An Introduction to ArUco Markers

ArUco marker is a 5x5 grid that is black and white in color. ArUco markers are based on Hamming code. In the grid, the first, third and fifth columns represent parity bits. The second and fourth columns represent the data bits. Hence, there are ten total data bits. So the maximum number of markers that can be encoded are- 2^10 = 1024. Data and parity bits in an ArUco marker are as shown in Figure 1:

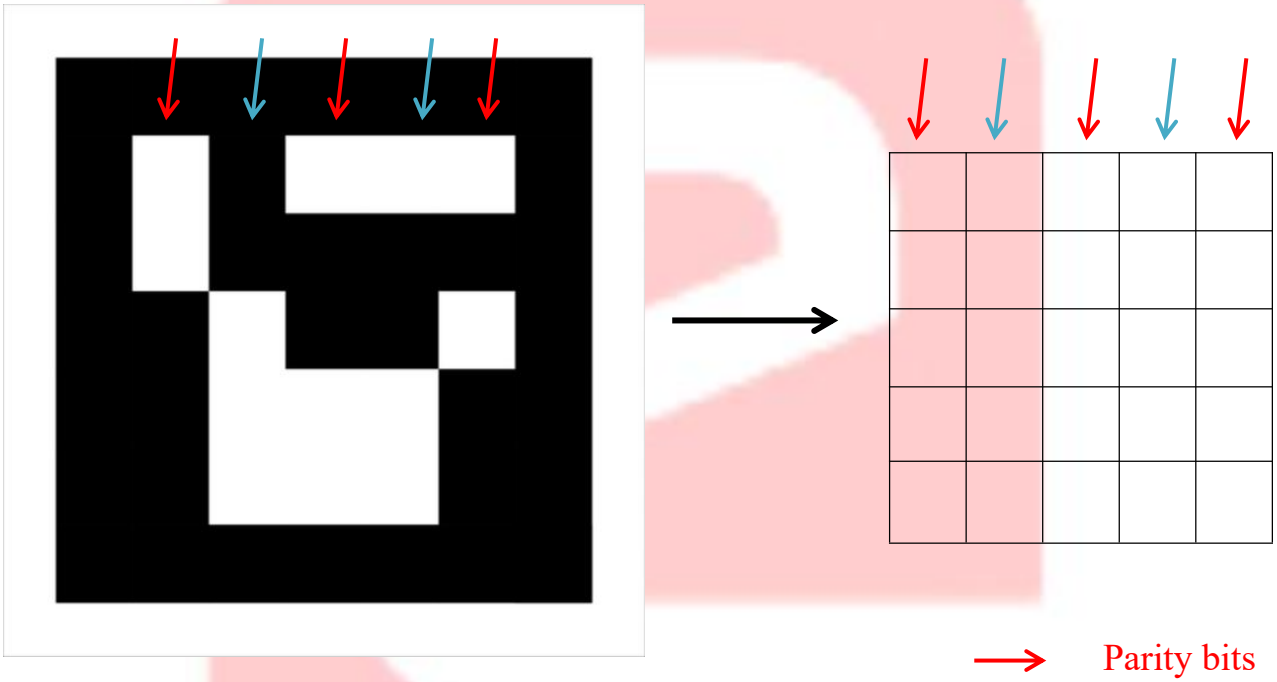

Figure 1: ArUco Marker

Encoding:

Let us consider the number 650. Its binary representation is 1010001010. As shown above in Figure 1, there are two data bits in each row.

Now, each row is encoded separately using slightly modified Hamming code. The first and third parity bits are calculated using even parity while the second parity bit uses odd parity. The number 650 is filled accordingly and we get the following encoded values:

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

We rearrange the bits in each row and get the following data rows as shown in Figure2:

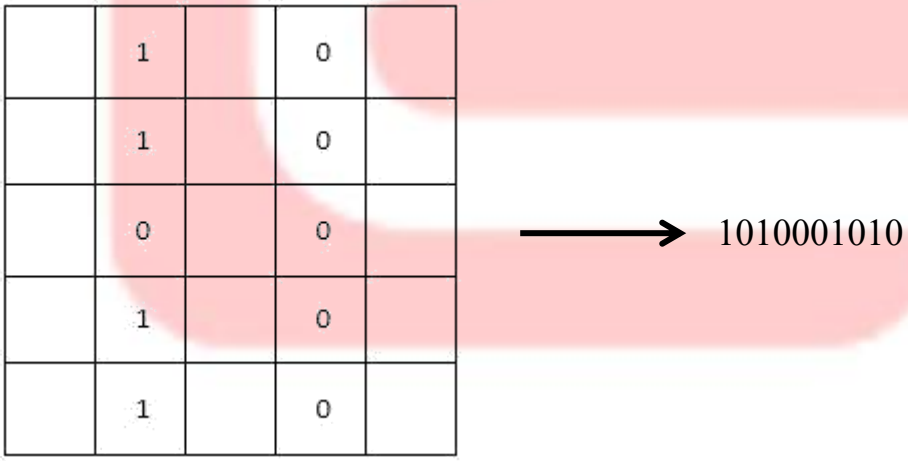

Figure 2: Data Rows

After arranging the parity bits as well, ArUco bits for the number 650 looks as shown in Figure3:

Parity1	Data1	Parity3	Data2	Parity2
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

Figure 3: ArUco bits for 0b1010001010

If the cell value is 0, color it black; if value is 1, color it white. This will give us the ArUco marker as shown in Figure 4:

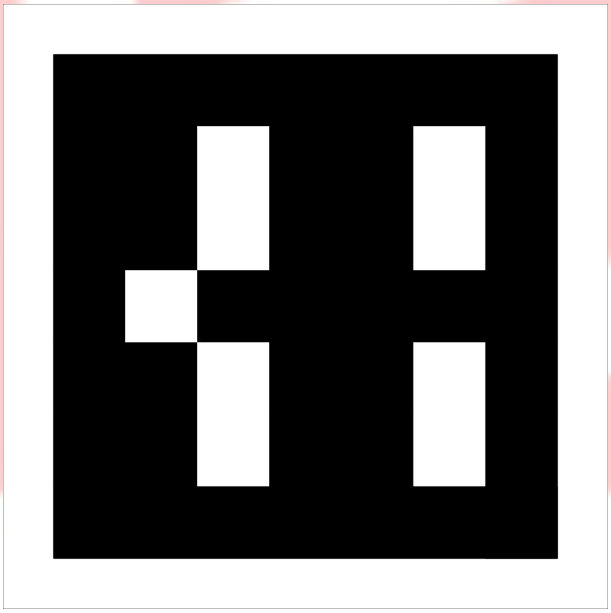

Figure 4: ArUco marker for 650

You can find that the above marker is padded with one layer of black cells. That layer will be removed during decoding.

Decoding:

After understanding the above section, decoding is an extremely simple process. The following steps are to be followed while decoding a perfect, computer-generated image of an ArUco marker.

Step 1: Extract the ArUco from the image.

Step 2: Remove the extra padding.

Step 3: Divide the resulting image into a 5x5 grids and check the color in each cell of the second and fourth columns (in that order) in a top to bottom manner.

Step 4: If the color is white, write 1; else, write it 0.

Step 5: The resulting number will be in binary. Convert it into decimal

Generating ArUco markers:

You can generate ArUco marker from this link. The task_0 uses ArUco marker of id's: 1,2 and 3 of size 150 mm with 20 mm padding.

How to use the ArUco ROS package?

Follow the steps given below:

1. Change the directory to your catkin workspace and clone the github repository in "src" folder of workspace by typing the following command:

   ```
   >> git clone https://github.com/pal-robotics/aruco_ros.git
   ```

2. The above command will clone aruco-ros package into your workspace. To build this package navigate to catkin workspace and run the following command:

   ```
   >> catkin_make
   >> source devel/setup.bash
   ```

3. Navigate to "launch" folder of "aruco_ros" folder in the package. You will find a file "*marker_publisher.launch*". This launch file initializes the run, all the required nodes and algorithm. This launch file subscribes and publishes the topics as follows:
   a) **Subscriber:**
      i. **/camera_info :** This topic contains the necessary information about camera like frame_id, height, width etc
      ii. **/image :** This topic contains the image on which the algorithm performs.

   Map the above topics to their corresponding requirements.

b) **Publisher:**
   i. **/aruco_marker_publisher/result:** This topic publishes the output image after running the algorithm on the camera frame image.
   ii. **/aruco_marker_publisher/markers:** This topic has the information of the pose of each ArUco marker within the camera frame.

NOTE: The above discussed topics are relevant to the task you are working on. aruco_marker_publisher also publishes other topics like /aruco_marker_publisher/marker_list and aruco_marker_publisher/debug. You may check for all topics using the command: "*rostopic list*"

As discussed in step 3, *marker_publisher* node subscribes to topics "/camera_info" and "/image". *aruco_ros* package provides output only when it receives the information on these topics. Now the question is - who will publish the information on these topics so that *marker_publisher* node can take that topic and apply the algorithm to detect the ArUco markers?. Let us take a look at the "marker_publisher.launch" in the launch folder of *aruco_ros.* **Do not make any changes to the *marker_publisher.launch* when you go through the following details. Just understand what are the different steps involved. The changes mentioned below will already be made for you in the repository which you will be downloading for Task 0.**

1. Let us take a look at "*marker_publisher.launch*" of *aruco_ros* package. This file is explained line by line as:

```
1  <launch>

2      <arg name="markerSize"      default="0.05"/>    <!-- in m -->

3      <arg name="side"            default="left"/>

4      <arg name="ref_frame"       default="base"/>
```

These are the arguments that are passed to the marker_publisher node.

I. Size of the ArUco marker. Change the default value to "0.15" as the ArUco marker in the scene is of 150 mm dimension.
II. Ref_frame indicates the frame in which the marker pose must be referred to. As we want the pose wrt parent_name, delete "base" and leave it empty as suggested in the comments. You will receive error if you don't change the reference as the target_frame does not exist.

```
5       <node pkg="aruco_ros" type="marker_publisher" name="aruco_marker_publisher">
6           <remap from="/camera_info" to="/cameras/$(arg side)_hand_camera/camera_info" />
7           <remap from="/image" to="/cameras/$(arg side)_hand_camera/image" />
8           <param name="image_is_rectified" value="True"/>
9           <param name="marker_size"        value="$(arg markerSize)"/>
10          <param name="reference_frame"    value="$(arg ref_frame)"/>
11          <param name="camera_frame"       value="$(arg side)_hand_camera"/>
12      </node>
```

The above node will run the algorithm by taking the image input on topic "/image" and camera information on topic "/camera_info" .

2. Let us suppose that the image is published on topic "task_0/image_raw". The algorithm is to be performed on this topic. Remaining information about the image is published on "task_0/camera_info". Both these topics should be given to the aruco_ros package in order to perform the algorithm on the image. Use *rqt_graph* for debugging.Then you have to use the remap parameter between the topics to connect them according to our requirement as you have learned from the video tutorial. More information about remap is explained *here*.
3. After a successful remapping between topics you can check the output of aruco_ros package by running the following command:

```
>> rosrun image_view image_view image:=/aruco_marker_publisher/result
```

The above command will show the image published by "*/aruco_marker_publisher/result*" topic. This topic is the output image of marker_publisher node. image_view is basically a ROS node inside a ROS package named image_view and takes image as the argument.

4. As you will have to run the node each time to see the output, it will be better if you call the node inside the launch file itself. In order to do that, you can add the following lines into the launch file to do same.

```
13      <node ns = "aruco" name="image_view" type="image_view"
    pkg="image_view" output="screen">
14          <remap from="image" to="/aruco_marker_publisher/result"/>
15      </node>
16  </launch>
```

All The Best!!