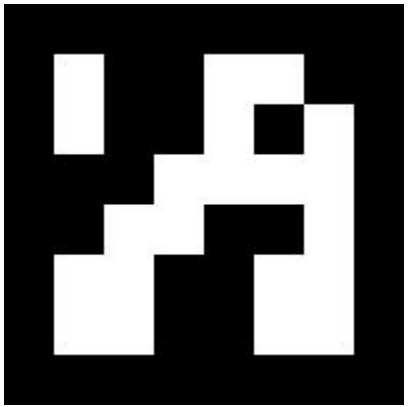
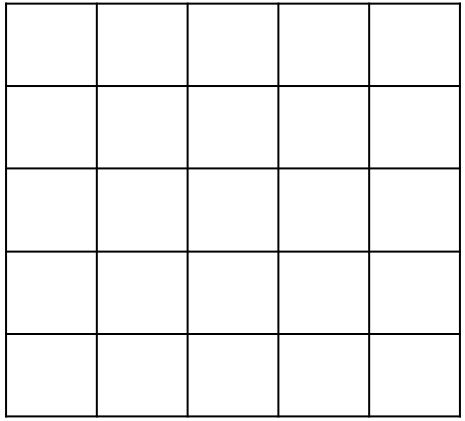
ARUCO MARKERS



- An ArUco marker is a NxN grid, that is black and white in color, where N=4 to 7. ArUco markers are based on Hamming code.
- Consider for example an ArUco marker of 5x5 grid as shown in figure below:



- In this grid, the first, third and fifth columns represent parity bits. The second and fourth columns represent the data bits.

- Thus, from 2 data bit columns there are a total of 10 data bits. So, the maximum number of markers that can be encoded are: 2^10 = 1024. Thus, a 5x5 ArUco marker can have a maximum of 1000 combinations or IDs.

ENCODING

- Suppose we choose the example ID of 650 i.e. ArUco ID=650; its binary representation is 1010001010 i.e. 10 Bits.
- Each row of the grid is encoded separately using a slightly modified Hamming code for each parity bit i.e.

    1. The first parity Bit is calculated using even parity
    2. The second parity Bit is calculated using odd parity
    3. The third parity Bit is calculated using even parity

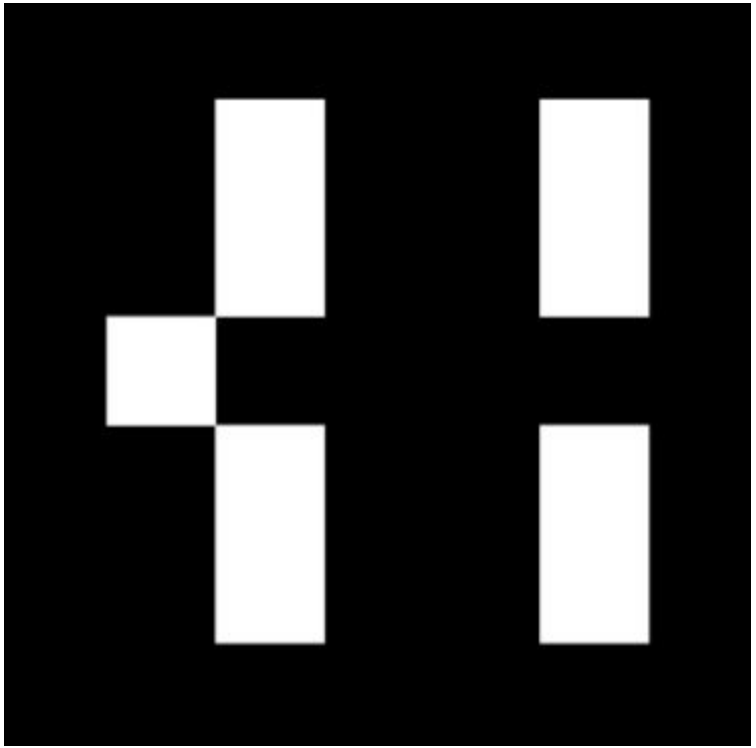
Let’s fill out the ArUco grid for ID = 650:

5	4	3	2	1
Data Bit 2	Parity bit 3	Data bit 1	Parity bit 2	Parity bit 1
0	0	1	0	1
0	0	1	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	0	1

- All the bit positions that are a power of 2 are marked as parity bits (1, 2, 4, 8, etc).
- All the other bit positions are marked as data bits.

By rearranging we get the following ArUco grid:

Parity 1	Data 1	Parity 3	Data 2	Parity 2
0	1	0	0	1
0	1	0	0	1
1	0	0	0	0
0	1	0	0	1
0	1	0	0	1



ArUco Library

- ArUco is an easy to use Python library for generation and detection of ArUco markers.

Generating ArUco Markers

- Create a Python script and add the required libraries.
  - **import numpy as np**
  - **import math**
  - **import cv2**
  - **import cv2.aruco as aruco**

- Select a Dictionary provided by the ArUco library module.
  - **aruco_dict = aruco.Dictionary_get(aruco.DICT_5X5_250)**
    - aruco.DICT_5x5_250: This is an example of a predefined dictionary of markers with 5x5 Bit size and a total of 250 marker ID combinations.
    - Some other ArUco Dictionaries you can choose from in OpenCV :

      DICT_4X4_250

      DICT_4X4_1000

      DICT_6X6_250

      DICT_ARUCO_ORIGINAL etc.

- aruco_dict: A Dictionary object of predefined dictionary (DICT_5x5_250 with respect to the example above)

- Generating markers of any ID from the specified dictionary with a required output image resolution.
  - **img = aruco.drawMarker(aruco_dict, 11, 400)**

    - *aruco_dict*: Dictionary object previously created.
    - *11*: Marker ID. In this case the marker 11 of the dictionary DICT_5x5_250. Since there are 250 possible combinations of IDs, the valid ids go from 0 to 249.
    - *400*: Resolution of the output marker image. In this case, the output image will have a size of 400x400 pixels.
    - *img*: ArUco marker image.

Detection of ArUco markers:

- Create a Python script and add the required libraries.
  - **import numpy as np**
  - **import math**
  - **import cv2**
  - **import cv2.aruco as aruco**
- Load the corresponding image matrix to a variable, for example img, using cv2.imread() operation.
- Convert the image matrix from RGB to grayscale using cv2 operation.
  - **gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)**
- Select a Dictionary provided by the aruco module. For example, if the ArUco belongs to the 5x5_250 dictionary, then -
  - **aruco_dict = aruco.Dictionary_get(aruco.DICT_5X5_250)**

- Create an object variable. For example, parameters: which includes all the options that can be customized during the marker detection process.
  - **parameters = aruco.DetectorParameters_create()**
- Detect the markers of the corresponding dictionary from the grayscale image considering marker detection parameters.
  - **corners, ids, _ = aruco.detectMarkers(gray, aruco_dict, *parameters* = parameters)**
    - *gray*: Grayscale image of the test image.
    - *aruco_dict*: Dictionary object previously created.
    - *parameters*: object returned by aruco.DetectorParameters_create()
    - *ids*: list of aruco id in the test image. If there are N markers in test image, then the size of the list becomes N.
    - *corners*: 2D position of its corner in the image and its corresponding identifier. It is a numpy array of corners of the detected markers. For each marker, its four corners are returned in their original order (which is clockwise starting with top left). So, the first corner is the top left corner, followed by the top right, bottom right and bottom left. For N detected markers, the dimension of corners will be Nx4.