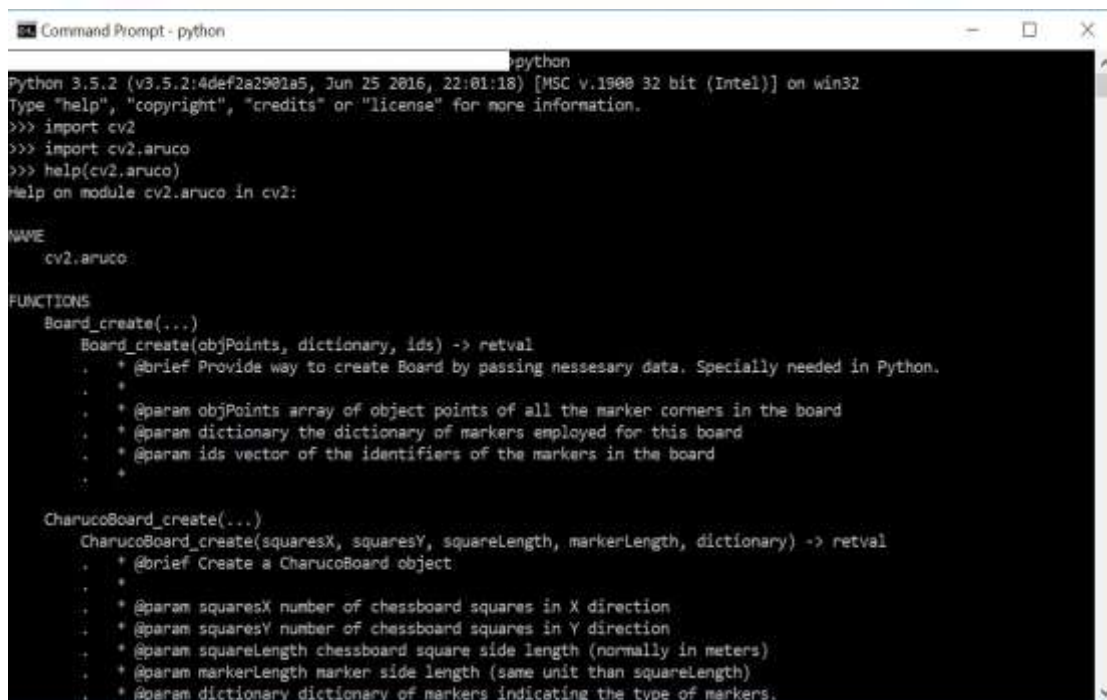


e-Yantra Robotics Competition (eYRC-2018)

ArUco Library**Introduction:**

ArUco is an easy to use Python library for generation and detection of ArUco markers. To know about the library functions of ArUco, open the terminal or command prompt and type **Python**. Then type the following commands and press Enter as shown in Figure 1 below:



```
Command Prompt - python
python
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> import cv2.aruco
>>> help(cv2.aruco)
Help on module cv2.aruco in cv2:

NAME
  cv2.aruco

FUNCTIONS
  Board_create(...)
    Board_create(objPoints, dictionary, ids) -> retval
    * @brief Provide way to create Board by passing necessary data. Specially needed in Python.
    *
    * @param objPoints array of object points of all the marker corners in the board
    * @param dictionary the dictionary of markers employed for this board
    * @param ids vector of the identifiers of the markers in the board
    *

  CharucoBoard_create(...)
    CharucoBoard_create(squaresX, squaresY, squareLength, markerLength, dictionary) -> retval
    * @brief Create a CharucoBoard object
    *
    * @param squaresX number of chessboard squares in X direction
    * @param squaresY number of chessboard squares in Y direction
    * @param squareLength chessboard square side length (normally in meters)
    * @param markerLength marker side length (same unit than squareLength)
    * @param dictionary dictionary of markers indicating the type of markers.
```

Figure 1. Accessing description of ArUco library functions (OpenCV-Python library)

A list of all available functions in the ArUco library can be browsed one after another on the terminal. The content of library functions in the ArUco module is self-explanatory.

Generating an ArUco Marker:

Step 1: Create a Python script and add the required libraries

```
import numpy as np
import math
import cv2
import cv2.aruco
```

Step 2: Select a Dictionary provided by the ArUco library module.

```
aruco_dict = aruco.Dictionary_get(aruco.DICT_5X5_250)
```

Parameters:

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.

Here is a list of ArUco Dictionaries you can choose from in OpenCV:

DICT_4X4_50

DICT_4X4_100

DICT_4X4_250

DICT_4X4_1000

DICT_5X5_50

DICT_5X5_100

DICT_5X5_250

DICT_5X5_1000

DICT_6X6_50

DICT_6X6_100

DICT_6X6_250

DICT_6X6_1000

DICT_7X7_50

DICT_7X7_100

DICT_7X7_250

DICT_7X7_1000

DICT_ARUCO_ORIGINAL

In general, Lower Bit sizes and higher number of combinations increase the inter-marker distance and vice-versa.

Return:

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

Step 3: Generating markers of any ID from the specified dictionary with a required output image resolution.

```
img = aruco.drawMarker(aruco_dict, 11, 400)
```

Parameters:

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**.

Note: Any specific ID out of the valid range will produce an exception.

400: Resolution of the output marker image. In this case, the output image will have a size of **400x400** pixels.

Note: This parameter should be large enough to store the number of bits for the specific dictionary. So, for instance, you cannot generate an image of 5x5 pixels for a marker size of 6x6 bits (and that is without considering the marker border). Furthermore, to avoid deformations, this parameter should be proportional to the (**number of bits + border size**), or **much higher than** the marker size (like 400 in the example), so that deformations are insignificant

Return:

Img: ArUco marker image.