ENPM673 – Perception for Autonomous Robots – Spring 2021

Project 1

Srikumar Muralidharan, Samer Charifa

Due date: 8th March 2021, 11:59PM

Submission guidelines:

- This homework is to be done and submitted individually.
- Your submission on ELMS/Canvas must be a zip file, following the naming convention **YourDirectoryID_hw1.zip**. If you email ID is abc@umd.edu or abc@terpmail.umd.edu, then your Directory ID is **abc**. Remember, this is your directory ID and NOT your UID.
- Please submit only the python script(s) you used to compute the results, the PDF report you generate for the project and a detailed README.md file.
- For each section of the project, explain briefly what you did, and describe any interesting problems you encountered and/or solutions you implemented.
- Include sample outputs in your report.
- The video outputs are to be submitted as a separate link in the report itself. The link can be a YouTube video or a google drive link (or any other cloud storage). Make sure that you provide proper sharing permission to access the files. **If we are not able to access the output files you will be awarded 0 for that part of the project.** On elms you can only submit one .zip file which can be of maximum 100 MB.
- **Allowed functions**:
  - numpy: svd, inv, matmul, etc
  - scipy: fft and inverse fft
  - some cv2 functions such as Harris corner or Shi-Tomasi, goodfeaturestotrack, sobel, canny, blob detection.
- **Disallowed functions:**
  - cv2 findHomography
  - cv2 warpPerspective
  - any inbuilt function which calculates the homography or warps the image for you.
  - any other inbuilt function that solves the question in less than 5 lines.

Introduction:

This project will focus on detecting a custom AR Tag (a form of fiducial marker), that is used for obtaining a point of reference in the real world, such as in augmented reality applications. There are two aspects to using an AR Tag, namely detection and tracking, both of which will be implemented in this project. The detection stage will involve finding the AR Tag from a given image sequence while the tracking stage will involve keeping the tag in "view" throughout the sequence and performing image processing operations based on the tag's orientation and position (a.k.a. the pose). You are provided multiple video sequences on which you test your code.

Note: You will need the intrinsic camera paramters for this project which are given here.

$$K = \begin{bmatrix} 1406.08415449821 & 0 & 0 \\ 2.20679787308599 & 1417.99930662800 & 0 \\ 1014.13643417416 & 566.347754321696 & 1 \end{bmatrix}$$

Data :

You are given 3 videos to test your code against – link.

Problem 1 – Detection **[45 Points]**

Problem 1.a) AR Code detection: [15 Points]

The task here is to detect the April Tag in any frame of Tag1 video (just one frame). Notice that the background in the image needs to removed so that you can detect the April tag. You are supposed to use Fast fourier transform (inbuilt scipy function fft is allowed) to detect the April tag. Notice that you are expected to use inbuilt functions to calculate FFT and inverse FFT.

Problem 1.b) Decode custom AR tag: [30 Points]

You are given a custom AR Tag image, as shown in Fig. 1, to be used as reference. This tag encodes both the orientation as well as the ID of the tag.

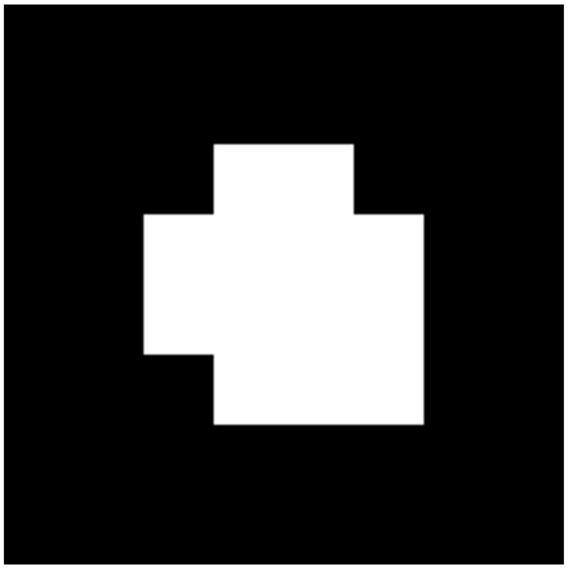
Figure 1: Reference AR Tag to be detected and tracked

Encoding Scheme:
To properly use the tag, it is necessary to understand how the data is encoded in the tag. Consider the marker shown in Fig. 2:

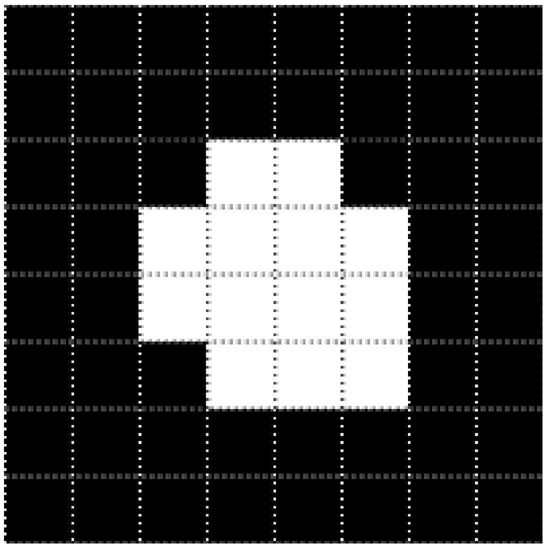
Figure 2: Grid pattern overlayed on reference AR Tag

- The tag can be decomposed into an 8 x 8 grid of squares, which includes a adding of 2 squares width (outer black cells in the image) along the borders. This allows easy detection of the tag when placed on any contrasting background.

- The inner 4 x 4 grid (i.e. after removing the padding) has the orientation depicted by a white square in the lower-right corner. This represents the upright position of the tag.
- Lastly, the inner-most 2 x 2 grid (i.e. after removing the padding and the orientation grids) encodes the binary representation of the tag's ID, which is ordered in the clockwise direction from least significant bit to most significant. So, the top-left square is the least significant bit, and the bottom-left square is the most significant bit.

You are free to develop any detection pipeline, as long as the encoding scheme specified is followed. This part of your code should return the corners of the tag as well as its ID with respect to its original orientation (compensated for any rotation you might encounter). You may use any corner detector algorithm implemented in OpenCV (such as Harris or Shi-Tomasi). Please refer to supplementary document on homography estimation for further details on how to achieve this part of your pipeline.

Problem 2 – Tracking [75 Points]

Problem 2.a) Superimposing image onto Tag: [30 Points]

Once you have the four corners of the tag, you can perform homography estimation on this in order to perform some image processing operations, such as superimposing an image over the tag. The image you will use is the testudo.png file provided, see Fig. 3. Let us call this the template image.

Figure 3: testudo.png image used as template

- The first step is to compute the homography between the corners of the template and the four corners of the tag.
- You will then transform the template image onto the tag, such that the tag is "replaced" with the template.

It is implied that the orientation of the transferred template image must match that of the tag at any given frame.

Note: For transforming the image using homography, you will develop your own version of "cv2.warpPerspecive" function. In doing so you will observe holes in your output image which depict loss of information. You can find more details on why this happens and how to come up with a solution to this problem here.

Problem 2.b) Placing a virtual cube onto Tag: [45 Points]

Augmented reality applications generally place 3D objects onto the real world, which maintain the three-dimensional properties such as rotation and scaling as you move around the "object" with your camera. In this part of the project you will implement a simple version of the same, by placing a 3D cube on the tag. This is the process of "projecting" a 3D shape onto a 2D image.

The "cube" is a simple structure made up of 8 points and lines joining them. There is no need for a complex shape with planes and shading. However, feel free to experiment.

- You will first comupte the homography between the world coordinates (the reference AR tag) and the image plane (the tag in the image sequence).
- You will then build the projection matrix from the camera calibration matrix provided and the homography matrix.
- Assuming that the virtual cube is sitting on the top of the marker, and that the Z axis is negative in the upwards direction, you will be able to obtain the coordinates of the other four corners of the cube.
- This allows you to now transform all the corners of the cube onto the image plane using the projection matrix.

Please refer to the supplementary document on homography to understand how to compute the projection matrix from the homograhy matrix.

Extra Credit **[30 Points]**

In the process of projecting the testudo image and the cube you may encounter jitters in the output due to noise in the detection process. To correct these jitters you can try and implement a simple Extended Kalman Filter (or any other filtering technique) to work as an object tracker. If you can implement this and show the improvement in the output videos with some analysis, you are entitled to 25% extra credit.