Exploration of Lightweight Processors and Vision Stack for Quadrotors: ArUco Marker Detection

Independent Project (BIP398)

Monsoon Semester 2023

Submitted by: **Dharani Kumar .S**
Undergraduate, Dept. of Computer Science and Engineering, IIIT Delhi

Faculty Advisor: **Dr. Sayan Basu Roy**
Assistant Professor, ECE Dept., IIIT Delhi

New Delhi
December 2023

Abstract

ArUco markers have become ubiquitous in the field of robotics for their simplicity and robustness in providing spatial orientation and identification. The detection and decoding of these markers, traditionally reliant on pre-built libraries, presents an opportunity for exploration into a scratch implementation. This work focuses on building a step-by-step framework for ArUco marker detection using Python without the use of specialized libraries such as OpenCV's ArUco module.

1 Introduction

Given the broader problem of landing a drone autonomously, one of the initial milestones would be target identification, i.e. detecting the candidate landing spot. This can be achieved with the help of ArUco markers. ArUco markers are 2-dimensional barcodes designed to be an efficient and portable solution for pose estimation. Despite the availability of comprehensive libraries such as OpenCV, there exists functional value in developing a rudimentary implementation from scratch. Such an endeavor not only highlights the underlying mechanics of marker detection but also allows us to optimize the pipeline for constrained environments where library dependencies are a bottleneck.

2 Methodology

The proposed implementation is comprised of the following steps: image preprocessing, potential marker detection, corner refinement, and marker decoding. Each step leverages basic image processing techniques facilitated by the core functionality of OpenCV, yet refrains from utilizing its ArUco module.

2.1 Image Preprocessing

The initial phase of preprocessing is pivotal for enhancing the subsequent detection processes. This phase consists of three primary actions applied to the input image:

Grayscaling converts the image from RGB to shades of gray, resulting in reduced computational complexity by collapsing the three color channels into a single luminance channel. This is sufficient for the purposes of marker detection where color information is not critical.

Blurring applied to the grayscale image using a Gaussian filter. The choice of Gaussian blurring is due to its efficacy in reducing image noise and smoothing out variations while preserving edge properties. By mitigating noise, we reduce the likelihood of false contour detection in later stages.

Thresholding is the final step in preprocessing, where the blurred image is converted into a binary image. The binary image creation process involves assigning pixels to either a high value (white) if they are above a certain threshold or a low value (black) if below. This binary segmentation accentuates the markers against the background, allowing for their isolation and identification in the contour detection phase.

2.2 Potential Marker Detection

The second step involves identifying contours within the thresholded image that may correspond to ArUco markers. Each detected contour is approximated to a polygon to reduce the complexity of its shape. The approximation process is aimed at transforming irregular contours into quadrilaterals, as ArUco markers are square in shape. Only those contours that approximate to four-sided polygons are considered potential markers. Furthermore, an area-based filtering is applied where only those quadrilaterals whose area falls within predefined thresholds are kept for further processing. This filter ensures that irrelevant contours or noise that may have been detected as potential markers are excluded based on the expected size of the ArUco markers. In this work, 4X4, 5X5, and 6X6 markers were studied.

2.3 Corner Refinement

The corner refinement process improves the precision of detected ArUco marker corners to sub-pixel accuracy, an important step for applications demanding high positional fidelity. Post initial detection, corner points—often quantized to pixel resolution are inadequate for precise pose estimation. Refinement step uses the intensity gradient within a localized window around each corner, iteratively shifting the corner estimate towards the gradient's weighted centroid. When these changes fall below a minimal threshold, we get coordinates with floating-point precision. The refined corners ensure a more accurate perspective transformation, which is vital in the studied use case - to identify the pose of the drone relative to the ArUco marker placed at the landing spot. Minute discrepancies in pose estimation can lead to significant error in the drone's motion control step, hence necessitating the corner refinement step.

2.4 Marker Decoding

The final step in the detection pipeline is the decoding of the identified markers. This involves transforming the skewed perspective of the quadrilateral marker into a top-down, frontal view using perspective transformation techniques. Once the marker is in this canonical view, it is analyzed on a grid basis to interpret the binary code it encodes. Each cell within the grid is evaluated to determine whether it is part of the black or white regions of the ArUco marker. The binary pattern obtained from this grid analysis corresponds to the marker's unique identifier, completing the detection and decoding process.

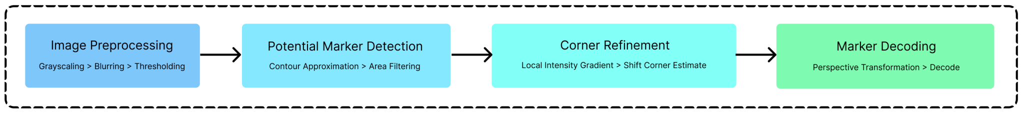
Image Preprocessing
Grayscaling > Blurring > Thresholding
Potential Marker Detection
Contour Approximation > Area Filtering
Corner Refinement
Local Intensity Gradient > Shift Corner Estimate
Marker Decoding
Perspective Transformation > Decode

Figure 1: ArUco Marker Detection Pipeline

3 Conclusion

The pipeline explored in this work was implemented on Raspberry Pi 4, one of the candidate on-board computers, using input from Arducam 64MP Auto-focus camera. Promising results were observed for still images obtained from the camera, however an implementation on live video feed failed due to an existing unsolved boot configuration issue on some Raspberry Pi 4 boards. The above bare-bone algorithm will have significantly lesser memory overhead compared to OpenCV's ArUco marker detection library (which has been discontinued).

Future endeavors will concentrate on addressing the video feed processing limitations and refining the algorithm for enhanced real-time performance. In addition, other alternatives to Raspberry Pi offering better computational power and reduced processing speed will be studied. A significant milestone of this project-to make a customizable vision stack, has hopefully been met, paving way for studying its performance in other candidate on-board computers.

The repository for this project can be found here