the PrQuaternion class. We used the later solution as it seemed to be the easiest, yet the best working solution.
So first we saved the current quaternion, had it as a rotation matrix representation and multiplied the end effector velocity vector with it.

5. Workspace Limitation [10 Points]

To test the reachable radius around the base frame we simply needed to check the first 3 elements of the state vector and calculate the absolute Eucledian-distance value around the base frame. If he desired pose after the current loops iteration would extend the allowed reachable radius we reduce the latest step from the desired pose, this way commanding the arm to stand still, still within the allowed radius.

Student Name	A-a	A-b	A-c	A-d	B-1	B-2	B-3	B-4	B-5
Anupama	33.34	33.33	33.33	33.34	33.33	33.33	33.34	33.33	33.33
Aditya	33.33	33.34	33.33	33.33	33.34	33.33	33.33	33.34	33.33
David	33.33	33.33	33.34	33.33	33.33	33.34	33.33	33.33	33.34

B4: 10/10B5: 10/10