**ASSIGNMENT 3**

**REPORT - OPTICAL GLYPH**

Deepthy Rose Jose

AM20S052

Pose estimation is of great importance in many computer vision applications: robot navigation, augmented reality, and many more. This process is based on finding correspondences between points in the real environment and their 2d image projection. This is usually a difficult step, and thus it is common to use synthetic or fiducial markers to make it easier.

One of the most popular approaches is the use of binary square fiducial markers. The main benefit of these markers is that a single marker provides enough correspondences (its four corners) to obtain the camera pose. Also, the inner binary codification makes them specially robust, allowing the possibility of applying error detection and correction techniques.

The aruco module is based on the [ArUco library](http://www.uco.es/investiga/grupos/ava/node/26), a popular library for detection of square fiducial markers developed by Rafael Muñoz and Sergio Garrido:

**Markers and Dictionaries**

An ArUco marker is a synthetic square marker composed by a wide black border and an inner binary matrix which determines its identifier (id). The black border facilitates its fast detection in the image and the binary codification allows its identification and the application of error detection and correction techniques. The marker size determines the size of the internal matrix. For instance a marker size of 4x4 is composed by 16 bits.



It must be noted that a marker can be found rotated in the environment, however, the detection process needs to be able to determine its original rotation, so that each corner is identified unequivocally. This is also done based on the binary codification.

A dictionary of markers is the set of markers that are considered in a specific application. It is simply the list of binary codifications of each of its markers.

* The main properties of a dictionary are the dictionary size and the marker size.
* The dictionary size is the number of markers that compose the dictionary.
* The marker size is the size of those markers (the number of bits).

**Marker Detection**

Given an image containing A markers, the detection process has to return a list of detected markers. Each detected marker includes:

* The position of its four corners in the image (in their original order).
* The id of the marker.

The marker detection process is comprised of two main steps:

1. Detection of marker candidates. In this step the image is analyzed in order to find square shapes that are candidates to be markers. It begins with an adaptive thresholding to segment the markers, then contours are extracted from the thresholded image and those that are not convex or do not approximate to a square shape are discarded. Some extra filtering is also applied (removing contours that are too small or too big, removing contours too close to each other, etc).
2. After the candidate detection, it is necessary to determine if they are actually markers by analyzing their inner codification. This step starts by extracting the marker bits of each marker. To do so, a perspective transformation is first applied to obtain the marker in its canonical form. Then, the canonical image is thresholded using Otsu to separate white and black bits. The image is divided into different cells according to the marker size and the border size. Then the number of black or white pixels in each cell is counted to determine if it is a white or a black bit. Finally, the bits are analyzed to determine if the marker belongs to the specific dictionary. Error correction techniques are employed when necessary.

**Pose Estimation**

The next thing you probably want to do after detecting the markers is to obtain the camera pose from them.

To perform camera pose estimation you need to know the calibration parameters of your camera. These are the camera matrix and distortion coefficients. If you do not know how to calibrate your camera, you can take a look at the [calibrateCamera()](https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html" \l "ga3207604e4b1a1758aa66acb6ed5aa65d" \o "Finds the camera intrinsic and extrinsic parameters from several views of a calibration pattern...) function and the Calibration tutorial of OpenCV. You can also calibrate your camera using the aruco module as explained in the Calibration with ArUco and ChArUco tutorial. Note that this only needs to be done once unless the camera optics are modified (for instance changing its focus).

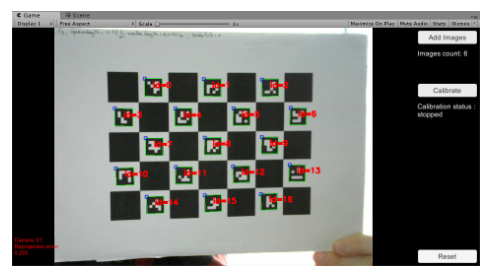
In the end, what you get after the calibration is the camera matrix: a matrix of 3x3 elements with the focal distances and the camera center coordinates (a.k.a intrinsic parameters), and the distortion coefficients: a vector of 5 or more elements that models the distortion produced by your camera.

When you estimate the pose with ArUco markers, you can estimate the pose of each marker individually. If you want to estimate one pose from a set of markers, use ArUco Boards. Using ArUco boards instead of single markers allows some markers to be occluded.

The camera pose with respect to a marker is the 3d transformation from the marker coordinate system to the camera coordinate system. It is specified by rotation and translation vectors (see [solvePnP()](https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html" \l "ga549c2075fac14829ff4a58bc931c033d" \o "Finds an object pose from 3D-2D point correspondences. This function returns the rotation and the tra...) function for more information).

The aruco module provides a function to estimate the poses of all the detected markers:The marker coordinate system that is assumed by this function is placed at the center of the marker with the Z axis pointing out, as in the following image. Axis-color correspondences are X: red, Y: green, Z: blue. Note the axis directions of the rotated markers in this image.

[The video of Calibration is attached with this file]



**Library details : ArucoUnity Architecture**

ArucoUnity is made of three parts:

1. A [plugin](https://github.com/NormandErwan/ArucoUnityPlugin/) that wraps into a C interface the aruco, calib3d and ccalib modules of OpenCV.
2. A C# interface ([ArucoUnity.Plugin namespace](https://github.com/NormandErwan/ArucoUnity/blob/master/Assets/ArucoUnity/Scripts/Plugin/)) using the plugin to reproduce the OpenCV modules classes and functions.
3. [Unity C# scripts](https://normanderwan.github.io/ArucoUnity/api/ArucoUnity.Calibration.html) to calibrate cameras and to track markers directly in the editor with good performances.

You can code directly with the OpenCV C# equivalent interface but we advise you to work with and extend the Unity scripts. The Unity scripts were originally one camera display and tracking script and one calibration script. For performances and to support multiple types of camera (fisheye, stereoscopic) we decoupled these scripts

**[[ PLEASE FIND THE ATTACHED VIDEO WITH THE DOCUMENT]]**