TUTORIAL 5 – MACHINE VISION FOR MECHATRONICS

EQUIPMENT NEEDED

Your Laptop
ChArUco calibration board (needs to be on paper)
ArUco markers (can be paper or on phone screen)

THE OPENCV LIBRARY

SIMPLE IMAGE PROCESSING

First, we are going to get familiar with the OpenCV system and its use in Python. You will need to download from Moodle the template *CV_Python_template*. This template will demonstrate some basic steps for image processing, i.e. connect to a camera, take a frame, process a frame (convert to grey) and show the result. We are going now to alter it to perform Edge detection.

1. Based on the code *CV_Python_template* try to implement a Canny Edge detection.
   The function to use is
   `canny = cv2.Canny(gray,100,200)`
   and should go after the original gray scale conversion. The values after the image are defining the threshold for the Edge detection
2. Create an extra window to show the canny outcome (see lines 17-23 from *CV_Python_template* on how to do this*)*
3. Experiment with the threshold values to achieve a desired effect.
4. Use the Canny Edge on the original image and compare the results.

You might have some small artefacts on the reslts . Maybe you can remove them using blurring.

5. Implement a Gaussian Blur with a kernel of 5x5 and symmetrical kernel distribution with the command
   `cv2.GaussianBlur(gray,(5,5),0)`
6. Run the Edge detection on the blurred image and compare results with the others.

ARUCO

The next step is to implement the ArUco library and detect markers, their pose, etc. To do so you need to download the calibration files *Sample_Calibration.npz* which is a dummy calibration file. It contains a camera and a distortion matrix generated by a random web-camera.

You should also print some Aruco markers from the ArUco 4x4 50 library (included in Moodle) or load images from that library to your phone.

Starting from the *CV_Python_template*:

1. Import the Aruco library as aruco using the command:
   ```
   import cv2.aruco as aruco
   ```

2. Read and store the calibration information from *Sample_Calibration*
   ```
   Camera=np.load('Sample_Calibration.npz') #Load the camera calibration values
   CM=Camera['CM'] #camera matrix
   dist_coef=Camera['dist_coef']# distortion coefficients from the camera
   ```

3. Load the ArUco Dictionary *Dictionary 4x4_50* and set the detection parameters
   ```
   aruco_dict = aruco.Dictionary_get(aruco.DICT_4X4_50)
   ```

```
pa = aruco.DetectorParameters_create()
```

4. After the convertion to gray (line 35 of the original) run the detection function
   `corners, ids, rP = aruco.detectMarkers(gray, aruco_dict)`

5. Draw the detected markers as an overlay on the original frame
   `out = aruco.drawDetectedMarkers(frame, corners, ids)`

6. Show the result to a window (you can use the *grey-image* if you want)

Now that you have the markers detected you want their information to use, i.e. their id and their pose.

7. Calculate the pose of the marker based on the Camera calibration using:
   `rvecs,tvecs,_objPoints = aruco.estimatePoseSingleMarkers(corners,70,CM,dist_coef)`

   Note: The number 70 is for the actual size of the ArUco marker in mm. This means if you have printed the marker you need to measure its size. If you use a marker on a screen be careful not to zoom-in or alter the size of the marker while running the code

8. Overlay the axis on the image

   This will fail if no marker is present, so we need to create a try-capture exception structure. This is a useful method to stop your code crashing (using a software interrupt)

```
try:
        out = aruco.drawAxis(out, CM, dist_coef, rvecs, tvecs, 10)
except:
        out = out
```

   or we need to check if the ids have been defined. This is a more hack-y (still valid) way to do the above.

```
if ids is not None:
        out = aruco.drawAxis(out, CM, dist_coef,rvecs, tvecs, 10)
```

9. Adapt the *print* command (line 47 of the original) or use the *logging* command seen in the communication tutorials to print the transformation matrix `tvecs` (you will need to write the code for this ☺, there are details you need to consider) and the `id` of the visible marker.

   Note: `tvecs` is a 1x1x3 matrix, i.e. x element is [0][0][0], y is [0][0][1], and z [0][0][2]

   Warning: **The tvecs numbers will be wrong! The calibration is a generic one <u>not the one for your camera.</u> The calibration process is explained later in the tutorial**

10. Experiment with exporting ArUco Ids and individual pose information (steps 8&9 above) for **multiple Aruco markers**.

    Notes:
    (1) You must use a structure that loops through all ids like:
    `for id in ids:`
    (2) The arrays `rvecs` and `tvecs` will now be three dimensional for each id so you might need to consider a loop command like:
    `for ind, id in enumerate(ids):`
    where you will get the id of the ArUco marker and the index

11. Using the information from the ArUco (ids or distance/orientation) send commands to the Arduino using WiFi (look at the communication tutorials) for performing a task (control motor, LED, etc.)

CALIBRATING A CAMERA

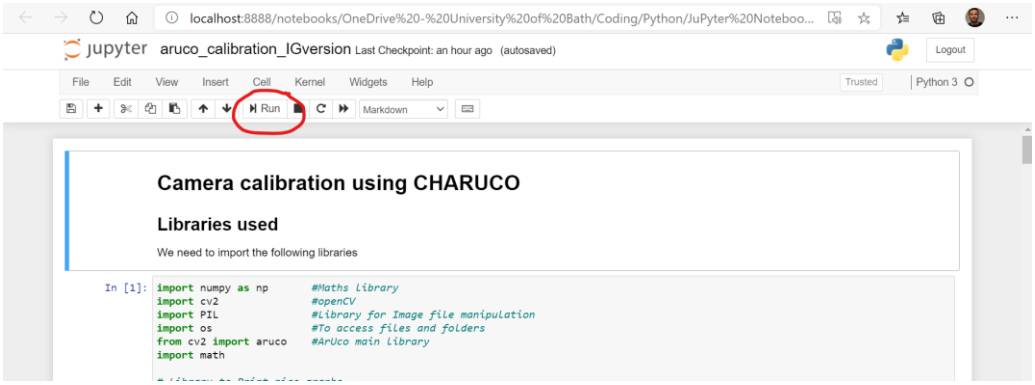
In order to be able to get meaningful information from the camera in terms of the position of the markers you will need to calibrate your camera and store this information to load everytime you run the ArUco library.

This process is described as a juPyter notebook titled *aruco_calibartion_IGversion* from Moodle.

To run a JuPyter notebook you can run the application from the Anaconda navigator or directly from the start menus. What will open will be a webpage similar to a File Exporel (Finder for Mac) in your browser. In effect this webpage has the conda Python environment enabled on the background.

Using this interface navigate where you have downloaded the above file and select it, starting a new webpage. A JuPyter notebook is very much like a live-script in Matlab and is used quite extensively when you are doing one-off processing, like the calibration we are about to do.

Follow the steps in the notebook and in order to run each cell (code or markdown) you must click on the run button.



12. After you have created in the Calibration.npz file use that to rerun the ArUco tests above.
13. Using a ruler or callipers check how precise the values reported by tvecs are.

(OPTIONAL) OTHER STUFF TO DO WITH OPENCV

These two examples might be interesting as a mean to control your robot/contraption

Implement the ball tracking from:

https://www.pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/

Implement the face tracking from:

https://docs.opencv.org/3.1.0/d7/d8b/tutorial_py_face_detection.html