# TU Berlin Robotics WS2019/2020

Version: 018/11/2019, 20:14

# Lab Assignment #3

Please upload your solution by **December 9th, 2019 at 23:59**, using your ISIS account. Remember that this is a hard deadline, extensions are impossible!

## A Control Theory

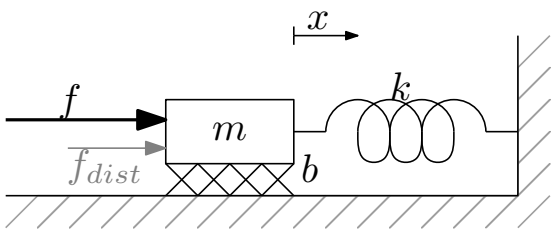


Figure 1: Damped spring-mass system with an actuator.

1. Consider a 1-DOF spring-mass system with friction (Fig. 1). We assume that the mass is $m = 16$, the spring constant equals $k = 4$, and the friction coefficient is given by $b = 16$. Thus, we can describe the system by the equation of motion: $16\ddot{x} + 16\dot{x} + 4x = f$.

   (a) [5 Points] Find the natural frequency $\omega_n$ and the natural damping ratio $\zeta_n$ of the natural system, i.e., $f = 0$. What kind of system is this (oscillatory, overdamped,...)?

   (b) [5 Points] Design a PD controller $f$ that realizes a critically damped system with a closed-loop stiffness of $k_{CLS} = 16$. Assume that the desired position is $x_d = 0$.

   (c) [5 Points] Assume that the friction changes from linear friction ($b = 16\dot{x}$) to Coulomb friction: $b = 30 \cdot sign(\dot{x})$. Design a controller that uses a non-linear model-based portion with trajectory following to critically damp the system at all times and maintain a closed-loop stiffness of $k_{CLS} = 16$. In other words, let $f = \alpha f' + \beta$ and $f' = \ddot{x_d} - k'_v(\dot{x} - \dot{x_d}) - k'_p(x - x_d)$. Note that $f$ is an m-mass control, while $f'$ is a unit-mass control. Use the definition of error $e = x - x_d$. (Hint: Closed-loop stiffness refers only to the servo part).

   (d) [5 Points] What is the steady-state error $e = x - x_d$ of the system in part (c) (i.e. when $\ddot{e} = \dot{e} = 0$) if it is disturbed by a constant force $f_{dist} = 8$?

# B Visual Servoing

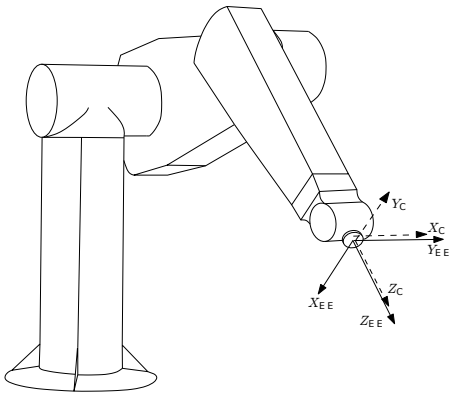


Figure 2: End-effector frame $(X_{EE}, Y_{EE}, Z_{EE})$ and camera frame $(X_c, Y_c, Z_c)$ of the 6-DOF Puma

The task in visual servoing is to use visual information to control the pose of the robot's end-effector relative to a target object (*position-based* VS) or a set of target features (*image-based* VS). We also distinguish between systems that only observe the target (*endpoint open-loop*, *EOL*) and those which observe both target and end-effector (*endpoint closed-loop*, *ECL*).
We have mounted a webcam at the end-effector of our Puma robot (*eye-in-hand*) to implement an image-based EOL approach to Visual Servoing. Your goal is to keep a certain distance between end effector and tracked object, and a fixed viewing direction.

We only control the end effector position relative to the tracked object and keep the end effector orientation fixed. Once we have decided which features we need to extract from the image, we'll take advantage of a concept you are already familiar with: the Jacobian. We compute the *Image Jacobian* $J_I$ that (in our case) maps desired velocity of the end effector in the camera frame to to a corresponding feature velocity in image feature space:

$$\dot{s} = J_I \dot{p}_c$$

where $\dot{s}$ is the velocity of features in feature space (feature frame) and $\dot{p}_c$ the camera velocity in the camera frame. Using the Image Jacobian's pseudoinverse we can first compute a desired camera velocity and from that the desired end effector velocity in the base frame. The latter is then resolved to a desired position for the next vision loop timestep and sent to an operational space position controller.

1. [15 Points] **Image Features:** Propose *three* possible image features $s$ that we could use to accomplish visual servoing to control both position and orientation in operational space. Describe and illustrate their pros and cons with respect to the following questions:

   - How easy/difficult is it to extract the feature?
   - How can you parametrize the feature? What is the dimensionality of the feature (i.e., are the parameters independent)? What is the dimensionality of a minimal parametrization?
   - How many DoFs of the operational space task can you control with the feature?
   - Which dimensionality will the Image Jacobian have given your feature set and the number of DoFs we want to control?
   - If we detect this feature, is the resulting system of equations $\dot{s} = J_I \dot{p}_c$ well-formed, overconstrained or underconstrained?

2. **Find Circle:** Let's use a circle $f = (u, v, d)$ as our feature to track. It is described by the coordinates $u$ and $v$ of its centroid and its diameter $d$ in the image plane.
   Note: The visual servoing code template offers a compile-time switch to change between using the simulated camera and a real camera as image source (`#undef USE_SIMULATED_CAMERA` in cv_control.h)

(a) [10 Points] Use the Hough transform to find the circle feature in the preprocessed image (`backproject`). Implement the feature extraction inside the method `findCircleFeature(...)`. Make sure your implementation is tolerant to adverse camera images and document the steps you did to ensure this
Hint: The OpenCV library already provides an implementation[1].

(b) [5 Points] Implement the transformation of the features from the OpenCV image frame into the feature frame in `transformFromOpenCVFToFF. (Hint: only translation!)`

3. **Image Jacobian computation:** In the following you will specify and implement the Image Jacobian $J_I$.

   (a) [10 Points] We need to know the depth distance $z$ of the circle to compute our Image Jacobian. Derive the formula for calculating $z$ and implement it in `estimateCircleDepth(...)`. Include a short explanation of the derivation in your report.

   (b) [10 Points] In addition to the circle centroid's position, the Image Jacobian also depends on the focal length of the camera. Describe a way to determine this parameter experimentally.

   (c) [10 Points] Specify the image Jacobian $J_I \in \mathbb{R}^{3\times3}$ and implement it in the method `getImageJacobianCFToFF(...)`. Describe the equation for each element of the Jacobian in your written report.

4. **Velocity vector transformation:** Note that the desired velocities you get from your inverse image Jacobian is the camera velocity in camera frame but you can only command end effector position in the robot base frame. You have to transform a velocity in the camera frame to a velocity in the base frame.

   (a) [10 Points] Implement the two functions `transformVelocityFromCFToEEF(...)` and `transformVelocityFromEEFToBF(...)` that transform a velocity from Camera Frame to End Effector Frame, and from End Effector Frame to Base Frame respectively.
   Note: You can assume a static transformation between the camera and the end effector because they are rigidly attached (Fig. ). However, you should not assume the transformation between the end effector and the base to be constant!

5. **Workspace Limitation**:

   (a) [10 Points] Limit the commanded end effector position to stay within the robot's work space in `controlRobot`(...) to avoid singular configurations. Assume a sphere of $0.85m$ radius around the base frame as the robot's reachable workspace.

Your solution must contain the following files:

- Your **commented** vision loop **code**: `cv_control.cpp` (from `cv/`)
  - Do not add, modify or upload any other source code files.
- The 6-DOF mode **gains**: `gains_3.txt (from your current working directory)`
  - The file will only be created when you click on "Store gains" in the GUI.
  - Important: You must **not** hard-code the gains inside `control.cpp`; always use the appropriate global variables (`gv.kp` etc.)
- **one** pdf file containing:
  - the solutions for the **Control Theory** part
  - Text answers of the **Visual Servoing** part.
  - **Figures**. All plots must have a legend and all axes must be labeled.
  - **For all tasks, state the specific subtasks each team member was working on and list how much each team member contributed (in percent). The contributions should add up to 100%.**

[1] http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_circle/hough_circle.html