e-Yantra Robotics Competition (eYRC-2018)

ArUco Library

Introduction

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

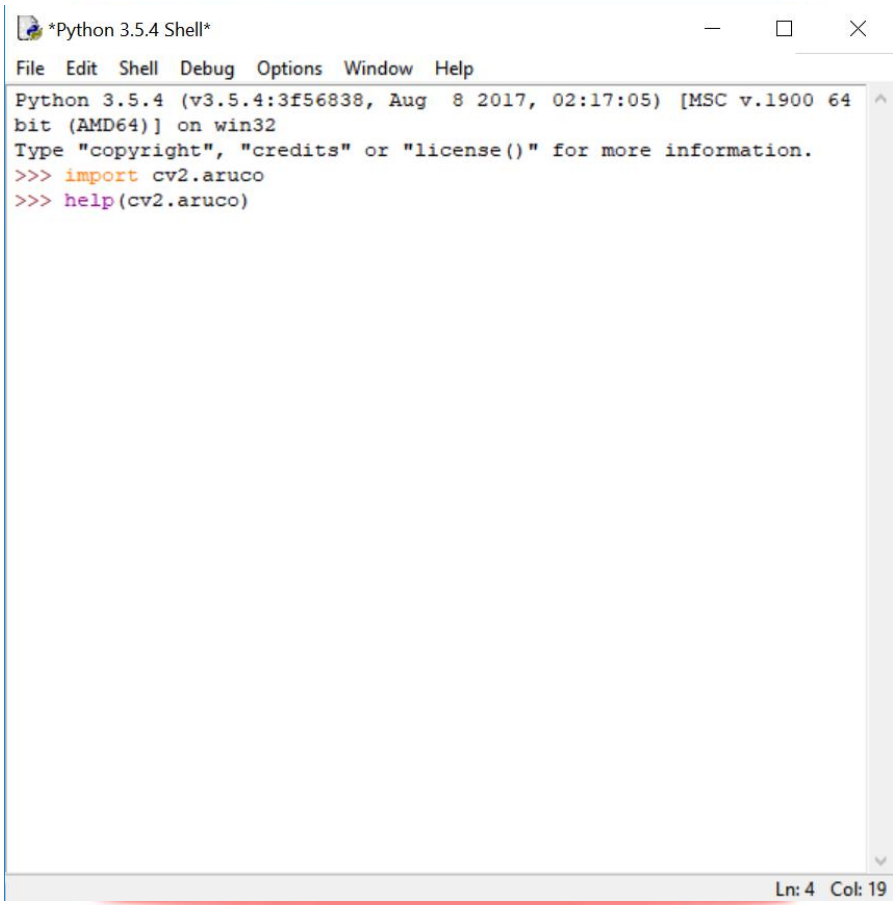

Figure 1:Accessing description of ArUco library functions

Then you can see all the available functions in ArUco library. These contents are self-explanatory of library functions in aruco module.

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 module.

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

Parameters:
aruco.DICT_5x5_250: 5x5 is an example of predefined dictionary of markers with 5x5 bits and a total of 250 markers. In general, Lower dictionary sizes and higher marker sizes increase the inter-marker distance and vice-versa. However, the detection of markers with higher sizes is more complex, due to the higher amount of bits that need to be extracted from the image. For instance, if you need only 10 markers in your application, it is better to use a dictionary composed by those 10 markers than using one dictionary composed by 1000 markers. The reason is that the dictionary composed of 10 markers will have a higher inter-marker distance and, thus, it will be more robust to errors.

Return:
aruco_dict: a Dictionary object of predefined dictionary (DICT_5x5_250)

Step 3: Generating markers of any id from the defined dictionary with a required output image size.

```
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. In this case, the valid ids go from 0 to 249. Any specific id out of the valid range will produce an exception.

400: Size of the output marker image. In this case, the output image will have a size of 400x400 pixels. Note that 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 at least much higher than the marker size (like 400 in the example), so that deformations are insignificant.

Return:
Img: ArUco marker image.

Detection of ArUco Markers

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

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

Step 2: Load the corresponding image matrix to a variable, for example img, using cv2 operation.

Step 3: Convert the image matrix from RGB to grayscale using cv2 operation.

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

Step 4: Select a Dictionary provided by the aruco module.

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

Step 5: 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()
```

Step 6: Detect the markers of the corresponding dictionary from the grayscale image considering marker detection parameters.

```
corners, ids, _ = aruco.detectMarkers(gray, aruco_dict,
parameters = parameters)
```

Parameters:

gray: Grayscale image of the test image.

aruco_dict : Dictionary object previously created.

Parameters: object returned by aruco.DetectorParameters_create()

Return:

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. For N detected markers, the dimension of corners will be Nx4.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.

Pose Estimation of Aruco Markers

Step 7: Import the Camera Matrix and Distortion Matrix. Camera matrix and Distortion matrix were obtained in Task 0.2 and saved in a .npz file.

```
with np.load('Camera.npz') as X:
      camera_matrix, dist_coeff, _, _ = [X[i] for i in
('mtx','dist','rvecs','tvecs')]
```

Step 8: Estimate the Pose of ArUco markers. In order to estimate Pose, we need to obtain the rotation vector and translation vectors. They are obtained using the the following function given below.

```
rvec, tvec= aruco.estimatePoseSingleMarkers(corners,
markerLength, camera_matrix, dist_coeff)
```

Parameters
corners: 2D position of corners in the image. Obtained during detection of ArUco markers.

markerLength: Length of side of Aruco Marker. Default value of this variable is 100.

camera_matrix: Camera matrix extracted from .npz file

dist_coeff: Distortion matrix extracted from .npz file

Return
rvec : Rotation Vector of ArUco marker

tvec : Translation Vector of ArUco marker

Draw Axis on Aruco Marker

The aruco module in openCV gives a handy function to draw axes perpendicular to the ArUco marker as shown in Figure 2.

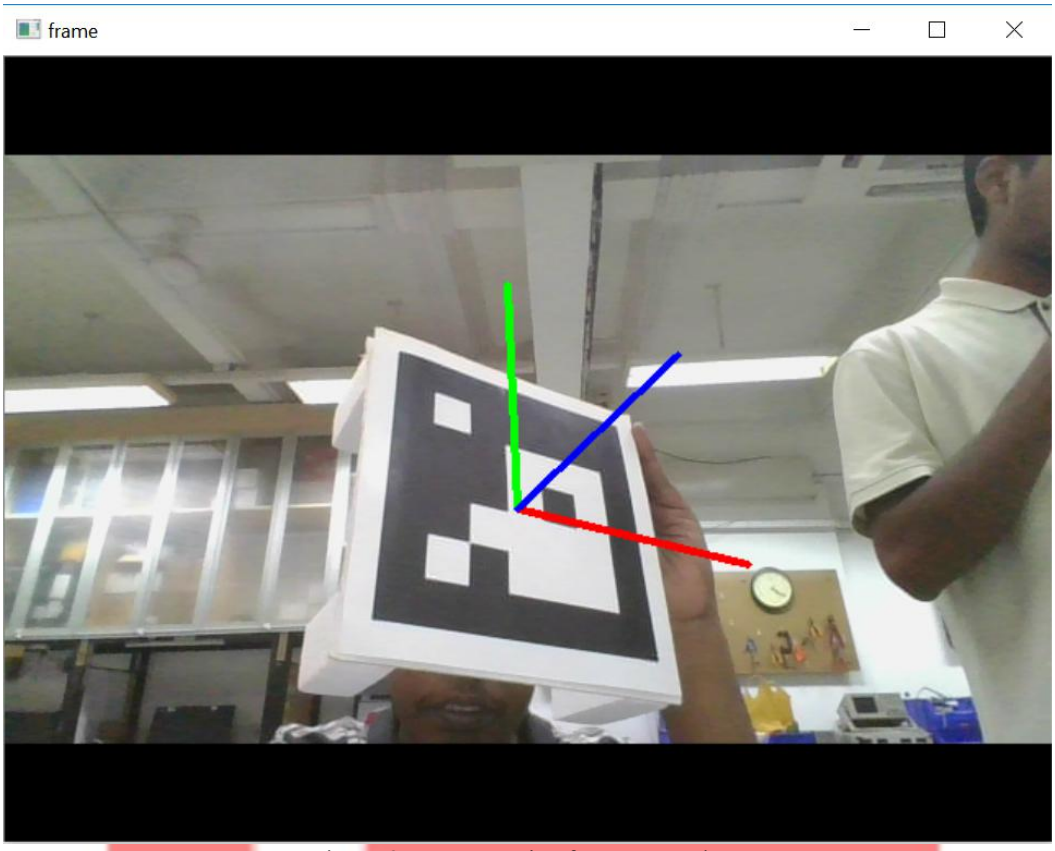

Figure 2: X, Y, Z axis of ArUco markers

```
aruco.drawAxis(img, camera_matrix, dist_coeff, rvec, tvec, length)
```

Parameters

img **-** Image expressed as a numpy array.

camera_matrix: Camera matrix extracted from .npz file

dist_coeff: Distortion matrix extracted from .npz file

rvec :Rotation Vector of ArUco marker calculated by estimatePoseSingleMarkers()

tvec :Translation Vector of ArUco marker calculated by estimatePoseSingleMarkers()

length: Length of axis. Default value of this variable is 100.

e-yantra Robotics Competition

2018