# [ENPM673] Project 1

Anton Mitrokhin, Kanishka Ganguly, Cornelia Fermüller

**Due Date: February 27, 2019**

## 1 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 to test your code.

## 2 Detection

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.

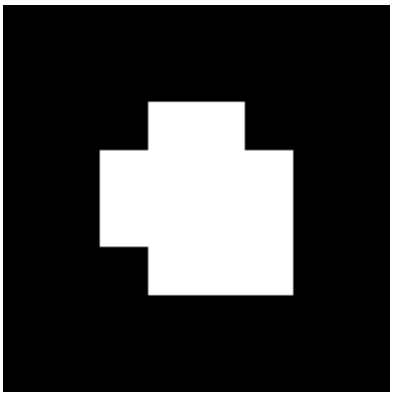

Figure 1: Reference AR Tag to be detected and tracked

### 2.1 Encoding Scheme

In order 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

- The tag can be decomposed into an $8 \times 8$ grid of squares, which includes a padding of 2 squares width along the borders. This allows easy detection of the tag when placed on white background.
- The inner $4 \times 4$ 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.

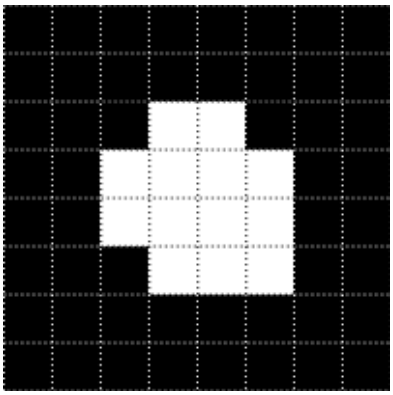

Figure 2: Grid pattern overlayed on reference AR Tag

- Lastly, the inner-most $2 \times 2$ 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 (i.e. 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.

# 3 Tracking

## 3.1 Superimposing an image onto the tag

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 `Lena.png` file provided, see Fig. 3. Let us call this the *template image*.

Figure 3: `Lena.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 transformed template image must match that of the tag at any given frame.

### 3.2 Placing a virtual cube on the tag

Augmented reality applications generally place 3D objects onto the real world, which maintain 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 attempt to 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 compute 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 "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 homography matrix.

## 4 Submission Guidelines

The data can be downloaded from here. Also, please submit only the python script(s) you used to compute the results and the PDF report you generate for the project. All sample outputs can be included as screenshots in the PDF itself.