

Vision-Augmented Object Detection and Pose Estimation Using ArUco Markers for Real-Time Industrial Applications

Fatimah Rahman¹, Shamir Nadeem^{1*}, Muhammad Owais¹, Nasir Rahman Jadoon¹, Muhammad Ali², Rizwan Mufti³

¹ Department of Electrical Engineering, Pakistan Institute of Engineering and Applied Sciences, Islamabad, Pakistan
Email: fatimahr2304@gmail.com, shamirnadeem098@gmail.com, owaisabbasi9822@gmail.com, jadoon@pieas.edu.pk

² Department of Physics and Mathematics, Pakistan Institute of Engineering and Applied Sciences, Islamabad, Pakistan
Email: muhammadali@pieas.edu.pk

³ Department of Mechanical Engineering, Pakistan Institute of Engineering and Applied Sciences, Islamabad, Pakistan
Email: mufti@pieas.edu.pk

*Corresponding author: shamirnadeem098@gmail.com

Received: 13-08-2025, Received in Revised form: 07-10-2025, Accepted: 13-12-2026, Published: 31-12-2025

Abstract

This paper presents a vision-based system for object detection and pose estimation in industrial settings. The combination of ArUco markers and computer vision enables the system to provide accurate localization and real-time orientation estimation of objects, which applies to automated assembly lines, warehouse inventory deployment, pick-and-place industrial robots, and quality control procedures. ArUco markers serve as fiducial references, enabling reliable spatial analysis: measurement of object dimensions (height and width), centroid location, horizontal and vertical distances and orientation. These capabilities enable accurate manipulation and monitoring of industrial robots, making operations more efficient and less reliant on manual intervention. The experimental results demonstrate the system's accuracy and its responsiveness to relevant events, which can be leveraged to improve the precision, reliability, and automation of various industrial applications. A quantitative comparison of processing latency, throughput, and memory consumption against a deep learning baseline is also reported, confirming that the proposed pipeline runs in real time on CPU-only hardware. This work emphasizes that integrating computer vision methods and fiducial markers can be highly efficient for industrial automation and operational improvement.

Keywords: Object Detection, Pose Estimation, ArUco Markers, Computer Vision, Robotic Vision, Centroid Calculation, Spatial Relationships, Shape-Based Object Recognition, Robotic Manipulation, Automation

Introduction

Computer vision enables robots to recognize object shapes and estimate their poses for navigation and manipulation; therefore, it has numerous applications in robotics. By employing ArUco markers, the system provides more effective object tracking and greater precision in spatial orientation. This study proposes combining object detection with ArUco markers to maximize robotic systems' perception. The system performs object identification and size estimation, while ArUco markers significantly improve detection reliability and robotic spatial awareness, making them essential tools. Industrial applications include integration into automated assembly lines, warehouses, packaging systems, and quality inspection stations. In such settings, robotic systems often need to execute tasks that require accurate object localization, alignment, and tracking to manipulate parts, tools, or packages. By using computer vision, robots can recognize objects and their shapes, determine their dimensions, and operate in more complex environments that include conveyor belts, storage racks, or assembly fixtures.

The system enables the visual inspection of structural components, such as casings, machine parts, and product assemblies, to identify misalignments, defects, and wear, thereby facilitating preventive maintenance and quality control. These markers can also be used in indoor industrial environments where GPS is not available and manual measurements are unreliable. This system gives robots a clear perception of their environment and allows them to perform tasks such as part orientation, quality inspection, and automated material handling. This method improves the accuracy, efficiency, and safety of robotic systems across a variety of industrial applications by enhancing how robots perceive and interact with their environment.

Literature Review

A. Object Detection Techniques

Object detection is crucial because it enables robots to understand their surroundings. Traditional methods, such as shape and pattern recognition, include notable contributions like Suzuki and Abe's line detection technique [1]. However, these techniques can struggle under non-uniform lighting conditions and obscured or altered objects. Modern approaches utilize machine learning and deep learning, with CNN-based methods such as Faster R-CNN [2] and YOLO [3] for highly accurate real-time performance. These systems learn to identify objects by analyzing datasets, allowing them to adapt effectively to different environments. Integrating the center of gravity of moving objects enables robots to identify items and adjust their positions, thereby reducing human intervention and enhancing pick-and-place precision in industrial robotics [4]. A study by Kumar et al. introduces a six-DOF articulated robotic manipulator equipped with a 3D vision system that employs a camera and computer vision algorithms to detect object deformation and measure distances for precise manipulation tasks [5]. Later advancements show that CNN-based vision grasping features (Convolutional Neural Network) utilize oriented anchor boxes to improve the accuracy and efficiency of grasp angle prediction [6]. Additionally, combining YOLOv4 with a particle filter enables highly accurate real-time object detection and reliable tracking in dynamic environments [3]. Moreover, integrating multiple cameras and sensors with an EKF (Extended Kalman Filter) improves resilience to occlusions and external disturbances, thereby enhancing robustness in visual servoing [7]. Current methods are either not robust in constrained conditions or require computationally intensive resources that are not always suitable for practical real-time industrial applications. Our approach incurs minimal computational overhead while maintaining sufficient accuracy and flexibility for use in industrial robotic systems. Moreover,

our classical computer vision technique uses fiducial ArUco markers to ensure robust and cost-effective detection.

B. Localization and Pose Estimation in Robotics

Determining the position and orientation of an object in space is known as pose estimation, an essential aspect of robotic systems in tasks such as handling objects and moving through their surroundings with accuracy and precision. Traditional pose estimation techniques rely on geometric relationships, such as triangulation and Perspective-n-Point algorithms [8]. These strategies establish correspondence between 2D image points and 3D world points to determine the object's location. Recent advancements in deep learning have introduced data-driven methods for pose estimation, such as PoseNet [9], which applies CNNs to directly compute object position and orientation. These methods can offer high precision. However, they require extensive training, creating significant computational overhead and making them unsuitable for fast tasks in robotics. A low-cost, sensor-free robotic arm system has also been developed using semi-supervised learning, which integrates labeled synthetic data and unlabeled real data to train the pose estimation module. This approach reduces the need for extensive labeled datasets and leverages geometric constraints for domain adaptation [10]. ArUco markers, as discussed by Garrido-Jurado et al. [11], are commonly used to estimate pose effectively in applications such as robotics, particularly for accurately estimating an object's position relative to the camera without requiring complex deep learning systems, in sectors that demand immediate and precise localization. Another solution focuses on a vision-based SCARA robot system designed to pick and place tools in factory environments. This system employs image calibration, recognition, and localization to identify objects accurately within a defined workspace [12]. Existing pose estimation methods are either too computationally demanding for real-time applications or require large labeled datasets for training, which prevents their use in constrained and dynamic industrial settings. Our system provides an effective and low-cost alternative for pose estimation by integrating ArUco markers. It does not require extensive training data or specialized sensors to obtain accurate localization and orientation, making it well suited for industrial applications.

C. Fiducial Markers in Robotic Systems

Fiducial markers such as ArUco and AprilTags provide precise real-time localization and pose estimation while maintaining a low computational cost. They are particularly well suited for vision-based control systems [13] and are optimized for real-time applications like robotic navigation [14]. However, these markers can be affected by non-uniform lighting [15] and may perform inconsistently in underwater and reflective settings, with AprilTags generally offering superior detection capabilities [16]. Motion blur can hinder detection, which can be mitigated through filtering and sensor fusion techniques [17]. The use of thermal or depth cameras can enhance detection in low-visibility conditions [17], and employing multi-scale strategies can achieve frame rates exceeding 1000 fps [14]. Additionally, machine learning models trained on synthetic

data can improve robustness [15], and 3D-shaped ArUco markers can enhance detection across various angles and distances [18]. The Optimized Marker Placement (OMP) framework also improves localization accuracy in complex environments [19]. Although fiducial markers perform effectively, performance degrades significantly in unstructured or harsh conditions, and recent advances usually require additional hardware or complex filtering algorithms. By applying image preprocessing techniques (e.g., gamma correction and adaptive thresholding) and simplified but efficient spatial transformation algorithms, our system improves the reliability of ArUco marker-based localization. This improves marker detection in low-light environments and reduces reliance on supplementary sensors. In addition, our method estimates object size and orientation based on markers, which is essential for robotic manipulation in industrial applications.

Vision System

A. Setup

This section describes the hardware components, software tools, and ArUco marker setup.

1) Hardware (Camera Specifications)

Our vision system uses a standard 480p webcam for affordability, ease of integration, and sufficient resolution for object detection and localization in controlled industrial environments. The camera was mounted in one of two configurations: An eye-to-hand configuration or a fixed-mount configuration. Table 1 lists the camera specifications.

Table 1. Camera Specifications

Specifications	Value
Resolution	640 x 480 (480p)
Frame Rate	30 FPS
Interface	USB 2.0
Lens	Fixed Focus

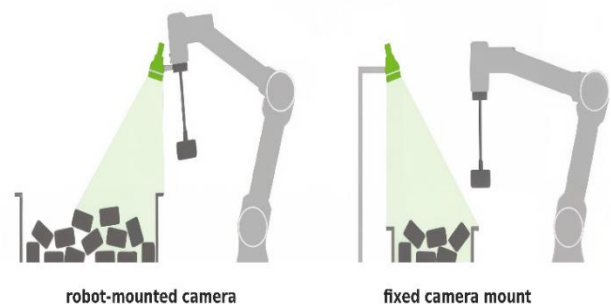


Figure 1. Camera mounting configurations: (left) eye-to-hand configuration with the camera mounted on the manipulator, and (right) fixed-mount camera observing the workspace from a static overhead position.

2) Software

The vision system uses OpenCV for image processing and ArUco marker detection. Python's OpenCV library, which provides a wide range of tools for image processing, computer vision, and machine learning. We used Visual Studio for software development and code implementation.

3) ArUco Marker

ArUco markers serve as reference points for coordinate transformations, facilitating accurate localization. Various Python libraries support ArUco marker detection, each offering different dictionaries and customization options. Prominent among these is OpenCV's ArUco module (exposed in code as `cv2.aruco`), which provides extensive functionality for marker detection, pose estimation, and dictionary management. ArUco markers are available in different dictionary types, each varying in marker size and number of unique markers, such as `DICT_4X4_50`, `DICT_5X5_100`, and `DICT_6X6_250`, allowing users to select the appropriate dictionary based on their application's requirements for marker robustness and detection speed. Other libraries, such as ArUco by Raul Mur-Artal, offer additional features and optimizations tailored to specific applications.

B. Workflow

- **Image Acquisition:** An image of the workspace is captured using a camera at regular time intervals.
- **Image Nondistortion:** Lens distortions are corrected using precomputed calibration parameters.
- **Image Preprocessing:** Region of interest selection, gamma correction, Gaussian blurring, grayscale conversion, color inversion, and adaptive thresholding are applied sequentially to condition the image for contour extraction.
- **Shape-Based Detection:** Contours are extracted and classified by vertex count into circles, rectangles, or triangles, and are then filtered based on area and aspect ratio.
- **Localization and Orientation:** The centroid of each detected object is referenced to the ArUco marker frame to determine its position in millimeters and its orientation in degrees.

C. Camera Calibration

Camera calibration is the process of determining a camera's intrinsic parameters (focal lengths, optical center, and skew) and extrinsic parameters (rotation and translation relative to the world frame) to enable accurate 3D position estimation. Using known patterns like a chessboard and functions such as `cv2.calibrateCamera()`, the calibration process computes the intrinsic matrix $\mathbf{K}$, which captures the internal camera properties, and extrinsic matrix $[\mathbf{R} \mid \mathbf{t}]$, which defines the camera's orientation and position in space. The projection from 3D world coordinates to 2D image coordinates is modeled by the equation:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{K} [\mathbf{R} \mid \mathbf{t}] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (1)$$

where (u, v) are the image-plane coordinates in pixels, (X, Y, Z) are the corresponding world coordinates, and s is a non-zero scale factor. The intrinsic matrix is given by:

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (2)$$

where f_x and f_y are the focal lengths in pixel units along the horizontal and vertical axes, respectively, and (c_x, c_y) is the principal point.

Lens Distortion Correction: Camera calibration also estimates distortion coefficients to correct lens distortions such as barrel, pincushion, and mustache distortions, which can significantly affect the accuracy of 3D measurements if left uncorrected. The commonly used distortion model includes radial (k_1, k_2, k_3) and tangential (p_1, p_2) distortion coefficients:

$$\mathbf{d} = [k_1 \ k_2 \ p_1 \ p_2 \ k_3] \quad (3)$$

The calibration process eliminates these distortions, producing undistorted images that provide accurate real-world measurements and coordinate transformations.

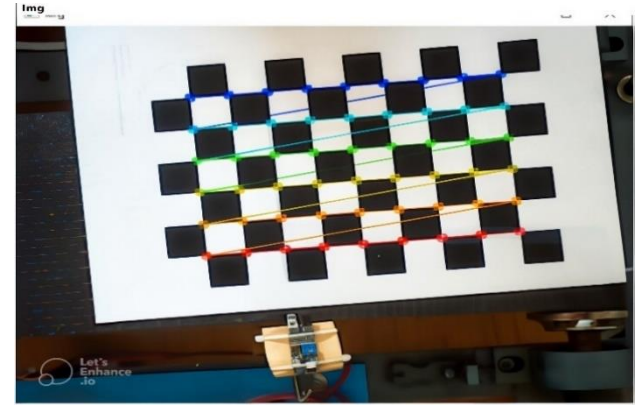


Figure 2. Chessboard pattern used for camera calibration, with detected corner correspondences overlaid.

D. Image Nondistortion

To ensure accurate object shape detection, images are processed to correct lens distortion using OpenCV. Initially, an optimal camera matrix is calculated with `cv2.getOptimalNewCameraMatrix()` to reduce border distortion by removing unwanted pixels. Next, `cv2.undistort()` is applied to eliminate lens-induced warping by utilizing the camera matrix and distortion coefficients. Finally, the image is cropped to the region of interest (ROI), removing any black borders and ensuring accurate object detection.

E. Preprocessing

The object detection algorithm operates against a homogeneous background, enhancing detection accuracy and reducing false positives.

The preprocessing steps prepare the image for effective object detection.

Region of Interest (ROI): Focuses processing on specific areas of the image, improving efficiency by ignoring irrelevant regions. Using an ROI simplifies and speeds up image processing, reducing noise and unnecessary data.

Gamma Correction (Exposure Adjustment): Adjusts image brightness and contrast by transforming pixel intensity values. This compensates for lighting variations and enhances visibility. The human eye perceives brightness nonlinearly, so gamma correction helps make

images appear more natural and balanced by correcting for this non-linearity. The gamma correction formula is:

$$I_{out} = 255 * \left(\frac{I_{in}}{255}\right)^\gamma \quad (4)$$

where I_{in} and I_{out} are the input and output pixel intensities and γ is the gamma exponent. In OpenCV, gamma correction can be implemented using a custom function such as `gamma_correction()`.

Gaussian Blur: Reduces noise and smooths the image, aiding in texture removal. It works by applying a weighted average of pixel values in the neighborhood of each pixel, creating a blurred effect. The formula for the Gaussian function used in Gaussian blur is:

$$G(x, y) = \frac{1}{2\pi\sigma^2} * e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (5)$$

where x and y are the horizontal and vertical distances from the kernel origin, respectively, and σ is the standard deviation of the Gaussian kernel. In OpenCV, Gaussian blur can be applied using the `cv2.GaussianBlur()` function.

Grayscale Conversion: Each pixel's color information is reduced to a single intensity value, simplifying the image while preserving its essential structure. It reduces the complexity of the image data, making it easier to process by reducing computations. Color information can often be unnecessary for tasks like edge detection or feature extraction. In OpenCV, an image can be converted to grayscale using the `cv2.cvtColor()` function.

Color Inversion: Inverts the image colors, replacing each pixel's color with its complementary color. This helps highlight features by enhancing contrast between objects and the background. In OpenCV, color inversion can be achieved using the `cv2.bitwise_not()` function.

Adaptive Thresholding: Adaptive thresholding calculates a threshold for smaller image regions, allowing it to handle varying lighting conditions more effectively. It is useful for images with uneven illumination, and it provides better segmentation of objects from the background. In OpenCV, adaptive thresholding can be performed using the `cv2.adaptiveThreshold()` function.

F. Shape-Based Object Detection

Object detection was accomplished using contour detection using the OpenCV function `cv2.findContours()` by utilizing the `cv2.RETR_EXTERNAL` parameter; only the outer contours were extracted. Meanwhile, `cv2.CHAIN_APPROX_SIMPLE` was used to reduce the number of points in the contour representation to boost processing performance. Objects were detected based on their shape. Each object was classified as a circle, rectangle, or triangle depending on the number of vertices identified after contour detection. `cv2.approxPolyDP()` function approximates a contour with the closest polygonal shape having fewer vertices. Finally, contours were filtered based on size and shape criteria. This step uses area and aspect ratio filters to isolate only the objects of interest (e.g., rectangular shapes) while ignoring irrelevant objects.



Figure 3. Object Detection of Standard-shaped Objects.

G. Localization and Orientation

This project focuses on object localization through a custom vision algorithm that analyzes the centroid data of ArUco markers and applies mathematical transformations to ascertain the position and orientation of objects.

Centroid and Orientation Calculation: OpenCV's `cv2.aruco.detectMarkers()` function was used to detect ArUco markers and extract their corner coordinates and IDs. The detected corners were used to calculate the marker's side length in pixels, while `cv2.aruco.estimatePoseSingleMarkers()` estimated the marker's position and orientation. The known marker side length was then used to compute a pixel-to-mm scaling factor for distance measurements. The scaling factor and marker center coordinates were used as reference points for locating objects. For each valid object contour, the centroid was calculated using image moments:

$$(C_x, C_y) = \left(\frac{M_{10}}{M_{00}}, \frac{M_{01}}{M_{00}} \right) \quad (6)$$

where $M_{ij} = \sum_x \sum_y x^i y^j I(x, y)$ is the raw image moment of order (i, j) of the binary contour mask $I(x, y)$, computed in OpenCV using `cv2.moments()`.

The distance between the object and the ArUco marker is calculated using the Euclidean distance formula:

$$D_{px} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (7)$$

where (x_1, y_1) and (x_2, y_2) are the centroid coordinates of the ArUco marker and the object, respectively. The pixel distance is then converted to millimeters using the scaling factor ρ , determined from the known physical dimensions of the ArUco marker:

$$D_{mm} = \rho \cdot D_{px}, \quad \rho = \frac{L^{mm}}{L^{px}} \quad (8)$$

where L^{mm} is the known side length of the printed marker in millimeters, and L^{px} is its measured side length in pixels. The orientation angle was obtained from the detected ArUco marker corners using the bottom edge of the marker as the reference axis for coordinate alignment. The rotation matrix was then applied to align the object's local coordinates with the marker frame. The transformation is defined as:

$$\begin{bmatrix} x_{marker} \\ y_{marker} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_{object} \\ y_{object} \end{bmatrix} \quad (9)$$

Where θ is the rotation angle determined from the orientation of the ArUco marker in the vision system. This step is crucial to ensure the object's coordinates are accurately represented relative to the ArUco marker.

Results

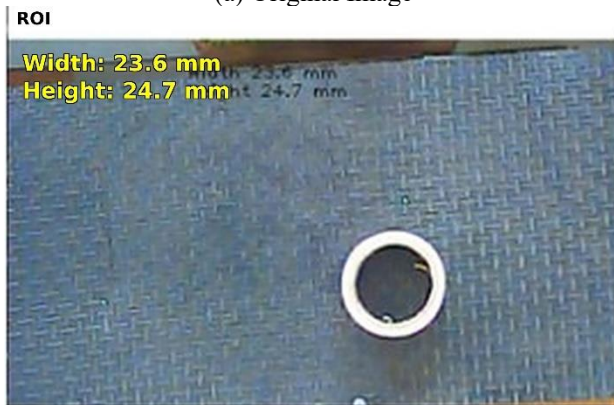
A. Image Preprocessing

The image preprocessing pipeline begins with the original image of a circular object on a conveyor belt. A region of

interest (ROI) is selected to focus on the relevant part of the image. Gamma correction enhances contrast, followed by Gaussian blur to reduce noise. The image is then converted to grayscale and color-inverted to improve feature visibility. Finally, adaptive thresholding isolates the object by dynamically segmenting it from the background, making it suitable for further analysis.



(a) Original Image

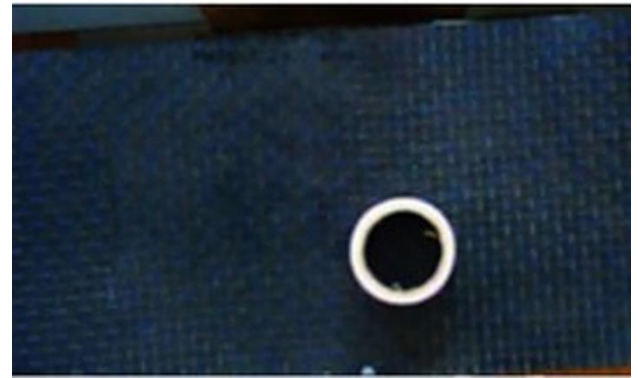


(b) Region of Interest



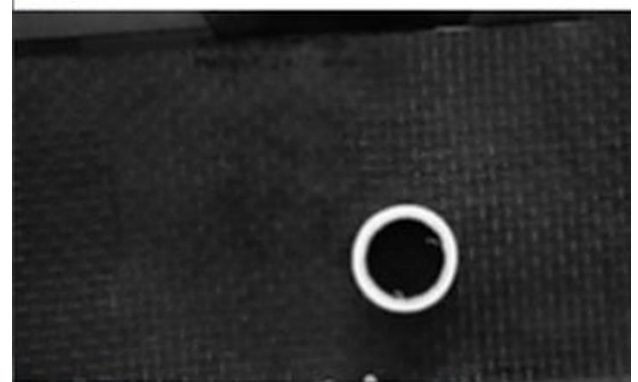
(c) Gamma Correction

Gaussian Blur



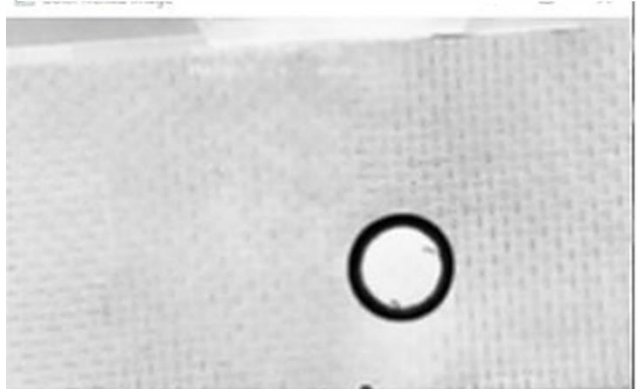
(d) Gaussian Blur

Gray Scale



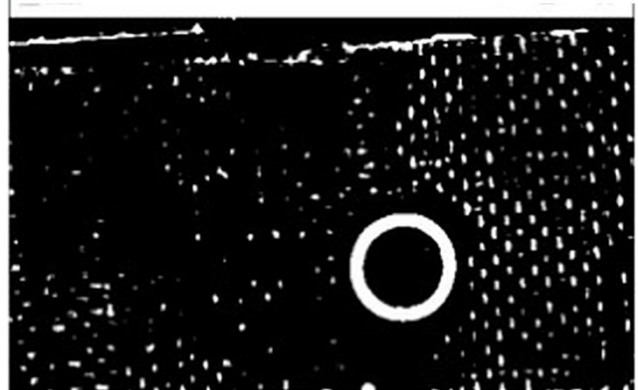
(e) Grayscale Conversion

Color Inverted Image



(f) Color Inversion

mask

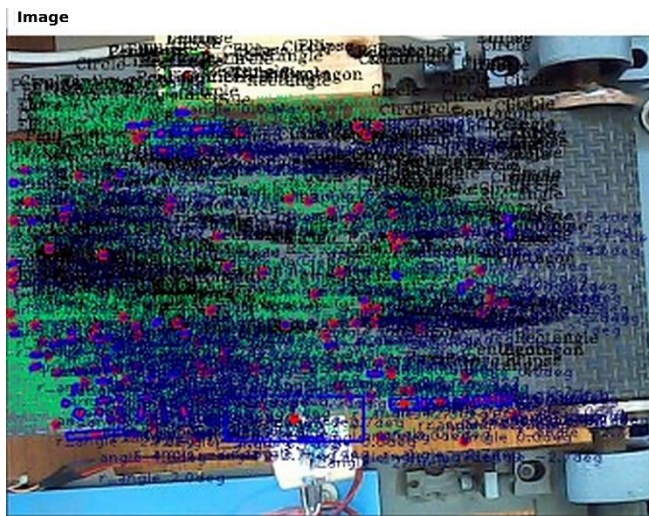


(g) Adaptive Thresholding

Figure 4. Image preprocessing steps: From the original image to the filtered and segmented output, enhancing object visibility and detection.

B. Masking and Filtering

The system achieved high accuracy in detecting objects of various shapes, sizes, and colors through effective masking and filtering techniques. Binary and grayscale masks, applied using `cv2.threshold()`, improved edge detection in noisy environments. Color-based masking with `cv2.inRange()` enhanced object identification under varying lighting conditions. Gaussian blur (`cv2.GaussianBlur()`) reduced noise, while adaptive thresholding improved contour detection. Shape- and area-based filtering using `cv2.contourArea()` ensured that only significant objects were detected. These techniques minimized false positives and enhanced real-time object detection, making the system more reliable.



(a) Area and Shape-Based Filtering



(b) Color (RGB) Mask

Figure 5. Image processing stages: RGB filtering and contour filtering.

C. Localization and Orientation Determination Using ArUco Markers

The images below show that the objects are accurately detected, as indicated by the blue boundaries drawn around them. Moreover, the text displayed in each image indicates the distance and orientation of the object with respect to ArUco marker.

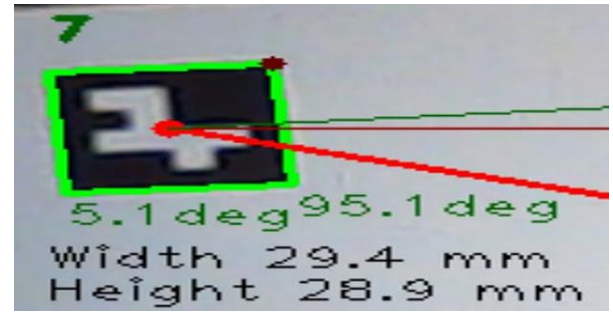


Figure 6. Marker Detection.

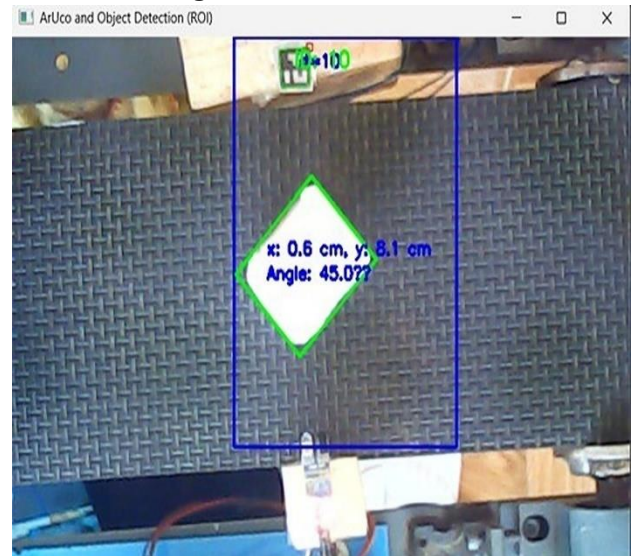


Figure 7. Sample result showing the detected object boundary together with the measured distance ($x = 0.6$ cm and $y = 8.1$ cm) and orientation (45 deg) relative to the ArUco marker.

D. Natural Language Processing

Integrating Natural Language Processing (NLP) into the system enhances user interaction by enabling voice-based control of object detection. Using the Vosk speech recognition engine, the system listens for simple spoken commands such as “circle”, “rectangle”, or “all” to dynamically filter and identify specific object shapes in the video stream.

E. Performance Evaluation

Table 2 compares the positions of ten objects estimated by the proposed system with those measured manually using the ArUco marker technique. Moreover, the system accurately estimated the objects' orientation within a margin of ± 3 degrees.

Table 2. Comparison of Calculated and Measured Positions

Object	Calculated (mm)	Measured (mm)	Error (mm)
Object 1	(72.8, 103.8)	(74, 102)	(1.2, 1.8)
Object 2	(94.2, 101.5)	(95, 100)	(0.8, 1.5)

Object 3	(83.5, 117.3)	(85, 116)	(1.5, 1.3)
Object 4	(69.4, 138.2)	(70, 137)	(0.6, 1.2)
Object 5	(75.6, 124.3)	(75, 124)	(0.6, 0.3)
Object 6	(84.5, 132.4)	(85, 133)	(0.5, 0.6)
Object 7	(97.4, 184.6)	(96, 185)	(1.4, 0.4)
Object 8	(52.2, 113.7)	(53, 115)	(0.8, 1.3)
Object 9	(86.5, 154.8)	(85, 155)	(1.5, 0.2)
Object 10	(57.2, 165.9)	(57, 165)	(0.2, 0.9)

Table 3. Computational cost of the proposed pipeline compared with the deep learning baseline (YOLOv8n).

Method	Latency (ms)	FPS	Peak RAM (MB)	Model (MB)
Proposed (ArUco + classical CV)	33.97	29.4	226.3	None
YOLOv8n	515.79	1.9	504.9	6.2

F. Computational Efficiency against a Deep Learning Baseline

To demonstrate that our proposed method requires less computation than deep learning methods, we measured its performance rather than relying solely on previous studies. We evaluated the complete processing pipeline, which includes image acquisition, camera distortion correction, image preprocessing, contour detection, ArUco marker detection, and coordinate transformation. The performance of our method was then compared with a widely used deep learning object detection model (YOLOv8n) using the same input images and the same computer hardware.

All methods were tested using the same set of 1,000 images with a resolution of 640×480 pixels. The average processing time per image (latency) was measured from the beginning to the end of the pipeline. The first 50 images were excluded to eliminate the effects of program initialization. Frames per second (FPS) were calculated from the average processing time, and the maximum memory (RAM) usage during execution was also recorded. The deep learning model was evaluated using its official pretrained weights, and all experiments were performed on the CPU only, without using a GPU, to represent the type

of hardware commonly used in industrial edge computing. The experiments were conducted on an Intel Pentium Silver N5030 processor using Python 3.13, PyTorch 2.7, and OpenCV 4.11.

The results demonstrate that our proposed method is computationally efficient, supporting the claims made in this study. Since the proposed system does not use deep learning, it does not require loading a trained model into memory. Instead, it relies on a small number of lightweight OpenCV image processing functions, making it faster and less memory-intensive than deep learning methods.

Conclusion

Our research introduced a computer vision method that utilizes ArUco markers for object detection, localization, and measurement. This system effectively recognizes objects of different shapes and sizes and accurately determines their dimensions, positions, and orientations relative to a reference marker.

A. Issues and Limitations

Despite the system's strengths, several challenges were identified. First, the accuracy of distance measurements is susceptible to factors such as lens calibration and environmental factors that can cause discrepancies when identifying target objects. Furthermore, using ArUco markers for point identification can restrict the system's adaptability in cluttered environments or during object motion. One key focus of the image processing algorithms is ensuring real-time performance. Although most modern high-performance hardware can sustain the complex computations, the proposed system operates effectively on standard and resource-constrained systems without requiring GPUs or specialized processing hardware.

B. Improvements and Future Work

Using multiple markers with different orientations and color-based designs can improve accuracy. Tracking stability can also be enhanced through techniques like Kalman filters or optical flow. In future work, the following improvements can be implemented:

- Situational awareness can be enhanced by continuous monitoring and analysis of the physical environment through sensors and the vision systems.
- Integrate a binocular vision system to obtain depth information, thereby improving object recognition and manipulation accuracy through stereo vision algorithms to enhance spatial understanding and reduce localization errors.

Acknowledgments

The authors gratefully acknowledge the Department of Electrical Engineering, Pakistan Institute of Engineering and Applied Sciences (PIEAS), Islamabad, Pakistan, for providing the laboratory facilities and computing resources used in this work.

Competing Interests

The authors of this paper have acknowledged no conflicts of interest.

Authors' Contribution

Conceptualization and methodology, F.R., S.N. and M.O.; software and implementation, F.R., S.N. and M.O.; experiments, data curation and formal analysis, F.R., S.N., M.O. and M.A.; validation, N.R.J., M.A. and R.M.;

visualization, F.R. and S.N.; writing—original draft preparation, F.R., S.N. and M.O.; writing—review and editing, N.R.J., M.A. and R.M.; supervision, N.R.J. All authors have read and agreed to the published version of the manuscript.

References

1. Suzuki. S and Abe. K, "Topological structural analysis of digitized binary images by border following", *Computer Vision, Graphics, and Image Processing*, Vol. 30, No. 1, 1985, pp. 32-46.
[https://doi.org/10.1016/0734-189X\(85\)90016-7](https://doi.org/10.1016/0734-189X(85)90016-7)
2. Ren. S, He. K, Girshick. R and Sun. J, "Faster R-CNN: Towards real-time object detection with region proposal networks", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 39, No. 6, 2017, pp. 1137-1149.
<https://doi.org/10.1109/TPAMI.2016.2577031>
3. Gao. M, Cai. Q, Zheng. B, Shi. J, Ni. Z, Wang. J and Lin. H, "A hybrid YOLOv4 and particle filter based robotic arm grabbing system in nonlinear and non-Gaussian environment", *Electronics*, Vol. 10, No. 10, 2021, p. 1140.
<https://doi.org/10.3390/electronics10101140>
4. Roshni. N and Kumar. T. K. S, "Pick and place robot using the centre of gravity value of the moving object", *2017 IEEE International Conference on Intelligent Techniques in Control, Optimization and Signal Processing (INCOS)*, 2017, pp. 1-5.
<https://doi.org/10.1109/ITCOSP.2017.8303079>
5. Kumar. V, Wang. Q, Wang. M, Rizwan. S, Shaikh. S. M and Liu. X, "Computer vision based object grasping 6-DoF robotic arm using PiCamera", *2018 4th International Conference on Control, Automation and Robotics (ICCAR)*, 2018, pp. 111-115.
<https://doi.org/10.1109/ICCAR.2018.8384653>
6. Zhang. H, Zhou. X, Lan. X, Li. J, Tian. Z and Zheng. N, "A real-time robotic grasping approach with oriented anchor box", *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, Vol. 51, No. 5, 2021, pp. 3014-3025.
<https://doi.org/10.1109/TSMC.2019.2917034>
7. Fernandes. R. O, Fonseca. M and Conceição. A. G. S, "Visual servoing with fiducial landmarks for differential drive mobile robot", *2017 Latin American Robotics Symposium (LARS) and 2017 Brazilian Symposium on Robotics (SBR)*, 2017, pp. 1-6.
<https://doi.org/10.1109/SBR-LARS-R.2017.8215268>
8. Hartley. R and Zisserman. A, *Multiple View Geometry in Computer Vision*, 2nd ed., Cambridge University Press, Cambridge, UK, 2004.
<https://doi.org/10.1017/CBO9780511811685>
9. Kendall. A, Grimes. M and Cipolla. R, "PoseNet: A convolutional network for real-time 6-DoF camera relocalization", *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 2938-2946.
<https://doi.org/10.1109/ICCV.2015.336>
10. Zuo. Y, Qiu. W, Xie. L, Zhong. F, Wang. Y and Yuille. A. L, "CRAVES: Controlling robotic arm with a vision-based economic system", *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 4214-4223.
<https://doi.org/10.1109/CVPR.2019.00434>
11. Garrido-Jurado. S, Muñoz-Salinas. R, Madrid-Cuevas. F. J and Marín-Jiménez. M. J, "Automatic generation and detection of highly reliable fiducial markers under occlusion", *Pattern Recognition*, Vol. 47, No. 6, 2014, pp. 2280-2292.
<https://doi.org/10.1016/j.patcog.2014.01.005>
12. Hu. C.-Y, Chen. C.-R, Tseng. C.-H, Yudha. A. P and Kuo. C.-H, "Visual servoing spanner picking and placement with a SCARA manipulator", *2016 IEEE International Conference on Industrial Technology (ICIT)*, 2016, pp. 1632-1637.
<https://doi.org/10.1109/ICIT.2016.7475006>
13. Lightbody. P, Krajník. T and Hanheide. M, "An efficient visual fiducial localisation system", *ACM SIGAPP Applied Computing Review*, Vol. 17, No. 3, 2017, pp. 28-37.
<https://doi.org/10.1145/3161534.3161537>
14. Romero-Ramírez. F. J, Muñoz-Salinas. R and Medina-Carnicer. R, "Speeded up detection of squared fiducial markers", *Image and Vision Computing*, Vol. 76, 2018, pp. 38-47.
<https://doi.org/10.1016/j.imavis.2018.05.004>
15. Mondéjar-Guerra. V, Garrido-Jurado. S, Muñoz-Salinas. R, Marín-Jiménez. M. J and Medina-Carnicer. R, "Robust identification of fiducial markers in challenging conditions", *Expert Systems with Applications*, Vol. 93, 2018, pp. 336-345.
<https://doi.org/10.1016/j.eswa.2017.10.032>
16. dos Santos Cesar. D. B, Gaudig. C, Fritsche. M, dos Reis. M. A and Kirchner. F, "An evaluation of artificial fiducial markers in underwater environments", *OCEANS 2015 - Genova*, 2015, pp. 1-6.
<https://doi.org/10.1109/OCEANS-Genova.2015.7271491>
17. Khattak. S, Papachristos. C and Alexis. K, "Marker based thermal-inertial localization for aerial robots in obscurant filled environments", *Advances in Visual Computing (ISVC 2018), Lecture Notes in Computer Science*, Vol. 11241, Springer, Cham, 2018, pp. 565-575.
https://doi.org/10.1007/978-3-030-03801-4_49
18. Makabe. L, Santo. H, Okura. F and Matsushita. Y, "Shape-coded ArUco: Fiducial marker for bridging 2D and 3D modalities", *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2022, pp. 2315-2324.
<https://doi.org/10.1109/WACV51458.2022.00237>
19. Huang. Q, DeGol. J, Frago. V, Sinha. S. N and Leonard. J. J, "Optimizing fiducial marker placement for improved visual localization", *IEEE Robotics and Automation Letters*, Vol. 8, No. 5, 2023, pp. 2756-2763.
<https://doi.org/10.1109/LRA.2023.3260700>

