Orain Ferguson

Intro to Robotics (ELEE 4280)

Final Project Literary Review/Project Writing

4/15/23

Table of Contents

I. Introduction

The OpenCV (Open-Source Computer Vision Library) ArUco marker detection is a popular and widely used framework for marker-based computer vision applications. This library allows users to use computer vision technique to detect and track the location and orientation of a specific pattern or marker in an image or video. Markers are typically created by combining different shapes, colors, and patterns to create a unique and easily identifiable feature. This means that each marker has its own ID number. The detection process involves detecting the marker's boundaries and identifying its specific pattern, enabling the calculation of the marker's position and orientation relative to the camera.

ArUco marker detection is a widely used technique in computer vision that provides an accurate and efficient way to track and localize objects in 2D and 3D space. ArUco markers are square-shaped fiducial markers that are designed to be easily detectable and recognizable in various lighting conditions and environments. The ArUco marker detection algorithm uses image processing to detect, identify, and estimate the position and orientation of the ArUco markers in an image or video stream. The detection algorithm is robust, fast, and can handle multiple markers in the same image or video frame.

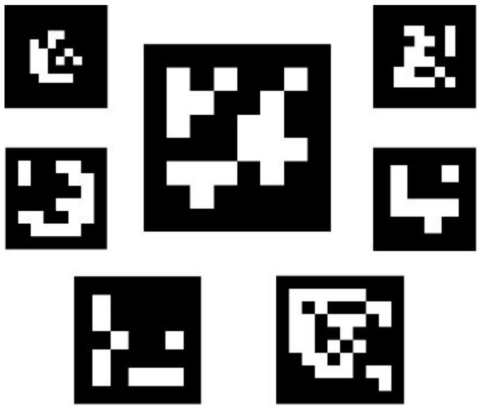


Figure 1 : [2]

The advantages of using ArUco marker detection include its simplicity, versatility, and accuracy. ArUco markers can be easily printed on a variety of surfaces, including paper, plastic, and metal, making them ideal for a wide range of applications. They can be detected and tracked in real-time, allowing for dynamic and interactive applications such as augmented reality and robotics. Furthermore, the use of ArUco markers does not require prior knowledge of the environment, making them suitable for use in a wide range of lighting and environmental conditions. This makes ArUco marker detection a reliable and robust method for tracking and localization in various applications such as medical imaging, object tracking, and robotics.

In this paper, we will provide an overview of the ArUco marker detection technique, its algorithm, and its advantages. We will also provide implementation examples of ArUco marker detection using the OpenCV library, one of the most popular open-source computer vision libraries. Then we describe its applications in various fields and evaluate its performance through experiments and case studies. The aim of this paper is to provide an in-depth understanding of ArUco marker detection and its potential for various computer vision applications.

II. Features and properties of ArUco markers

ArUco markers are square-shaped fiducial markers that are designed to be easily detectable and recognizable in various lighting conditions and environments. They are made up of a black and white checkerboard pattern with a unique identification number embedded within the marker. The properties and features of ArUco markers include:

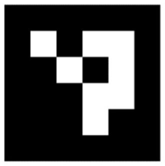
1. Unique Identification: Each ArUco marker has a unique identification number embedded in the marker. This identification number enables the system to identify and differentiate

between multiple markers in the same image or video frame. These identification numbers can be set to represent different meanings as need be.

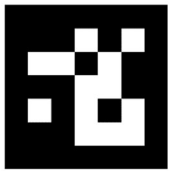
2. Checkerboard pattern: The checkerboard pattern is used to provide contrast and make the marker easily detectable. The black and white squares provide a clear boundary for the marker, making it easy for the detection algorithm to identify the marker's position and orientation.
3. Size: The size of the ArUco marker can vary depending on the application. In general, the larger the marker, the easier it is to detect and track. However, the size of the marker must be balanced against the size and resolution of the camera used to detect the marker.
4. Orientation: ArUco markers can be detected and tracked in 2D, and 3D space. The orientation of the marker can be estimated using the pose estimation algorithm, which calculates the marker's position and orientation in the camera frame.
5. Robustness: ArUco markers are designed to be robust and can be detected in various lighting conditions and environments. They can be used in both indoor and outdoor environments and can withstand changes in lighting conditions such as shadows, reflections, and variations in brightness.
6. Easy to print: ArUco markers can be easily printed on various surfaces such as paper, plastic, and metal. This makes them cost-effective and easy to use in a wide range of applications.

The unique identification, checkerboard pattern, size, orientation, robustness, and ease of printing are some of the key features and properties of ArUco markers that make them suitable for a wide range of computer vision applications. ArUco markers are unique in that they are designed with a specific dictionary that defines the marker's pattern and identification number. The dictionary defines the number of bits used to encode the marker, the size of the marker, and the number of markers in the dictionary. There are five mainstream dictionaries that are used today.

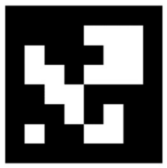
1. DICT_4X4: This is the smallest dictionary of ArUco markers, containing 50 markers. Each marker is 4x4 pixels in size and is encoded using 16 bits. This dictionary is useful in applications where markers need to be placed in tight spaces or where marker size is limited.



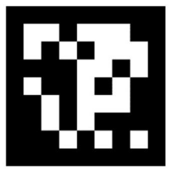
2. DICT_5X5: This dictionary contains 250 markers, with each marker being 5x5 pixels in size and encoded using 25 bits. This dictionary is commonly used in robotics and automation applications, where larger markers are required for better detection and tracking.



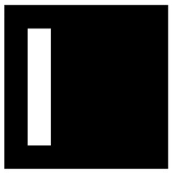
3. DICT_6X6: This dictionary contains 1000 markers, with each marker being 6x6 pixels in size and encoded using 36 bits. This dictionary is useful in applications where high accuracy and precision are required, such as in medical imaging and navigation.



4. DICT_7X7: This dictionary contains 4000 markers, with each marker being 7x7 pixels in size and encoded using 49 bits. This dictionary is commonly used in augmented reality and gaming applications, where larger markers are required to enhance the user experience.



5. DICT_ARUCO_ORIGINAL: This is the original dictionary of ArUco markers, containing 1024 markers. Each marker is 5x5 pixels in size and encoded using 25 bits. This dictionary is commonly used in research and development applications, where a large number of unique markers are required. Note: This marker counts like a binary counter going up the marker from the bottom right to left.



Overall, the different dictionaries of ArUco markers provide a wide range of options for various applications. The choice of dictionary depends on the specific requirements of the application, such as marker size, accuracy, and the number of unique markers required. The ability to customize the dictionary also allows for the creation of unique markers tailored to specific applications.

III. ArUco Marker Detection Applications

ArUco marker detection has a wide range of applications in various fields, including robotics, automation, and augmented reality. It enables the calculation of the position and orientation of an object relative to the camera, which is essential.

In the robotics and automation applications ArUco maker has grants accuracy, speed, and simplicity. The ability to detect and track markers in real-time is critical for robots to perform tasks such as object tracking, localization, and navigation. Allowing autonomous navigation systems to provide precise localization information, meaning robots can navigate and interact with their environment. “Unmanned quadrotors are likely to become an important vehicle in humans' daily life. However, their automatic navigation and landing in indoor environments are among the commonly discussed topics in this regard. In fact, the quadrotor should be able to automatically find the landing point from the nearby position, navigate toward it, and finally, land on it accurately and smoothly” [1]. This is just one example of how it can be used.

In augmented reality, marker detection is used to superimpose virtual objects onto real-world images Figure 2. It enables the creation of interactive and immersive experiences where virtual objects can be placed and manipulated in real-time in a physical environment.

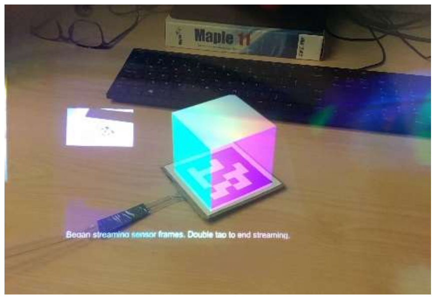


Figure 2 [2].

This overlay in education applications is used to teach and learn concepts such as geometry, physics, and astronomy. ArUco markers can be used to create interactive simulations and models, where students can explore and interact with virtual objects in a real-world context. In advertising applications, ArUco markers are used to create interactive product experiences. For example, a company can attach ArUco markers to its products, and customers can use a smartphone app to scan the markers and see additional information and content about the product.

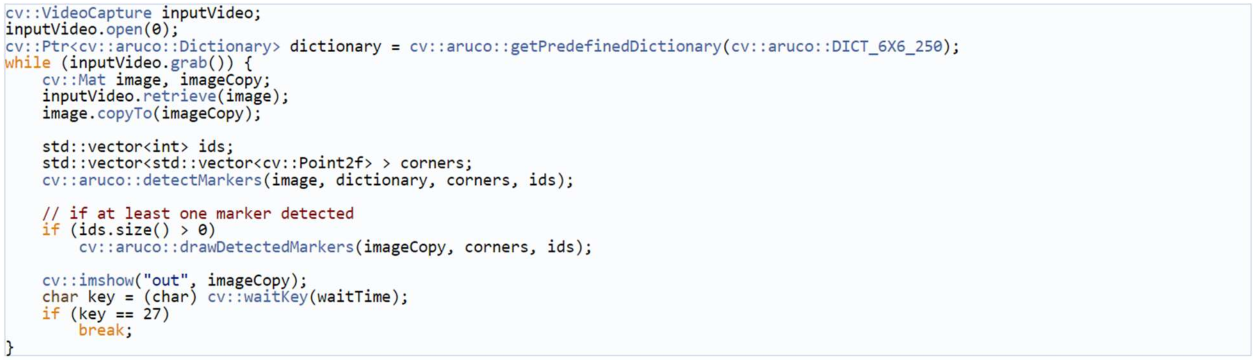
Overall, ArUco marker detection is an essential tool in computer vision applications, providing a reliable and accurate method for identifying and tracking objects or patterns. It enables the development of advanced applications that require precise localization and orientation information, improving efficiency, accuracy, and safety in various fields.

III. ArUco Marker Detection Algorithm

The ArUco marker detection algorithm is a computer vision algorithm that is used to detect and identify ArUco markers in images or video streams. The algorithm consists of several steps, which we will discuss in detail below:

1. Image Acquisition: The first step in the ArUco marker detection algorithm is to acquire an image or video frame of the scene. This can be done using a camera or other imaging device.
2. Camera Calibration: Before detecting ArUco markers, the camera needs to be calibrated to correct for lens distortion and other imaging parameters. Camera calibration involves capturing images of a known calibration pattern and using these images to calculate the camera's intrinsic and extrinsic parameters. In most classes the camera manufacturers should provide these numbers. There are also programs/scripts that you can run to get exact numbers if you would like.
3. Image Preprocessing: Once the camera is calibrated, the next step is to preprocess the image to enhance the features of the ArUco markers. This may involve converting the image to grayscale, applying filters to reduce noise, and thresholding to binarize the image. In often cases a Kalman filter is applied.
4. Marker Detection: The core of the ArUco marker detection algorithm is the detection of the ArUco markers in the preprocessed image. This is done by searching for rectangular regions in the image that match the size and shape of ArUco markers. Once a potential marker is identified, it is further analyzed to confirm whether it is an actual ArUco marker or a false positive.
5. Marker Identification: Once the ArUco markers are detected, they need to be identified to determine their IDs and their positions in 3D space. Each ArUco marker has a unique ID that can be decoded using a lookup table. The position of the ArUco marker can be calculated using the camera calibration parameters and the perspective projection equation.
6. Pose Estimation: Once the position of the ArUco markers is known, the pose of the camera relative to the markers can be estimated. This involves solving a set of equations that relate the 3D position of the markers to their 2D image coordinates.
7. Visualization: Finally, the detected ArUco markers and their poses can be visualized in the image or video stream. This may involve drawing a bounding box around the markers, displaying the marker IDs, or overlaying virtual content on top of the markers.

Overall, the ArUco marker detection algorithm is a complex process that involves several steps of image processing, marker detection, identification, and pose estimation. The algorithm relies on the use of camera calibration and mathematical equations to accurately detect and identify ArUco markers in real-time. This is where OpenCV comes in as it already has premade scripts you can use to do the calculation. The only thing you need to feed the functions is the image, corners, what dictionary you want to use, and the ids you're looking for.



```
cv::VideoCapture inputVideo;
inputVideo.open(0);
cv::Ptr<cv::aruco::Dictionary> dictionary = cv::aruco::getPredefinedDictionary(cv::aruco::DICT_6X6_250);
while (inputVideo.grab()) {
    cv::Mat image, imageCopy;
    inputVideo.retrieve(image);
    image.copyTo(imageCopy);

    std::vector<int> ids;
    std::vector<std::vector<cv::Point2f> > corners;
    cv::aruco::detectMarkers(image, dictionary, corners, ids);

    // if at least one marker detected
    if (ids.size() > 0)
        cv::aruco::drawDetectedMarkers(imageCopy, corners, ids);

    cv::imshow("out", imageCopy);
    char key = (char) cv::waitKey(waitTime);
    if (key == 27)
        break;
}
```

Figure 3 : Basic script in C for video capture and to detect markers from our camera. [2]

Steps 4 through 6 have many steps in between that I'll go a little more in dept with. In step 4 there are two sub-steps, the first being detection of marker candidates. "In this step the image is analyzed to find square shapes that are candidates to be markers. It begins with an adaptive thresholding to segment the markers, then contours are extracted from the thresholded image and those that are not convex or do not approximate to a square shape are discarded. Some extra filtering are also applied" [2]. The filtering process is removing candidates that are too large, too small, too close to each other, or any other way that would affect the reading of the code from the pool of candidates. Then after all the possible candidates are detected it "it is necessary to determine if they are markers by analyzing their inner codification. This step starts by extracting the marker bits of each marker. To do so, first, perspective transformation is applied to obtain the marker in its canonical form. Then, the canonical image is thresholded using Otsu theorem to separate white and black bits. The image is divided into different cells according to the marker size and the border size and the amount of black or white pixels on each cell is counted to determine if it is a white or a black bit. Finally, the bits are analyzed to determine if the marker belongs to the specific dictionary and error correction techniques are employed when necessary" [2].

Step 5 is the actual detection which is the most important part of the process. The OpenCV library makes use of the detectMarkers() function to handle all of the calculations. The functions parameters are:

- The image where the markers are going to be detected.
- The dictionary of code you are using.
- The detected markerCorners as well as the markerIds. These are stored as structures that contain:
  - markerCorners is the list of corners of the detected markers. Each marker has its four corners going from top left clockwise to bottom left.
  - markerIds is the list of IDs of each of the detected markers in the markerCorners structure.

- The object of type DetectionParameters. This object includes all the parameters that could be edited/changed to your specifications during the detection process.
- Then there are the rejectedCandidates. This is a list of marker candidates that squares that could have been found but they don't present a valid codification. This is why it's crucial to make sure you're using the correct library for the markers you wish to detect. Note: This parameter is optional.

The detectMarkers() method first applies various preprocessing and filtering techniques to the input image to enhance the features of the ArUco markers and reduce noise. It then uses the specified dictionary of ArUco markers to detect and identify the markers in the image. The filtering method used is thresholding. An example of this is the photos below.

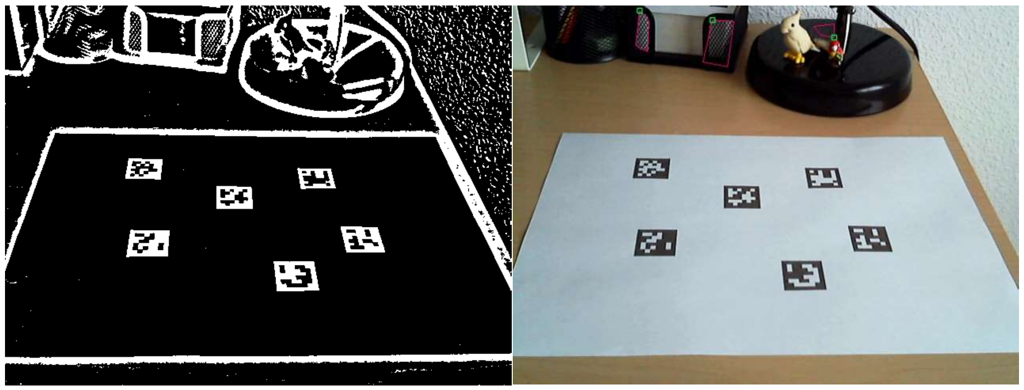


Figure 4 [2]

Thresholding is a common image processing technique used in ArUco marker detection to separate the foreground (i.e., the markers) from the background of the input image. Thresholding is used to convert a grayscale or color image into a binary image, where each pixel is either black or white. Having the pixels be either white or black also helps to reduce noise in the image. Once thresholding is applied, the resulting binary image can be further processed to remove noise and apply bit extraction.

After removing the perspective distortion and applying a filter to the image, the next step in the ArUco marker detection pipeline is to extract the marker's ID and other information encoded in its binary pattern. This process is called "bits extraction". Each ArUco marker has a binary pattern encoded in its interior. The number of cells in the grid depends on the marker's dictionary and the desired level of accuracy. Each cell in the grid is either black or white, representing a binary value of 0 or 1. This is an example of the image obtained after removing the perspective distortion of a marker:

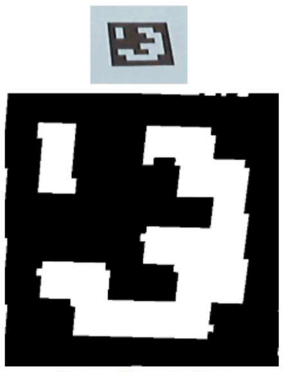


Figure 5 [2]

Once the grid pattern is placed, the bits extraction process can begin. This involves dividing the grid into a fixed number of rows and columns and computing the average intensity value of each cell. Based on the average intensity values, the binary pattern of the marker can be reconstructed, and its ID can be decoded.

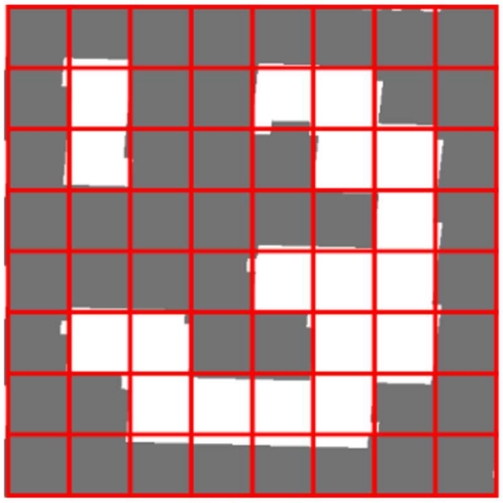


Figure 6 [2]

Its not recommended to count all the cell pixel because there might be some error in the perspective distortion. To solve this, we have a margin in the center or the cell that is used in the intensity calculations.

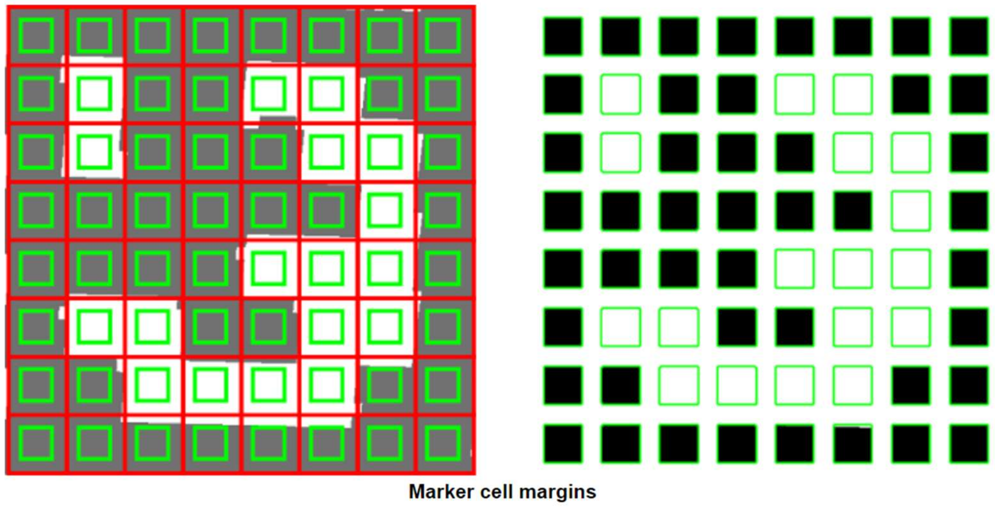


Figure 6 [2]

The bits extraction process also allows for the extraction of other information encoded in the marker's binary pattern, such as its orientation and scale. This information can be used in conjunction with pose estimation techniques to determine the marker's position and orientation in 3D space.

Step 6 is Pose Estimation meaning a computer vision technique that is used to estimate the position and orientation of an object in 3D space relative to a camera or observer. In the context of ArUco marker detection, pose estimation is used to determine the position and orientation of the ArUco markers in the camera's coordinate system. The pose estimation process involves solving a set of equations that relate the 3D position of the markers to their 2D image coordinates. There are several techniques for pose estimation, but one of the most common approaches is to use a process called perspective-n-point (PnP) estimation. This is the same method that OpenCV uses. To perform pose estimation in relation to the camera you need to know the calibration parameters of your camera or camera's intrinsic and extrinsic parameters. The camera intrinsic matrix is a 3x3 elements matrix with focal length and principal point. The extrinsic parameters define the position and orientation of the camera in the world coordinate system.

IV. Experimental Evaluation / Literary Review Results

Automatic Navigations of UAV Landing

Several papers seem to talk about using the idea of markers in UAVs for all kinds of different purposes. Some are using them inside and others outside. Judging from the number of these papers the demand for a UAV landing at a defined point with accuracy is very high. These drones need to do this while being safe and without human intervention. By using marker pose estimation the UAV or Quadcopter can determine the height of flight and touchdown with high accuracy or allow them to navigate. The landing of an aircraft is also important because if something lands too harshly it could damage the internals. We need these UAV to land as softly as possible. According to [3] the most popular landing aid system is the ILS (Instrument Landing System) developed in the first half of the 20th century. There are many perks to this method, but a large downside is the need for special airfield infrastructure. This isn't something the industry wants to do for every location a UAV might want to land. This is where the idea of markers comes in and having the UAV see a marker and then know where to land based on that.

There are two ways to “locate and track the moving object in the environment, there are two classes of vision-based techniques, namely the markerless and the marker-based approaches” [4]. Markerless uses colors, patterns, and other features to find the target they are looking for. This means there is no prior knowledge of the size or arrangement of the pattern. This does come without a downside. Computational speed is greatly hindered due to feature extraction and matching [5,6]. Marker use allows for pose information to be acquired at a greater speed than markerless. ArUco marks seem to be a very popular choice for researchers [1,3,4,7]. It is worth noting that many researchers [1,3,4,7] use the same method that was described in part III of our paper, which involves utilizing OpenCV functions to perform various calculations. This approach is quite popular in the field of computer vision due to the wide range of tools and functions available in OpenCV, which makes it easier for researchers to perform their calculations. In most cases, researchers will modify or augment the OpenCV functions with additional steps to further refine their research goals. For example, the Otsu algorithm [8] is used in paper [1] which is the process of automatic image thresholding involves determining an optimal threshold level that separates an image into two classes of pixels, namely foreground and background.

Separating the foreground and background can be difficult at long range and this is what [4] trying to do. They are proposing an algorithm that could be used in these UAVs to see ArUco markers from higher up. To improve pose estimation, they are using Levenberg-Marquardt algorithm for optimization [9]. The Levenberg-Marquardt algorithm is used “to solve nonlinear least squares problems. Least squares problems arise in the context of fitting a parameterized mathematical model to a set of data points by minimizing an objective expressed as the sum of the squares of the errors between the model function and a set of data points”[10]. Using this method, the algorithm was improved by a distance of 44%.

Another solution to this was in [7] where they embedded a smaller ArUco code inside a larger one Figure 7. By using two markers with different IDs, the issue of detecting a single marker throughout all of the landing stages is resolved. Without the smaller inner code, if the UAV sees the larger marker, it will initiate its descent based on that marker alone. However, when the marker gets too close to the camera, some corners of the marker might be cut out of frame, which could result in misdetection. The smaller inner code solves this problem by providing additional markers for the UAV to detect, which increases the likelihood of successful detection and tracking. By utilizing multiple markers with different IDs, the system is more robust and less prone to errors. Figure 8 is the Minimum and maximum detection ranges for original ArUco and embedded ArUco markers.

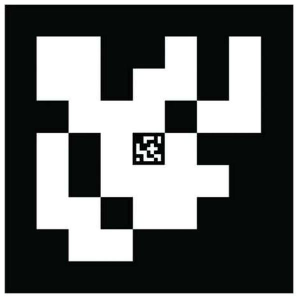
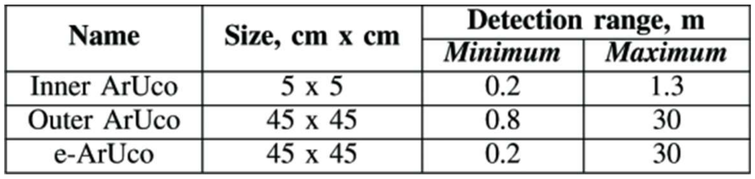


Figure 7 [7]



Name	Size, cm x cm	Detection range, m	
		Minimum	*Maximum*
Inner ArUco	5 x 5	0.2	1.3
Outer ArUco	45 x 45	0.8	30
e-ArUco	45 x 45	0.2	30

Figure 8 [7]

ArUco markers don't only have to be used on UAVs that fly, they can also be under water in AUV (autonomous underwater vehicle). "The conventional underwater navigation methods are achieved by acoustic equipment, such as the ultra-short-baseline localization systems and Doppler velocity logs, etc - suffer from low fresh rate, low bandwidth, environmental disturbance and high cost" [11]. The proposal is to develop an underwater visual navigation system based on the use of multiple ArUco markers. Xing developed a new multi sensor for indoor localization system based on the ArUco markers for UAVs. The sensor fusion includes markers, optical flow, ultrasonic and the inertial sensor. The researchers chose to use the method of Extended Kalman Filter (EFK) plus ArUco markers which has yielded satisfactory results [12]. Below in Figure 9 you can see the results of [12]. Here they compared the results of their EKF algorithm to a standard odometer for localization with the help of ArUco markers.

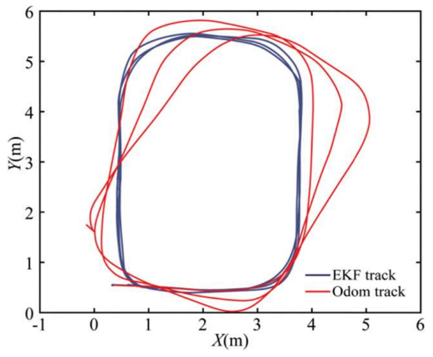


Figure 9 [12]

This camera and sensor need to be calibrated so that it can not only make up for distortion issues but also the reflection effects that are caused by the water. To solve these issues the method of [13] was used which does Single Viewpoint Omnidirectional Camera Calibration from Planar Grids. Formula 1.

$$E = \frac{1}{2} \sum_{i=1}^{m} \left[H\left(\Theta, g_i\right) - p_i \right]^2,$$

Formula 1 [11]

Where a grid based on a chessboard consists of *m* points of g_i coordinates with corresponding values p_i. The term *H* is a function mapping grid points into the image plane. The Θ involves intrinsic, distortion, and extrinsic parameters. Here they are also using the Levenberg–Marquardt approach to minimize the cost 1 of solving for Θ. In Figure 10 below we can see the chessboard being used to calculate the lens.

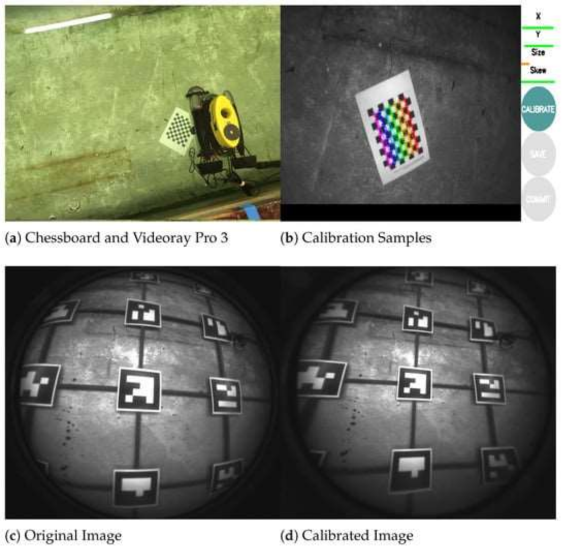


Figure 10 [11]

The camera calibration is done using OpenCV functions that are heavily based on [14]. These bassists are all self-adaptive recurrent neuro-fuzzy controls of an AUV. “Without a priori knowledge, the recurrent neuro-fuzzy system is first trained to model the inverse dynamics of the AUV and then utilized as a feedforward controller to compute the nominal torque of the AUV along a desired trajectory. ”[14]. The results of the paper state that sometime when run the detection algorithm may recognize the marker incorrectly. This happens when the marker is far from the camera or is only partially visible. Because of this, the trajectories generated by the marker navigation system are not consistent. In most cases this wasn’t the case and deviations between the marker position and the estimated position were less than 0.5 m at most. The average deviations were about 0.2 m and with this the researchers found acceptable. It is important to note that this experiment was done in a towing tank under controlled conditions with clear water and no currents. The results obtained in such conditions may not necessarily be representative of the performance of the system in more challenging environments, such as murky water or water with currents or waves. In more challenging conditions, the visual navigation system may face difficulties in accurately detecting and localizing the ArUco markers, which could lead to errors in the estimation of the vehicle's position and orientation.

Another hard environment for UAVs to navigate in is GNSS-denied environments. A GNSS (Global Navigation Satellite System) denied environment is an area or location where the signals transmitted by the GNSS systems are either not available or are unreliable due to various factors. These factors may include intentional or unintentional interference or jamming of the signals, or physical obstructions that block or weaken the signal, such as tall buildings, natural terrain, or atmospheric conditions. This means that they usually give up the connection between the map frame and the world frame. This is where [15] is trying to improve this by developing a Global ArUco-Based Lidar Navigation System for UAV.

In this paper it’s worth noting that the working conditions of the UAV are in a dry coal shed of a thermal power plant. The main function of this is to correctly total the amount of coal in and out of the plant within a two-to-three-day period. This must be done by a UAV because of safety concerns for humans. The current solution is to use UWB (Ultra-Wide Band) positioning [16].

The issue with this is there are large blind areas, the maintenance is complex, high hardware cost, and easily effected by dust and metals in environments. The paper researcher is wanting to connect an ArUco markers to certain points in the coal shed to which then the UAV can use lidar to use a navigation each maker and then check against the marker placed there. The results of the paper found that the named ArUco-LIO has high precision and could reliably navigate UAVS into known GNSS denied environments.

Improving ArUco makers with Filtering

There is one issue with ArUco markers and that is they tend to have a lot of noise. The common solution to this is the use of Kalman filters with ArUco markers [17]. Kalman filter is a recursive filter that estimates the next state of a process in discrete time by comparing it to prior measured values. Among various types of filters, the EKF (Extended Kalman Filter) is more commonly used with ArUco markers, as in [18]. The EKF is used to correct the position and orientation of the robot by comparing the viewing angle and the estimate to each datum. The approach presented in [18] is similar to the one described in part II of this paper, with the addition of solving the Extended Kalman filter equations to predict the position and orientation and then applying a correction phase if the prediction was inaccurate. Its results eliminate much of the noise that is created when trying to “look” for a marker.

ArUco in Augmented Reality

An important problem that Computer vision has is connecting the virtual world with the real one. This is where ArUco markers can come in to play. In most cases ArUco markers are in the physical world but in this study [18] they are using artificial markers. In this study they are planning on giving a solution to Simultaneous Localization and Mapping (SLAM) [19] or Visual SLAM when you are solving by extracting data points from 2D images [20] that utilize artificial markers. Estimating the 3D structure of a scene captured by 2D images can be a very daunting and complex task. How researchers in [18] are accomplishing this is by relying on multiple artificial makers scattered in the environment. They then plan to use [21,22] from the Augmented reality libraries to measure the position and orientation of each artificial marker. This will then automate the process of visual mapping and localization into two stages.

The first stage of the process involves capturing images of the environment with the artificial markers in view. These images are then processed to determine the relative positions and orientations of each marker with respect to the others. This information is used to create a map of the network of markers, which is aligned with a global reference frame. In the second stage, images from the camera are analyzed in conjunction with the map of markers to determine the position and orientation of the camera in the environment. This process involves identifying the markers in the image and using their known positions and orientations from the map to compute the camera's position and orientation relative to the markers. This allows for accurate localization of the camera in the environment.

To evaluate their work the researchers were made to take measurements then comparing actual positions of real-world markers and the computed positions of artificial markers obtained through the proposed process. The tests demonstrate that artificial markers provide an accurate solution for 3D reconstruction in commonly encountered scenarios.

ArUco vs Other Markers

There are many different kinds of fiducial markers out there. ArUco isn’t the only one there are also AprilTags, ARTags, STag, and more. In [23] a study was conducted to test these different tags to give the community the experimental data. First off what are AprilTags, ARTags, and STags. First let’s talk about ApirlTags [24] work almost the same as ArUco but with a few differences in the since that their functions have fewer tuning parameters, work well at longer range, but have more false detections. These tags work more like a 2D barcode than the ArUco’s chessboard style. Next is ARTags they are “a marker system that uses digital coding theory to get a very low false positive and inter-marker confusion rate with a small, required marker size, employing an edge linking method to give robust lighting variation immunity” [25]. These too look very similar to ArUco marker. They are what AprilTags are based off of. Finally, STags are “ designed to be robust against jitter factors, thus sustaining pose stability better than the existing solutions. This is achieved by utilizing geometric features that can be localized more repeatably.” [26]. It’s of note that all these tags were used in the same ROS implementations for all the algorithms “out-of-the-box” without making any hardware specific optimizations.

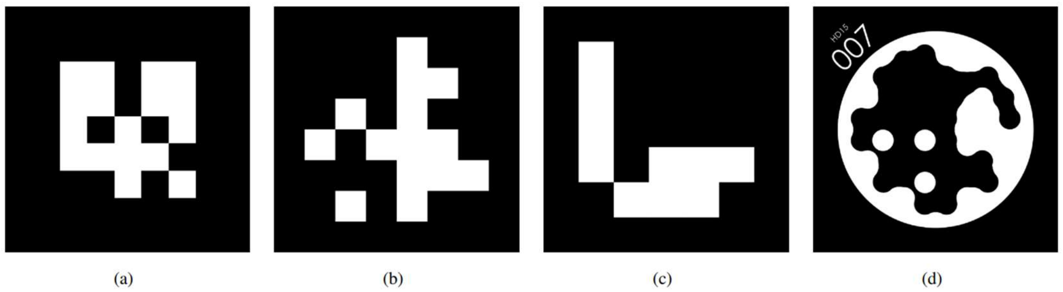
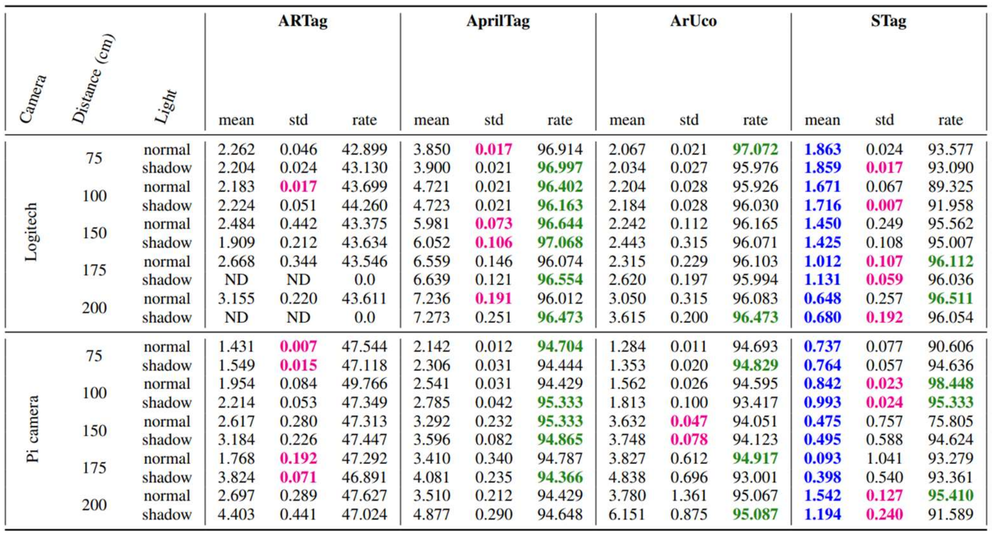


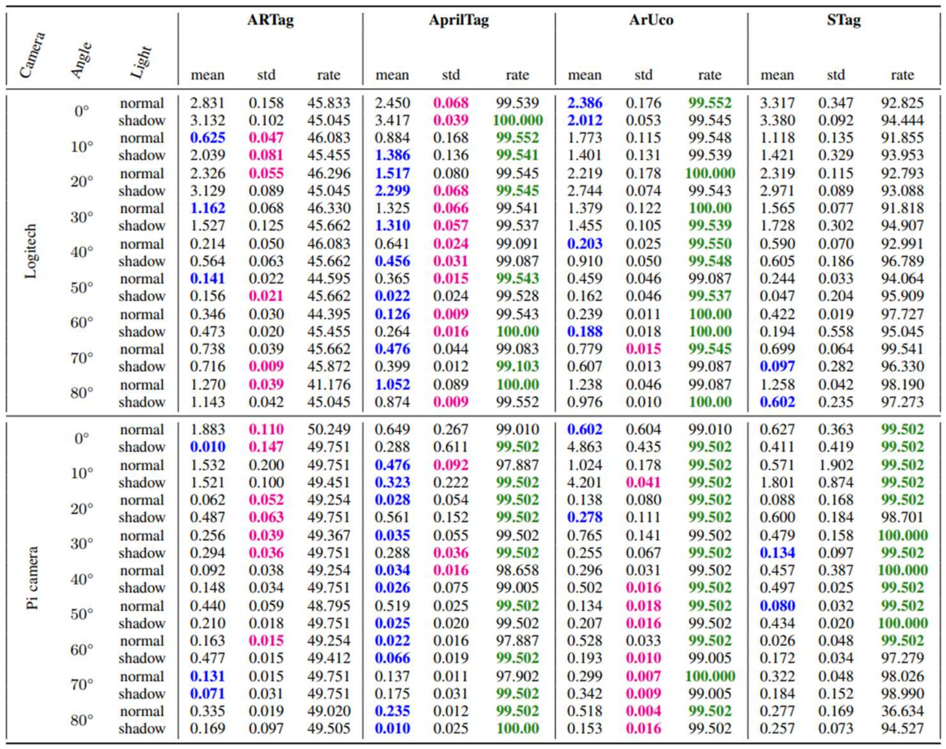
Figure 11: (a) ARTag (b) AprilTag, (c) ArUco, and (d) Stag. Photo [26]

In this paper they talk about some of the uses that each of these markers is being used so they can recreate some of the environmental. In [27] ArUco was used for localization in GNSS restricted environments. In [28] SLAM was used with ARTags. In [29] fiducial markers were used at different inclined platforms to calculate the relative position and orientation of the platform so that a UAV could land on it. This required the researchers to construct tests environments similar to real would uses cases. To accomplish this, they used two different cameras, a Raspberry Pi camera, and a Logitech camera, to capture images from various angles (0-80 degrees) and distances (between 75cm and 200cm) for calculations. These experiments were necessary to evaluate the algorithms' performance and ensure their suitability for practical applications. The following two Figures were the results.



Camera	Distance (cm)	Light	ARTag mean	ARTag std	ARTag rate	AprilTag mean	AprilTag std	AprilTag rate	ArUco mean	ArUco std	ArUco rate	STag mean	STag std	STag rate
Logitech	75	normal	2.262	0.046	42.899	3.850	**0.017**	96.914	2.067	0.021	**97.072**	**1.863**	0.024	93.577
		shadow	2.204	0.024	43.130	3.900	0.021	**96.997**	2.034	0.027	95.976	**1.859**	**0.017**	93.090
	100	normal	2.183	**0.017**	43.699	4.721	0.021	**96.402**	2.204	0.028	95.926	**1.671**	0.067	89.325
		shadow	2.224	0.051	44.260	4.723	0.021	**96.163**	2.184	0.028	96.030	**1.716**	**0.007**	91.958
	150	normal	2.484	0.442	43.375	5.981	**0.073**	**96.644**	2.242	0.112	96.165	**1.450**	0.249	95.562
		shadow	1.909	0.212	43.634	6.052	**0.106**	**97.068**	2.443	0.315	96.071	**1.425**	0.108	95.007
	175	normal	2.668	0.344	43.546	6.559	0.146	96.074	2.315	0.229	96.103	**1.012**	**0.107**	**96.112**
		shadow	ND	ND	0.0	6.639	0.121	**96.554**	2.620	0.197	95.994	**1.131**	**0.059**	96.036
	200	normal	3.155	0.220	43.611	7.236	**0.191**	96.012	3.050	0.315	96.083	**0.648**	0.257	**96.511**
		shadow	ND	ND	0.0	7.273	0.251	**96.473**	3.615	0.200	**96.473**	**0.680**	**0.192**	96.054
Pi camera	75	normal	1.431	**0.007**	47.544	2.142	0.012	**94.704**	1.284	0.011	94.693	**0.737**	0.077	90.606
		shadow	1.549	**0.015**	47.118	2.306	0.031	94.444	1.353	0.020	**94.829**	**0.764**	0.057	94.636
	100	normal	1.954	0.084	49.766	2.541	0.031	94.429	1.562	0.026	94.595	**0.842**	**0.023**	**98.448**
		shadow	2.214	0.053	47.349	2.785	0.042	**95.333**	1.813	0.100	93.417	**0.993**	**0.024**	**95.333**
	150	normal	2.617	0.280	47.313	3.292	0.232	**95.333**	3.632	**0.047**	94.051	**0.475**	0.757	75.805
		shadow	3.184	0.226	47.447	3.596	0.082	**94.865**	3.748	**0.078**	94.123	**0.495**	0.588	94.624
	175	normal	1.768	**0.192**	47.292	3.410	0.340	94.787	3.827	0.612	**94.917**	**0.093**	1.041	93.279
		shadow	3.824	**0.071**	46.891	4.081	0.235	**94.366**	4.838	0.696	93.001	**0.398**	0.540	93.361
	200	normal	2.697	0.289	47.627	3.510	0.212	94.429	3.780	1.361	95.067	**1.542**	**0.127**	**95.410**
		shadow	4.403	0.441	47.024	4.877	0.290	94.648	6.151	0.875	**95.087**	**1.194**	**0.240**	91.589

Figure 12 [26]



Camera	Angle	Light	ARTag mean	ARTag std	ARTag rate	AprilTag mean	AprilTag std	AprilTag rate	ArUco mean	ArUco std	ArUco rate	STag mean	STag std	STag rate
Logitech	0°	normal	2.831	0.158	45.833	2.450	**0.068**	99.539	**2.386**	0.176	**99.552**	3.317	0.347	92.825
		shadow	3.132	0.102	45.045	3.417	**0.039**	**100.000**	**2.012**	0.053	99.545	3.380	0.092	94.444
	10°	normal	**0.625**	**0.047**	46.083	0.884	0.168	**99.552**	1.773	0.115	99.548	1.118	0.135	91.855
		shadow	2.039	**0.081**	45.455	**1.386**	0.136	**99.541**	1.401	0.131	99.539	1.421	0.329	93.953
	20°	normal	2.326	**0.055**	46.296	**1.517**	0.080	99.545	2.219	0.178	**100.000**	2.319	0.115	92.793
		shadow	3.129	0.089	45.045	**2.299**	**0.068**	**99.545**	2.744	0.074	99.543	2.971	0.089	93.088
	30°	normal	**1.162**	0.068	46.330	1.325	**0.066**	99.541	1.379	0.122	**100.00**	1.565	0.077	91.818
		shadow	1.527	0.125	45.662	**1.310**	**0.057**	99.537	1.455	0.105	**99.539**	1.728	0.302	94.907
	40°	normal	0.214	0.050	46.083	0.641	**0.024**	99.091	**0.203**	0.025	**99.550**	0.590	0.070	92.991
		shadow	0.564	0.063	45.662	**0.456**	**0.031**	99.087	0.910	0.050	**99.548**	0.605	0.186	96.789
	50°	normal	**0.141**	0.022	44.595	0.365	**0.015**	**99.543**	0.459	0.046	99.087	0.244	0.033	94.064
		shadow	0.156	**0.021**	45.662	**0.022**	0.024	99.528	0.162	0.046	**99.537**	0.047	0.204	95.909
	60°	normal	0.346	0.030	44.395	**0.126**	**0.009**	99.543	0.239	0.011	**100.00**	0.422	0.019	97.727
		shadow	0.473	0.020	45.455	0.264	**0.016**	**100.00**	**0.188**	0.018	**100.00**	0.194	0.558	95.045
	70°	normal	0.738	0.039	45.662	**0.476**	0.044	99.083	0.779	**0.015**	**99.545**	0.699	0.064	99.541
		shadow	0.716	**0.009**	45.872	0.399	0.012	**99.103**	0.607	0.013	99.087	**0.097**	0.282	96.330
	80°	normal	1.270	**0.039**	41.176	**1.052**	0.089	**100.00**	1.238	0.046	99.087	1.258	0.042	98.190
		shadow	1.143	0.042	45.045	0.874	**0.009**	99.552	0.976	0.010	**100.00**	**0.602**	0.235	97.273
Pi camera	0°	normal	1.883	**0.110**	50.249	0.649	0.267	99.010	**0.602**	0.604	99.010	0.627	0.363	**99.502**
		shadow	**0.010**	**0.147**	49.751	0.288	0.611	**99.502**	4.863	0.435	**99.502**	0.411	0.419	**99.502**
	10°	normal	1.532	0.200	49.751	**0.476**	**0.092**	97.887	1.024	0.178	**99.502**	0.571	1.902	**99.502**
		shadow	1.521	0.100	49.451	**0.323**	0.222	**99.502**	4.201	**0.041**	**99.502**	1.801	0.874	**99.502**
	20°	normal	0.062	**0.052**	49.254	**0.028**	0.054	**99.502**	0.138	0.080	**99.502**	0.088	0.168	**99.502**
		shadow	0.487	**0.063**	49.751	0.561	0.152	**99.502**	**0.278**	0.111	**99.502**	0.600	0.184	98.701
	30°	normal	0.256	**0.039**	49.367	**0.035**	0.055	99.502	0.765	0.141	99.502	0.479	0.158	**100.000**
		shadow	0.294	**0.036**	49.751	0.288	**0.036**	**99.502**	0.255	0.067	**99.502**	**0.134**	0.097	**99.502**
	40°	normal	0.092	0.038	49.254	**0.034**	**0.016**	98.658	0.296	0.031	99.502	0.457	0.387	**100.000**
		shadow	0.148	0.034	49.751	**0.026**	0.075	99.005	0.502	**0.016**	**99.502**	0.497	0.025	**99.502**
	50°	normal	0.440	0.059	48.795	0.519	0.025	**99.502**	0.134	**0.018**	**99.502**	**0.080**	0.032	**99.502**
		shadow	0.210	0.018	49.751	**0.025**	0.020	99.502	0.207	**0.016**	99.502	0.434	0.020	**100.000**
	60°	normal	0.163	**0.015**	49.254	**0.022**	0.016	97.887	0.528	0.033	**99.502**	0.026	0.048	**99.502**
		shadow	0.477	0.015	49.412	**0.066**	0.019	**99.502**	0.193	**0.010**	99.005	0.172	0.034	97.279
	70°	normal	**0.131**	0.015	49.751	0.137	0.011	97.902	0.299	**0.007**	**100.000**	0.322	0.048	98.026
		shadow	**0.071**	0.031	49.751	0.175	0.031	**99.502**	0.342	**0.009**	99.005	0.184	0.152	98.990
	80°	normal	0.335	0.019	49.020	**0.235**	0.012	**99.502**	0.518	**0.004**	**99.502**	0.277	0.169	36.634
		shadow	0.169	0.097	49.505	**0.010**	0.025	**100.00**	0.153	**0.016**	99.502	0.257	0.073	94.527

Figure 13 [26]

The results of this study reveal that AprilTag, ArUco, and STag have high detection rates in almost all tested environments. STag performed best in position measurements, while AprilTag showed the best results in orientation measurements. Nevertheless, ArUco consistently ranked second across both measurements. Although STag was initially expected to be the most stable, AprilTag and ARTag demonstrated similar stability.

There is another kind of marker that has an attribute that really makes it stand out from among the rest. The ChromaTag [30] (Figure 14) is a fiducial marker and detection algorithm made to use opponent colors. They state it quickly rejects initial false detections and grayscale for precise localization.

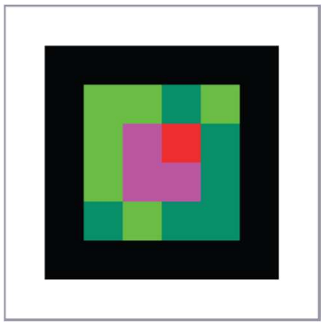


Figure 14 [30]

How it works is " ChromaTag uses red to green borders because they have a large image gradient in the A space. ChromaTag uses two shades of red and two shades of green as representations of 0 and 1 to embed a code in the tag. These different shades of red and green are only differentiable in the B channel, so detection is done in the A channel and decoding is done in the B channel. RGB values for the different red and green colors are shown" [30] see Figure 15 below.

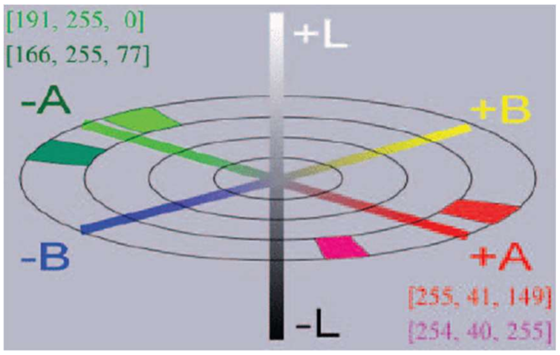


Figure 15 [30]

The used detection algorithm is very extensive and in depth if you would like to read about it, please find it in the sources. The results of the comparison against normal black and white fiducial markers. "We demonstrate on thousands of real images that ChromaTag achieves detection speeds significantly faster than the current state of the art while maintaining similar detection accuracy. We also show that ChromaTag detection fails at far distances and steep viewing angles and recommend AprilTag as a better option for applications that require detection in these conditions. Lastly, we provide evidence that ChromaTag detection is robust to color variation and break down which steps of the detection algorithm take the most time and fail most often" [30].

V. Conclusion

In conclusion, ArUco marker detection using the OpenCV library is a popular and widely used technique in computer vision for tracking and localizing objects in 2D and 3D space. ArUco markers are square-shaped fiducial markers that can be easily detected and recognized in various lighting conditions and environments. The advantages of using ArUco marker detection include its simplicity, versatility, and accuracy though there are better options depending on the application. The detection algorithm is robust, fast, and can handle multiple markers in the same image or video frame. Through experiments and case studies, it has been shown that ArUco marker detection is an accurate and efficient solution for 3D reconstruction of commonly encountered scenarios. Future works may involve testing ArUco markers in larger and more complex scenarios, and also in situations of calibration and tracking in real-time. Overall, ArUco marker detection is a reliable and robust method for tracking and localization in various computer vision applications.

VI. Citations

[1] M. F. Sani and G. Karimian, "Automatic navigation and landing of an indoor AR. drone quadrotor using ArUco marker and inertial sensors," 2017 International Conference on Computer and Drone Applications (IConDA), Kuching, Malaysia, 2017, pp. 102-107, doi: 10.1109/ICONDA.2017.8270408.

[2] doxygen. "Detection of ARUCO Markers." *OpenCV*, 26 July 2019, https://docs.opencv.org/4.1.1/d5/dae/tutorial_aruco_detection.html.

[3] A. Marut, K. Wojtowicz and K. Falkowski, "ArUco markers pose estimation in UAV landing aid system," 2019 IEEE 5th International Workshop on Metrology for AeroSpace (MetroAeroSpace), Turin, Italy, 2019, pp. 261-266, doi: 10.1109/MetroAeroSpace.2019.8869572.

[4] Ferrão, José, et al. "Detection of ARUCO Markers Using the Quadrilateral Sum Conjuncture." *SpringerLink*, Springer International Publishing, 1 Jan. 1970, https://link.springer.com/chapter/10.1007/978-3-319-93000-8_41.

[5] Heung-Jun, Kim, and Lee Yong-Hwan. "Evaluation of Feature Extraction and Matching Algorithms for the Use of Mobile Application." *Journal of the Semiconductor & Display Technology*, The Korean Society Of Semiconductor & Display Technology, 25 Nov. 2015, https://koreascience.kr/article/JAKO201506363291307.page.

[6] El-gayar, M M, et al. "A Comparative Study of Image Low Level Feature Extraction Algorithms." *Egyptian Informatics Journal*, Elsevier, 1 Aug. 2013, https://www.sciencedirect.com/science/article/pii/S1110866513000248.

[7]. A. Khazetdinov, A. Zakiev, T. Tsoy, M. Svinin and E. Magid, "Embedded ArUco: a novel approach for high precision UAV landing," 2021 International Siberian Conference on Control and Communications (SIBCON), Kazan, Russia, 2021, pp. 1-6, doi: 10.1109/SIBCON50419.2021.9438855.

[8] N. Otsu, "A threshold selection method from gray-level histograms", *Automatica*, vol. 11, pp. 23-27, 1975.

[9] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters", *Journal of the Society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431-441, 1963.

[10] Gavin, Henri P. *The Levenberg-Marquardt Algorithm for Nonlinear Least Squares Curve ...* Duke University, 27 Nov. 2022, https://people.duke.edu/~hpgavin/ExperimentalSystems/lm.pdf.

[11] Xu, Z.; Haroutunian, M.; Murphy, A.J.; Neasham, J.; Norman, R. An Underwater Visual Navigation Method Based on Multiple ArUco Markers. *J. Mar. Sci. Eng.* **2021**, *9*, 1432. https://doi.org/10.3390/jmse9121432

[12] J. Zheng, S. Bi, B. Cao and D. Yang, "Visual Localization of Inspection Robot Using Extended Kalman Filter and Aruco Markers," 2018 IEEE International Conference on Robotics and Biomimetics (ROBIO), Kuala Lumpur, Malaysia, 2018, pp. 742-747, doi: 10.1109/ROBIO.2018.8664777.

[13] C. Mei and P. Rives, "Single View Point Omnidirectional Camera Calibration from Planar Grids," Proceedings 2007 IEEE International Conference on Robotics and Automation, Rome, Italy, 2007, pp. 3945-3950, doi: 10.1109/ROBOT.2007.364084.

[14] Jeen-Shing Wang and C. S. G. Lee, "Self-adaptive recurrent neuro-fuzzy control of an autonomous underwater vehicle," in IEEE Transactions on Robotics and Automation, vol. 19, no. 2, pp. 283-295, April 2003, doi: 10.1109/TRA.2003.808865.

[15] Qiu, Z.; Lin, D.; Jin, R.; Lv, J.; Zheng, Z. A Global ArUco-Based Lidar Navigation System for UAV Navigation in GNSS-Denied Environments. *Aerospace* **2022**, *9*, 456. https://doi.org/10.3390/aerospace9080456

[16] Zahran, S, et al. "MICRO-RADAR AND UWB AIDED UAV NAVIGATION IN GNSS DENIED ENVIRONMENT." *Google Scholar*, Sensors to Methods and Applications, 10 Oct. 2018, https://noa.gwlb.de/servlets/MCRFileNodeServlet/cop_derivate_00004482/isprs-archives-XLII-1-469-2018.pdf.

[17] H. C. Kam, Y. K. Yu and K. H. Wong, "An Improvement on ArUco Marker for Pose Tracking Using Kalman Filter," *2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, Busan, Korea (South), 2018, pp. 65-69, doi: 10.1109/SNPD.2018.8441049.

[18] R. S. Xavier, B. M. F. da Silva and L. M. G. Goncalves, "Accuracy Analysis of Augmented Reality Markers for Visual Mapping and Localization," 2017 Workshop of Computer Vision (WVC), Natal, Brazil, 2017, pp. 73-77, doi: 10.1109/WVC.2017.00020.

[19] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I", *IEEE Robotics and Automation Magazine*, vol. 13, no. 2, pp. 99-110, 2006.

[20] F. Fraundorfer and D. Scaramuzza, "Visual odometry: Part II: Matching robustness optimization and applications", *IEEE Robotics and Automation Magazine*, vol. 19, no. 2, pp. 78-90, 2012.

[21] S. Garrido-Jurado, R. M. Noz Salinas, F. Madrid-Cuevas and M. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion", *Pattern Recognition*, vol. 47, no. 6, pp. 2280-2292, 2014.

[22] R. Muñoz-Salinas, M. J. Marín-Jiménez, E. Yeguas-Bolivar and R. M. Carnicer, "Mapping and localization from planar markers", *Arxiv Computing Research Repository (CoRR)*, vol. abs/1606.00151, 2016.

[23] Kalaitzakis, Michail, et al. "Experimental Comparison of Fiducial Markers for Pose Estimation." *University of South Carolina*, http://www.me.sc.edu/Research/USRL/publications/conferences/ICUAS2020_Kalaitzakis.pdf.

[24] E. Olson, "AprilTag: A robust and flexible visual fiducial system," 2011 IEEE International Conference on Robotics and Automation, Shanghai, China, 2011, pp. 3400-3407, doi: 10.1109/ICRA.2011.5979561.

[25] M. Fiala, “ARTag, a Fiducial Marker System Using Digital Techniques,” in 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05), vol. 2. IEEE, 2005, pp. 590–596

[26] Benligiray, Burak, et al. “STag: A Stable Fiducial Marker System.” *Image and Vision Computing*, Elsevier, 10 July 2019, https://www.sciencedirect.com/science/article/pii/S0262885619300903.

[27] J. Bacik, F. Durovsky, P. Fedor, and D. Perdukova, “Autonomous flying with quadrocopter using fuzzy control and ArUco markers,” Intelligent Service Robotics, vol. 10, no. 3, pp. 185–194, July 2017.

[28] H. Lim and Y. S. Lee, “Real-Time Single Camera SLAM Using Fiducial Markers.” Fukuoka, Japan: IEEE, 2009, p. 177.

[29] P. Vlantis, P. Marantos, C. P. Bechlioulis, and K. J. Kyriakopoulos, “Quadrotor landing on an inclined platform of a moving ground vehicle,” in 2015 IEEE International Conference on Robotics and Automation (ICRA). Seattle, WA, USA: IEEE, May 2015, pp. 2202– 2207.

[30] Degol, Joseph, et al. “ChromaTag: A Colored Marker and Fast Detection Algorithm.” *University of Illinois Urbana-Champaign*, Institute of Electrical and Electronics Engineers Inc., 22 Dec. 2017, https://experts.illinois.edu/en/publications/chromatag-a-colored-marker-and-fast-detection-algorithm.