TTK4255 Robotic Vision, Spring 2020

Exercise 0

Python/Matlab and Linear Algebra Review

The exercises in this course are a mix of programming problems and theory questions. You are free to use any programming language, but solutions will be provided in Python and Matlab, and we may not be able to help you effectively if you use a different language. We therefore recommend that you use either Python or Matlab.

This first assignment is aimed to get you acquainted with Python/Matlab features and libraries we will use throughout the course. **It is optional and there is no Blackboard submission**. You can also find the solution already posted on Piazza. Use it if you get stuck on programming details.

If you need help getting started or have questions regarding the exercise:

- Post a question on Piazza (you can post anonymously if you want)
- Ask for help during the weekly tutorial sessions (Wed. 16:15-18:00 in EL2)

If you don't have a personal computer, you can use the public computers at Gløshaugen which should have Matlab and Python installed. We haven't reserved any computer lab, so it's up to you to find available public computers if necessary.

Getting started: Python

While we don't require a specific version, the solutions are written in Python 3.7. If you want to be able to run them without significant modifications, use a version from 3.0 or above. Throughout the course we'll use linear algebra, including matrix and vector arithmetic, singular value decomposition and solving systems of linear equations. we'll also load and process images, and make lots of visualizations. To help with this, we recommend you install Numpy[1] and Matplotlib[2].

The Python Numpy Tutorial[3] by Justin Johnson is a good reference for Python, Numpy and Matplotlib functions we'll use. The last page of this exercise also has a list of useful functions and additional tips.

Getting started: Matlab

The solutions make use of local function definitions inside script files—a feature added in 2016b. If you have an earlier version, you may want to upgrade. We'll also use some functions in the Image Processing Toolbox[4]. If you don't have it, you can install new toolboxes to an existing Matlab installation by rerunning the installer[5].

The first eight pages of the Matlab Primer[6] from ETHZ is a good reference for Matlab functionality we'll use. The last page of this exercise also has a list of useful functions and additional tips.

1 Image processing

This part will familiarize you with loading, plotting and doing arithmetic on images using your chosen programming language.

(a) Extract the file `roomba.jpg` into your working directory. The image shows two Roomba vacuum cleaners with colored plates attached to them. The one in the center has a red plate, and the other has a green plate. Load the image and plot it. Print the image dimensions (width and height).

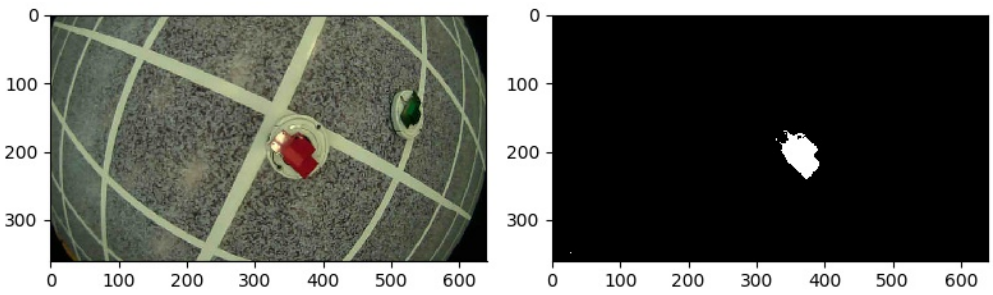
(b) The image contains three *channels* (red, green and blue). If you are using Python, a single-channel image can be extracted using `img[:,:,i]`, $i \in 0, 1, 2$. If you are using Matlab, the syntax is `img(:,:,i)`, $i \in 1, 2, 3$. Can you find out which i corresponds to the *red* channel by plotting the three single-channel images?

(c) *Thresholding* is commonly used in computer vision to isolate parts of interest in an image (a problem known as *segmentation*), and is easily implemented in Python and Matlab using the syntax `img > threshold`. The result is a binary image of 1's and 0's, where a value of 1 indicates that the original pixel value was above the given threshold. Based on the red channel image, can you isolate only the red parts of the image by using an appropriate threshold?

(d) The answer to the previous question may not be what you were expecting. You will learn about color spaces later in the course, but a rough explanation is that the R component of RGB mixes the brightness (how light the color is) and saturation (how strong the color is), and therefore cannot be used in isolation to determine the "red-ness" of something. For example, a bright white pixel (R=G=B=1) has a high R component, but doesn't look red. You will also learn about better segmentation methods later in the course, but for now you can try this simple strategy:

1. Choose an appropriate reference RGB color, $c_{\text{ref}} = (R_{\text{ref}}, G_{\text{ref}}, B_{\text{ref}})$.
2. For each pixel $c = (R, G, B)$ in the original image, compute the Euclidean distance $||c - c_{\text{ref}}||_2$ to obtain a single-channel distance image.
3. Binarize the distance image using an appropriate threshold.

Your output should look something like this:



2 Linear algebra

This part will refresh you on *rigid-body transformation* and make you comfortable with doing matrix- and vector arithmetic in your programming language of choice. Below are the relevant chapters from the course syllabus (see Piazza to access course material). To keep things simple, we'll stick to two dimensions in this exercise. You can therefore prioritize sections about two-dimensional transformations.

- *Robotics, Vision and Control.* Chapter 2.1.
- *An Invitation to 3D Vision.* Chapter 2.

Recall that a matrix is a transformation between coordinate frames. Given a point with coordinates $\mathbf{x}^b = (x^b, y^b)^T$ in frame b, its coordinates in frame a are

$$\mathbf{x}^a = \mathbf{R}_b^a \mathbf{x}^b \tag{1}$$

In general, $\mathbf{R}_b^a$ can be any 2×2 matrix, but a rigid-body transformation must preserve the size and shape of objects (hence the name *rigid*), so $\mathbf{R}_b^a$ must be a rotation or reflection.

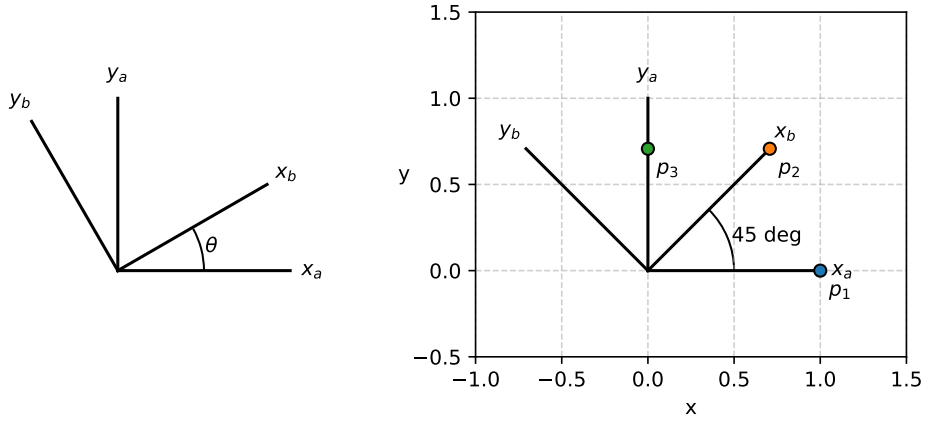


Figure 1: Left: A counter-clockwise rotation. Right: A set of points.

(a) Find an expression for the *counter-clockwise* rotation matrix $\mathbf{R}_b^a(\theta)$ shown in Fig. 1.

(b) Consider the points shown in Fig. 1 (right) with coordinates:

$$\mathbf{p}_1^a = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \mathbf{p}_2^b = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \mathbf{p}_3^b = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix} \tag{2}$$

By inspecting the figure, find rough values for the corresponding coordinates:

$$\mathbf{p}_1^b = \begin{pmatrix} ? \\ ? \end{pmatrix} \quad \mathbf{p}_2^a = \begin{pmatrix} ? \\ ? \end{pmatrix} \quad \mathbf{p}_3^a = \begin{pmatrix} ? \\ ? \end{pmatrix} \tag{3}$$

(c) Write a program to compute the above coordinates using matrix multiplication. Verify that the results agree with your visual estimates in (b).

If the coordinate frames are seperated by a translation, equation (1) is modified to:

$$\mathbf{x}^a = \mathbf{R}_b^a \mathbf{x}^b + \mathbf{t}_{b/a}^a \tag{4}$$

The subscript b/a (pronounced "b relative a") means that $\mathbf{t}$ is a vector going *to* b *from* a, and the superscript means that its coordinates are given in frame a. This can be written more simply using *homogeneous* coordinates:

$$\begin{pmatrix} \mathbf{x}^a \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{R}_b^a & \mathbf{t}_{b/a}^a \\ \mathbf{0} & 1 \end{pmatrix} \begin{pmatrix} \mathbf{x}^b \\ 1 \end{pmatrix} = \mathbf{T}_b^a \begin{pmatrix} \mathbf{x}^b \\ 1 \end{pmatrix} \tag{5}$$

where $\mathbf{T}_a^b$ is called a *homogeneous rigid-body transformation.*

(d) In a program, define the functions `rotate(angle)` and `translate(x,y)` that return a homogeneous rotation and translation matrix, respectively. Define also a function `draw_frame` that draws the axes of a rigid-body transformation.

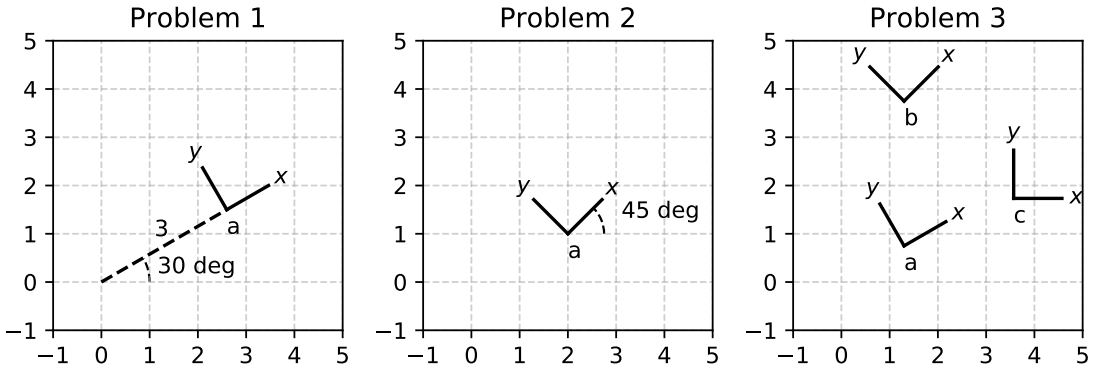


Figure 2: Rigid-body problems

(e) Consider Problem 1 shown in Fig. 2, where a frame a is rotated by 30 degrees, then translated along its new x-axis by 3 units. Find an expression for the transform relating frame a to the plot axes.

Do this with pen-and-paper first. Once you are 90% confident that your answer is correct, verify it by writing a program that defines the same transformation and plots it.

(f) Do the same for Problem 2, where frame a is rotated by 45 degrees, then translated along the *plot* axes by $(2, 1)$ units.

(g) Problem 3 is more difficult. Here:

- Frame a is rotated by 30 degrees about the origin, then translated along its x-axis by 1.5 units.
- Frame b rotates a 15 degrees about its origin, then translates along the plot y-axis by 3 units.
- Frame c rotates b -45 degrees about the plot origin.

Find an expression for the transform relating each frame to the plot axes.

Python tips

`import matplotlib.pyplot as plt`	Import a Matlab-like interface to Matplotlib.
`import numpy as np`	Common way to import Numpy.
`from numpy import *`	Make all functions accessible in global namespace.
`help(plt.plot)`	Print information about the `plot` function.
`np.array([[x],[y],[z]])`	Create a 3D column-vector.
`np.linalg.norm(p)`	Calculate $\sqrt{x^2+y^2+z^2}$ if $p=(x,y,z)$.
`h,w = img.shape[0:2]`	Retrieve the height and width of an image.
`plt.imread`	Load an image.
`plt.imshow`	Draw an image on the current plot.
`plt.imsave`	Save an image to disk.
`plt.show`	Make your plots actually show up on the screen.
`plt.savefig('figure.png')`	Save displayed figure to disk.
`plt.plot([x0,x1], [y0,y1])`	Draw a single line from (x_0,y_0) to (x_1,y_1).
`plt.plot(p[:,0], p[:,1])`	Draw lines connecting points stored as an $n\times 2$ array.
`plt.scatter(x, y)`	Draw a marker at (x,y).
`plt.scatter(p[:,0], p[:,1])`	Draw a set of points stored as an $n\times 2$ array.

Unlike Matlab, the $*$ operator in Numpy is element-wise multiplication, not matrix multiplication. The `dot` function or @ operator can be used instead to compute inner products of vectors, to multiply a vector by a matrix, and to multiply matrices: e.g. `A.dot(b)` and `A @ b` both perform standard matrix multiplication between A and b. Alternatively, you can use the `np.matrix` type, where $*$ does act as matrix multiplication.

Matlab tips

`help <function>`	Print information about a function.
`lookfor <keyword>`	Get a list of functions related to a keyword.
`doc <function>`	Documentation for a function.
`doc images`	Documentation for the Image Processing Toolbox.
`imread`	Load an image.
`imshow`	Draw an image on the current plot.
`imwrite`	Save an image to disk.
`print('myfigure', '-dpng')`	Save displayed figure to disk.
`plot(x,y)`	Draw lines.
`scatter(x,y)`	Draw points.
`norm(v)`	Calculate the Euclidean 2-norm of vector v.

`imread` stores images using 8-bit unsigned integers (values 0 to 255 in each channel). This can lead to weird outputs or mysterious error messages due to overflow or type mismatch. It can be useful to convert the image to floating-point using `im2double`.

Recent versions of Matlab come with Live Scripts, which lets you add numeric sliders and other controls to your code. This is useful for tweaking values (such as thresholds or pose parameters) and seeing your plots update live.

Notes

[1]https://pypi.org/project/numpy/

[2]https://pypi.org/project/matplotlib/

[3]http://cs231n.github.io/python-numpy-tutorial/

[4]https://www.mathworks.com/products/image.html

[5]https://se.mathworks.com/matlabcentral/answers/101885-how-do-i-install-additional-toolboxes-into-an-existing-installation-of-matlab

[6]http://rpg.ifi.uzh.ch/docs/teaching/2019/MatlabPrimer.pdf