DETECTION AND POSE ESTIMATION

Amrit Lal Singh

***Abstract*— In this we attempt to detect the ArUco marker along with identifying and conveying its pose through visual cues that involve drawing the axes on the video input and showing it in real time.**

The edges of markers are highlighted by the code to convey what we are detecting, and the marker tag is displayed. The fields of application of this are wide, from detection of the pose in space to ships and the relative motion of them from camera stabilization to its extensive use in robotics when interfacing robots with real-world objects. Using this, any ariel vehicle can detect its relative positioning and complete tasks. It could and is being used extensively in identification, let's say a robot has to select a hammer from a collection of tools. A way to do this would be to use deep learning-based image recognition for hammers. Otherwise, using this we could label the hammers with these and then we could easily identify the hammer along with detecting its pose along the way. This data could also help us to pick it up. We could also use ArUco to determine the trajectory of an object. Suppose a cube is thrown through the air, using ArUco we could detect the trajectory of the ArUco-marked cube its orientation its rate of rotation and much more. We could predict where it will fall, and what side will face down. It could be used by a system to check quickly if everything is in its right position. This technology is further empowered by the fact that it uses only the data provided by a camera which is cheaply available and to construct a tag only a printer is required. In the use cases of ariel vehicles it could be used to detect landing spots and could help us to land better, could be used to identify other UAVs, and identify packages to be carried and delivered as delivery using UAVs is an emerging field.

I. INTRODUCTION

The problem statement required us to detect the ArUco marker given along with its pose. the problem statement also wanted us to do the detection part without using in-built functions. Doing this via the data from my camera will require calibration, as the code or the camera does not know how will a point in front of the camera lens will project on the camera sensor. Along with this, the detection will need a dictionary of ArUcos to identify the one given also we do not know what type of ArUco is given to us, but we could by the looks of it determine that it is a 6×6 ArUco and proceed. How the detection works is that it first isolates objects that are square or projections of square and stores them as the four corners of the closed objects. There is a function for minimum or max size of these. then these corners are passed to the ArUco detector which realises some of these squares are not ArUco and rejects these and in the end gives us the corners of the selected tag and the ID of it. We use the corners we got to display the edges of the ArUco along with writing the tag identity along with it.

*The internet and to it's seeming infinite store of knowledge. The gods of stack overflow

II. PROBLEM STATEMENT

In this task, you have to take a picture of the ArUco marker given below and estimate its relative pose. You need to print the pose and tag id on the image. You can use inbuilt functions to find the pose of the ArUco. In this we take image data as an input, the data could be an image or a stream of frames in order to give live output. If an ArUco tag is present in the image then we have to detect the tag. In the first part we are allowed to use inbuilt functions to detect the ID and pose of the ArUco tag, however, you should be able to tell how they work, their limitations and the different parameters it takes up and how it affects its working. You also need to detect and estimate the pose using live video capture from your webcam at different distances.

As you now have the knowledge of detecting ArUco tags using inbuilt functions, we want you to detect the tag using your own functions. You can use a number of methods to detect the ArUco tag. Not using inbuilt functions for pose estimation will fetch you extra points. The detection of the marker has to be accomplished without using the functions we just used in the first part. That is an ArUco detection from scratch.

III. RELATED WORK

The OpenCV documentation documents the functions we will use to attempt the first part of the task. Otherwise, such task has been accomplished in c++. The second part has not been accomplished aside from the OpenCV library itself and the detection can not be extracted from the library without including the library itself. The second part's related work was not found, although what we use for scanning documents could be helpful.

IV. INITIAL ATTEMPTS

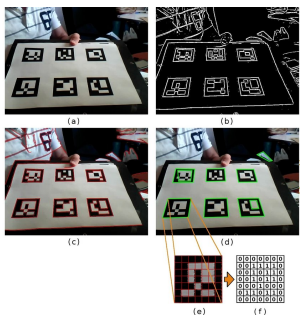

Fig. 1. ArUco detection

V. Final Approach

Markers are reference shapes which can be helpful to give some info to change our space from 2D to 3D. ArUco markers are normally squared binary (BlackWhite) markers. These markers are stored in the ArUco dictionary as binary. After the detection of the markers in the image, they are compared to the ones in the dictionary and calculated which marker has which ID.

According to the OpenCV documentation, the steps to find ArUco Markers in the images are below:

Finding Marker Candidates: The first step is finding square shapes so that we can have all the candidates. We need to apply the adaptive threshold to the image first. Adaptive thresholding is done by applying window sliding(3x3, 5x5, 11x11, . . .) and finding the optimum greyscale value for each window. Values that are below the calculated value will be black, and above is white. From the binary image, we find contours. If they are not convex or close to a square shape, they will be dropped. These conditions are defined with some filters (too big or small edges, the distances between the edges, etc.)

Perspective transform will be applied to square-shaped candidates. This form is known as the canonical form. After that, otsu thresholding will be applied. Otsu thresholding is finding an optimum point of the histogram of the image so that the threshold of the value will minimize black(background) and white(foreground) distribution differences. Let's say we are searching for 5x5 markers(it's defined in the code, we'll come to that). The last version of the images will be divided into 5x5 sub-images(with the border, 7x7). Since they are already thresholded, we have binary images and it's now easy to convert them to a binary matrix! These matrices will be searched in the dictionary and if they match with the markers, we'll have their IDs.

The first thing is choosing the dictionary. ArUco markers have sizes from 4x4 to 7x7 with 50,100,250 and 1000 available ids. If you need fewer markers, use a smaller dictionary because the search will be faster. Get the dictionary with Dictionary get. Create detector parameters to detect them in the image. Now we can use detectMarkers function to find our markers.

If ids not empty, we have the markers. We can iterate over the corners or ids and use estimatePoseSingleMarkers function. Parameters are almost the same, except 0.02 value. This is my marker size, in the same metric unit that I calculated my calibration matrix. We'll get rvec and tvec vectors which are rotation and translation vectors I mentioned above. We can also use drawAxis to see markers and make sure they are found correctly. I gave frame which is the first image I took from the camera, RGB. Value 0.01 is the length of the axis, I used half-length of the marker since the center of the marker is the point axis will be drawn.

You can track the ArUco markers now. The whole function is below:

We run the calibration-distortion code on the pictures of chess board the central piece in this is the cv2.findChessboardCorners() function and the cv2.calibratecamera the first one takes the pictures and returns corners which we append into the image points list. The objectpoints are appended into the objpoint list. After the data of all the images is appended into these lists. The lists are then input into cv2.calibrateCamera() which returns us the Camera matrix and Distortion Coefficients.

The main code works by first importing NumPy, opencv-contrib-python as cv2 and cv2.ArUco. Then we capture the live video by our primary camera, by writing cv2.Videocapture(0). The next lines of code stores the distortion coefficients and the camera matrix given by the calibration code. Then we start an infinite while loop and inside it we capture the current frame from our video input and convert it to greyscale, that reduces the computational power needed to compute each frame. Then after initialising the detector parameters with default values we run the ArUco.detectMarkers() with the greyscale frame that we formed along with the dictionary type, camera matrix and distortion coefficient. Then we draw lines on the edges of the detected markers. Also, the detect marker functions returns the corners which then we input into the estimate pose function along with the camera matrix and distortion coefficients to the ArUco.estimatePoseSingleMarkers() to obtain the rotation vector and the translation vector. now since we have the rotation and translation vector we input them into cv2.drawAxis to draw the pose in the image itself to understand if our code understands the pose well. Which it does!

The relative pose of the ArUco with respect to the camera is detected by a few steps. Imagine a reference frame in the perspective of camera and optical centre.

The second part of the problem is attempted by the same method first thresholding the image then using good points to track to find the corners then applying perspective transform to straighten the images and then blurring them to find the colours of the points in the right place. Those points or group of points are fixed because we are mapping the points into a fixed image size. Then calculating if those points are black are white by simple methods by this we will have binary data which then could be transformed into the marker ID but the implementation failed due to the incorrect corners being detected by the code and then the image is transformed into absolute randomness, it could be solved by choosing the correct contours but will require further research.

ObjectPoints A vector of vector of 3D points. The outer vector contains as many elements as the number of the pattern views. ImagePoints A vector of vectors of the 2D image points. ImageSize Size of the image CameraMatrix Intrinsic camera matrix DistCoeffs Lens distortion coefficients. These coefficients will be explained in a future post. vecs Rotation specified as a 3×1 vector. The direction of the vector specifies the axis of rotation and the magnitude of the vector specifies the angle of rotation. tvecs 3×1 Translation vector.

VI. Results and Observation

Explain the trend of results in details. Mention the drawbacks (if any) of your algo compared to other algo and the reason of picking up the approach over the other if you have implemented any algo over the other. The results are quite clear and at par with the other data available. Slightly better results could be expected if better methods are used for camera calibration. Maybe an extra dynamic thresholding could make the processing much faster.

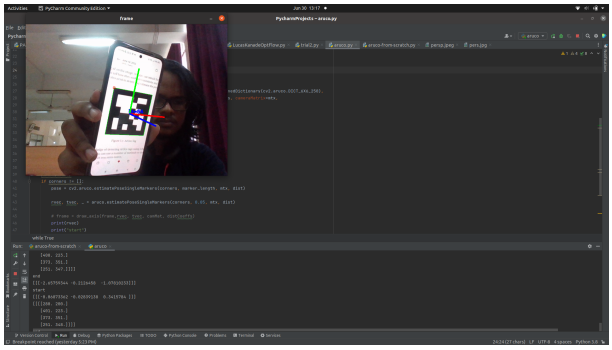

Fig. 2. Working of the first part

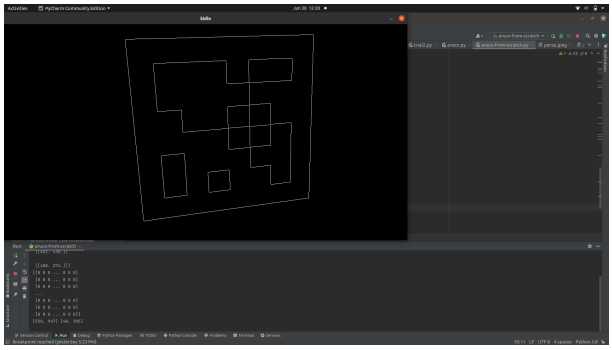

Fig. 3. Second Part

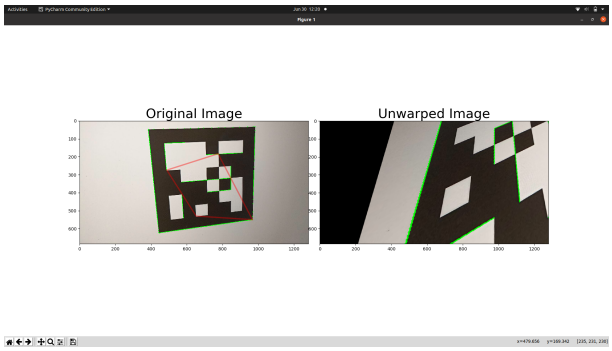

Fig. 4. Second Part

VII. Future Work

Te second part of the code has to be refined in order for it to work, probably including a contour will do the trick. Else the project can be expanded into tracking the relative distance or relative orientation between two aruco tags, this could be very useful for synchronized controlling of multiple ariel vehicles. This data could be extensively used for real life projects.

Conclusion

The problem wanted is to detect ID and pose of aruco and in the second part to detect it from scratch. The difficulties faced was the lack of opportunity to discuss the problem with seniors and batch mates as people are not present at campus. The project is very useful as Fiducial markers are very reliable and can be used for various applications.

References

[1] Strisciuglio, Nicola Vallina, Maria Petkov, Nicolai Muñoz-Salinas, Rafael. (2018). Camera Localization in Outdoor Garden Environments Using Artificial Landmarks. 1-6. 10.1109/IWOBI.2018.8464139.