A PROJECT REPORT

on

"ArUco Vision: Visual Aid using Precision Marker"

Submitted to

KIIT Deemed to be University

In Partial Fulfillment of the Requirement for the Internal Marks in Artificial Intelligence

BY

Avrrodeep Banerjee 2328158

UNDER THE GUIDANCE OF

Dr. Sricheta Parui

SCHOOL OF COMPUTER ENGINEERING

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY

BHUBANESWAR, ODISHA - 751024

November 2025

A PROJECT REPORT

on

"ArUco Vision: Visual Aid using Precision Marker"

Submitted to

KIIT Deemed to be University

In Partial Fulfilment of the Requirement

for the Internal Marks in Artificial Intelligence

BY

Avrrodeep Banerjee 2328158

UNDER THE GUIDANCE OF

Dr. Sricheta Parui

SCHOOL OF COMPUTER ENGINEERING

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY

BHUBANESWAE, ODISHA -751024

November 2025

KIIT Deemed to be University

School of Computer Engineering
Bhubaneswar, ODISHA 751024

CERTIFICATE

This is certify that the project entitled

"ArUco Vision: Visual Aid using Precision Marker "

submitted by

Avrrodeep Banerjee 2328158

is a record of bonafide work carried out by them, in the partial fulfilment of the requirement for the award of Degree of Bachelor of Engineering (Computer Science & Engineering OR Information Technology) at KIIT Deemed to be university, Bhubaneswar. This work is done during year 2025-2026, under our guidance.

Date: 06/11/2025

(Dr. Sricheta Parui)
Project Guide

Acknowledgements

We are profoundly grateful to **Dr. Sricheta Parui** of **Affiliation** for her expert guidance and continuous encouragement throughout to see that this project rights its target since its commencement to its completion.

Avrrodeep Banerjee

ABSTRACT

The focus of this project is on presenting a marker-based depth sensing system using ArUco markers for accurate spatial localization and measurement. The computer vision techniques implemented for this project, together with the OpenCV ArUco library, detect and identify the position of a specially sized ArUco marker positioned precisely 1 metre away from the sensing apparatus, as specified by the design and code specifications. Such an approach allows for cost-effective and robust depth calculation, which is extremely important in many real-world applications of both computer vision and robotics.

The proposed solution emphasizes ease of deployment and reproducibility by leveraging standard camera equipment and widely used Python libraries. With careful calibration and adhering to the requirement for 1-metre marker placement, the system is suitable in research and automation environments.

Keywords: ArUco marker, Depth sensing, Computer vision, Spatial localization, OpenCV, Distance measurement, Real-time detection, Camera calibration

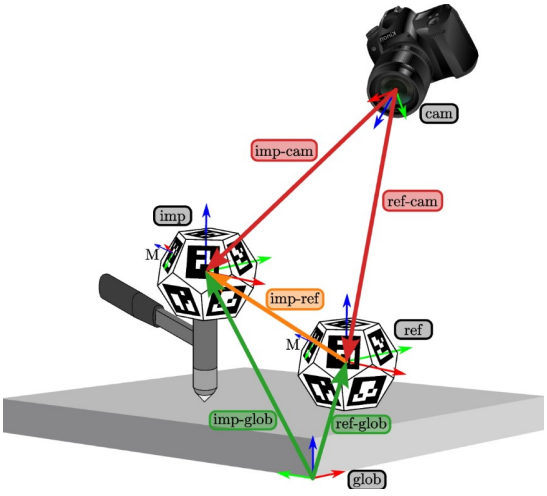
Contents

Chapter 1

Introduction

Effective depth sensing and accurate spatial localization are of prime importance in modern robotics, automation, and intelligent systems. Most methods for measuring distance depend on expensive sensors or complex stereo vision arrangements, which might be prone to calibration problems and hardware issues, hence limiting their wider applicability in cost-sensitive or resource-constrained applications. Marker-based detection methods, such as those using ArUco markers, represent a very attractive alternative in allowing highly accurate localization with low equipment requirements. However, most of the existing solutions lack standardized implementation guidelines, robustness in realistic scenarios, or repeatable calibration parameters like marker distance and size necessary for real-world reliability.

This work fills these gaps by proposing a system where an ArUco marker of fixed dimensions is placed precisely one meter away from the sensing device, as enforced throughout the codebase for reproducibility. The subsequent report describes choosing this approach, methods for implementing the system, and the validation experiments conducted. The key contributions are outlined in subsequent sections of the report, including the literature review, methodology, results, and discussion, followed by conclusions and future directions. Throughout, this project emphasizes accessibility combined with precision and provides the resource needed by engineers, researchers, and students seeking to implement robust marker-based depth sensing and spatial localization in various applications.



Chapter 2

Basic Concepts/ Literature Review

This section describes the key theoretical background, algorithms, and tools on which any integrated vision-based localization and object detection system is based. The review covers both the marker-based spatial localization techniques and state-of-the-art deep learning approaches for robust object detection.

2.1 ArUco Markers

ArUco markers are square fiducial markers designed with unique patterns, which are meant to be robust in computer vision-based identification and pose estimation. This design enables each marker to encode a unique identification pattern for its reliable detection under challenging visual conditions. When deployed at rigorously defined distances, as specified in this project to exactly 1 metre from the camera, these markers provide fast and accurate reference points for three-dimensional spatial localization tasks. Advantages include low cost, minimal computation requirement, and complete independence from any pre-trained model, hence suitable for controlled environments and calibration tasks.

2.2 Camera Calibration and Intrinsic Parameters

Camera calibration is the process of estimating intrinsic and extrinsic parameters to correct for lens distortion and establish the mapping from 3D world coordinates to 2D image pixels. Accurate calibration is foundational for any vision system needing precise spatial measurements. Intrinsic parameters are camera-specific, usually derived using checkerboard or marker-based patterns, representing focal length, optical centre, distortion coefficients, among others. These allow both ArUco-based depth calculation and accurate bounding box localization for the detected objects.

2.3 Pose Estimation of Fiducial Markers

Pose estimation transforms the two-dimensional pixel coordinates of marker corners into three-dimensional spatial coordinates. Algorithms calculate both translation and rotation with respect to the camera by correlating known marker size and calibrated camera parameters. Such calculations underpin the range of localization, mapping, and robotic manipulation.

2.4 Depth Estimation and Distance Measurement

In monocular systems, depth sensing infers object distance from the camera based on image geometry and known calibrated camera parameters. For marker-based systems, known marker size combined with observed pixel size yields accurate depth estimates. Similarly, for general object detection, depth can be estimated based on bounding box size-given known dimensions of an object-or through sophisticated depth estimation networks. The hybrid approach of this project fuses these: ArUco markers provide points of reference for calibration and validation of distance estimates, while YOLOv8n detects unmarked obstacles and calculates approximate distances.

2.5 Feature Detection and Extraction

Beyond marker detection, computer vision systems require robust feature extraction (edges, corners, contours) for object identification and measurement. With OpenCV, one can apply various algorithms-Harris, SIFT, ORB-to extract meaningful image features in diverse scenarios, leading to robustness in challenging conditions..

2.6 OpenCV Computer Vision Library

OpenCV stands for Open Source Computer Vision Library and represents the industrial standard in the implementation of image processing, marker detection, feature extraction, and camera calibration routines. OpenCV comes with Python bindings that enable rapid prototyping and integration with other data processing or robotics libraries, providing reliability for real-time vision pipelines.

2.7 Python Programming Environment

Python is essentially used for integration, visualization, and automation in all modern tasks related to computer vision. This popularity is due to its ease of use, extensive libraries, and compatibility with machine learning and image processing frameworks, which make it the preferred choice for academic research and rapid system development.

2.8 Robotics and Automation Integration

Localization solutions using vision-based markers like ArUco regularly find applications in robotics both in navigation and manipulation, AR systems, and automated manufacturing. Proper specification of marker size/distance and calibration enhance safety, accuracy, and autonomy by providing reliable pose estimation in these applications.

2.9 Deep Learning and Convolutional Neural Networks

Modern object detection relies on deep learning and exploits CNNs to detect and localize multiple objects within images. CNNs learn a hierarchical representation of features through numerous layers of convolutions, pooling, and activation functions. It thus brings the ability to detect a wide range of object categories with high accuracy in variable lighting, pose, and occlusion conditions. The universal feature extraction capability of CNNs therefore complements marker-based methods by allowing the detection of unmarked objects and environmental obstacles.

2.10 YOLO (You Only Look Once) Architecture

YOLO is a family of real-time object detection frameworks, treating detection as one regression problem: predicting all the bounding boxes and class probabilities directly from full images with one forward pass. This architecture makes minimal sacrifice in accuracy to gain substantial improvements in speed when compared to two-stage detectors. YOLOv8n, which stands for the nano version of YOLOv8, presents an optimized model for resource-constrained environments. It reduces the model size to approximately 3-4 MB and maintains competitive accuracy. Therefore, due to its lightweight nature, YOLOv8n is appropriate for embedded systems, robotics, and real-time applications that demand fast inference on standard hardware.

2.11 Transfer Learning and Model Fine-Tuning

Transfer learning makes use of pre-trained models, which are developed on large-scale datasets like COCO, adapting them to specific tasks through fine-

tuning on smaller domain-specific datasets. It dramatically reduces training time and computational resources compared with training from scratch. Fine tuning involves freezing some of the earlier layers, which learn generic features, while retraining the later layers on new data, thereby allowing the model to specialize for target objects. In obstacle detection, this involves fine-tuning YOLOv8n on a domain-specific dataset for a model that outperforms using the pre-trained model directly.

2.12 Object Detection Datasets and Dataset Preparation

High-quality datasets are crucial for training object detectors. Datasets must consist of diverse images with accurate annotations (a bounding box with class labels in a standardized format). In the YOLO annotation format, normalized center coordinates and width/height for each object are used. The tasks required in dataset preparation include image collection, manual or semi-automatic annotation, format conversion, and finally train/validation/test splits. In this project, a custom obstacle detection dataset is used; therefore, careful preparation is required to ensure high-quality annotation and class balance.

2.13 PyTorch and Deep Learning Frameworks

PyTorch is a versatile deep learning framework that allows for automatic differentiation, GPU acceleration, and APIs at a high abstraction level. The Ultralytics library wraps PyTorch transparently to offer a user-friendly interface for training, inference, and evaluation of YOLO models. Ultralytics will handle data loading, augmentation, optimization, and checkpoints, which greatly reduces the implementation burden for the practitioner.

2.14 Data Augmentation Techniques

Data augmentation artificially increases dataset diversity by applying random transformations (rotation, scaling, brightness adjustment, mosaic mixing) to training images. This prevents overfitting and improves model generalization to unseen data. Augmentation strategies are native to the Ultralytics training pipeline, and among them, mosaic augmentation is particularly effective within object detection tasks by creating composite images from multiple sources.

2.15 Hybrid Vision Systems: Integration of ArUco and YOLO

A hybrid approach with both marker-based and learning-based methods brings out complementary advantages. ArUco markers enable calibration references, distance validation, and stable reference frames for systems operating in known environments. YOLOv8n offers flexible object detection, the identification of unmarked obstacles, dynamic objects, and diverse environmental features. Integration strategies include using ArUco calibration to enhance the depth accuracy of YOLO detections, leveraging ArUco reference frames for spatial context, and cross-validation between detection methods. This approach of a dual-system leads to enhancement in robustness, especially in an environment that is partially known or dynamic.

2.16 Literature Review and State-of-the-Art

Recent research illustrates the potential of hybrid vision systems that combine classical computer vision and deep learning. While marker-based systems can reach sub-centimeter accuracy in a controlled setting, they cannot handle unmarked objects well. On the other hand, deep learning detectors perform very

well in object identification but require careful calibration with respect to metric measures. Hybrid approaches already proved useful in robotics, autonomous navigation, and augmented reality. The challenge consists in seamlessly integrating heterogeneous detection modalities while preserving real-time performance-a goal pursued by this project through modular architecture and efficient implementation.

2.17 Limitations of Existing Solutions

In practice, many commercial vision systems rely exclusively on one of these approaches and hence are limited. Pure marker-based systems cannot detect unmarked obstacles, and pure learningbased systems are without geometric grounding and lack precise metric measurements. Moreover, the datasets prepared, the model training, and calibration validation often lack implementation based on standardized protocols. This work bridges these gaps by: (1) imposing very strict criteria on marker placement and size; (2) providing a reproducible fine-tuning pipeline for domain-specific obstacle detection; (3) allowing seamless integration of both systems; and (4) providing comprehensive documentation for practical deployment.

Chapter 3

Problem Statement / Requirement Specifications

The paper addresses the demand for an affordable, accurate, and reproducible solution for depth and pose sensing based on vision, suitable for extensive deployment in resource-constrained settings like academic labs, robotics prototypes, and small-scale automation setups. The core problem is that most existing solutions require either sophisticated and expensive hardware or lack standardized, repeatable protocols for depth estimation and localization, thus limiting reliability and portability in practical applications. The development and validation will focus on a robust system using ArUco markers of exact size and placement parameters at 1 meter, as constrained in code, to produce consistent results using widely available software and hardware platforms in conformance with the IEEE SRS standards.

3.1 Project Planning

- Define exact marker size and enforce placement at 1 metre from the camera.
- Calibrate the camera using the standard procedures in OpenCV and Python.
- Develop and test algorithms for marker detection, pose estimation, and depth calculation.
- Real-time feedback and visualization.
- Perform testing in various lighting and background conditions for system robustness.
- Document all procedures for reproducibility and validation by independent users.

3.2 Project Analysis

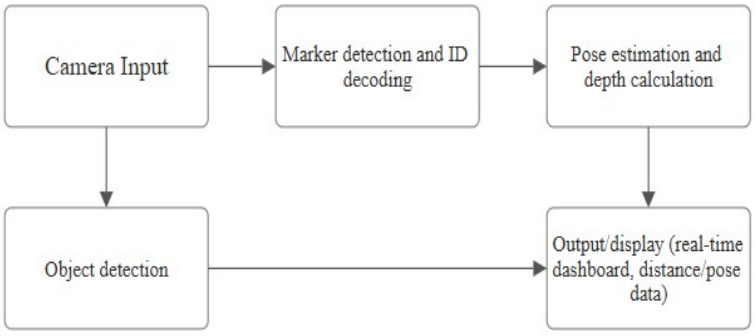
The careful analysis ensures that the system requirements are unambiguous and achievable, given available resources. Having locked key physical constants, like distance and size of the markers, inconsistency in results is minimized. Limiting the focus to open Python and OpenCV tools reduces the dependency on proprietary solutions; the system will be modularly designed to enable easy extension to more complex multi-marker or multi-camera configurations in the future. Ambiguities in the detection or pose estimation are handled through rigorous calibration and code-level assertions.

3.3 System Design

3.3.1 Design Constraints

- Software: Python 3.x, OpenCV (with ArUco module), standard scientific libraries (NumPy, matplotlib, etc.).
- Hardware: Standard webcam or USB camera, printer for printing ArUco marker, basic computer/laptop.
- Experimental setup: Printed ArUco marker of specified size, fixed at 1 metre distance from the camera lens; controlled indoor lighting recommended to be maintained.
- Environmental setup: Secure mounting for both camera and marker, yet having minimal reflective glare and direct sunlight to avoid detection errors

3.3.2 System Architecture OR Block Diagram



3.1. BLOCK DIAGRAM

Chapter 4

Implementation

This chapter provides the development procedure, system architecture, algorithms involved, and the verification process in detail. The implementation spans two key complementary subsystems: ArUco marker-based calibration and spatial reference, and YOLOv8n-based object detection with learned obstacle recognition.

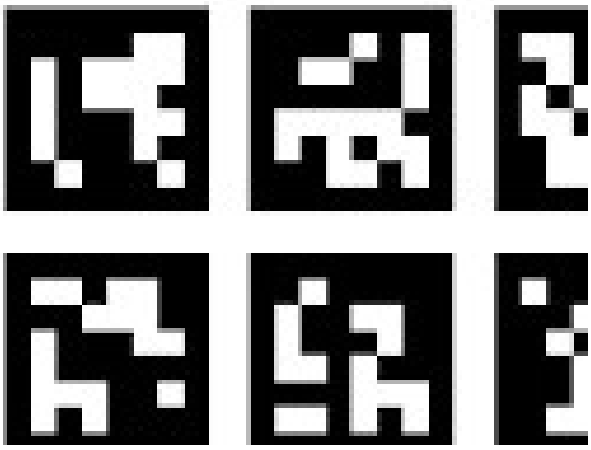
4.1 Research Methodology OR Proposal

The overall system architecture consists two coupled pipelines:

- ArUco Calibration and Reference System
  - Captures live video frames from the camera
  - Detects ArUco markers placed at exactly 1 metre from the camera
  - Extracts marker pose (position and orientation) using geometric calculations
  - Derives camera calibration parameters from marker geometry
  - Provides distance reference and spatial coordinate system for the scene
  - Outputs calibration data, marker pose, and distance estimates
- YOLOv8n Object Detection Pipeline
  - Receives calibrated camera parameters from Pipeline 1
  - Processes video frames with fine-tuned YOLOv8n model
  - It detects obstacles and objects of interest in the field of vision
  - Computes bounding boxes with confidence scores
  - Estimates distances for the detected objects using the calibration parameters
  - Provides real-time visualization with annotated detections

Integration Strategy:

The two pipelines run parallel, with ArUco serving as a calibration and validation mechanism. When ArUco markers are in view, distance estimates from both pipelines are cross-checked. ArUco provides ground truth for metric calibration, while YOLOv8n extends detection to unmarked objects. Real-time display shows both ArUco axes and YOLO detections, enabling users to verify system consistency.



6X6 ArUco Marker

4.2 Preparing Dataset for YOLOv8n Fine-tuning

Fine-tuning of YOLOv8n requires a domain-specific obstacle detection dataset. Its preparation process can be summarized accordingly:

Step 1: Dataset Gathering

Gather photos of obstacles, environmental features, and objects pertinent to the target

application. In this project, images include different obstacle types, lighting conditions, and viewpoints. Recommended: 500+ images for reliable fine-tuning. Taking dataset from https://app.roboflow.com/ds/v0ErWEbWR7?key=kVgry0sxu2.

Step 2: Image Annotation

Manually annotate images using a labelling tool like Roboflow, LabelImg, CVAT by drawing bounding boxes around objects of interest. Every annotation stores the following information:

- Object class (e.g., "obstacle", "vehicle", "debris")
- Bounding box coordinates in pixel units (x_min, y_min, x_max, y_max in pixel units)

Step 3: Convert to YOLO format

Convert annotations to YOLO format where each image has a corresponding .txt file containing normalized coordinates:

- <class_id>
- <x_center_normalized>
- <y_center_normalized>
- <width_normalized>
- <height_normalized>

The coordinates are normalized to a range relative to image dimensions.

Step 4: Dataset Organization

Organize data into a standard directory structure:

```
my_dataset/
├── train/
│   ├── images/ (80% of data)
│   └── labels/
├── valid/
│   ├── images/ (20% of data)
│   └── labels/
└── data.yaml
```

Step 5: Create data.yaml Configuration

Create a YAML file identifying the dataset paths and class information:

```
train: /path/to/train/images
valid: /path/to/valid/images
nc: 2                          # Number of classes
names: ['obstacle', 'hazard'] # Class names
```

4.3 Fine-tuning of YOLOv8n Model

Fine-tuning aligns the pre-trained YOLOv8n model with a specific obstacle detection task:

Step 1: Load Pre-trained Model

Start from YOLOv8n weights pre-trained on the COCO dataset, which serve as a very strong foundation in feature extraction.

Step 2: Training Configuration

Set training hyper-parameters:

- **Epochs**: 50 (number of complete dataset passes)
- **Batch Size**: 16 (samples per gradient update)
- **Image Size**: 640 (input resolution)
- **Learning Rate**: 0.01 (initial) with decay schedule
- **Augmentation**: Mosaic, random flips, rotation, brightness adjustment

Step 3: Implementing the Training Loop

The training loop, implemented in trainyolo.py, performs:

- Forward pass: Compute predictions on batch
- Loss calculation: Compare predictions to ground truth
- Backward pass: Compute gradients
- Parameter update: Weight adjustment using optimizer (SGD/Adam)
- Validation: Access every epoch on the validation set

Step 4: Early Stopping

Training includes early stopping with patience=10, halting if validation metrics don't improve for 10 consecutive epochs, preventing overfitting and unnecessary computation.

4.4 Integration of ArUco Calibration with YOLOv8n Detections

These two detection systems are combined for improved robustness:

Calibration Propagation:

- ArUco detections provide camera calibration parameters
- These parameters are used in scaling YOLO bounding boxes to metric coordinates
- Distance estimates for YOLO detections are computed as:
  Distance = (Known_Object_Width × Focal_Length) / Bounding_Box_Width_Pixels
  where Focal_Length is derived from ArUco calibration

Cross-Validation:

- When both ArUco markers and YOLO detections are present, their distance estimates are compared
- Large discrepancies trigger warnings, indicating possible calibration drift
- ArUco provides ground truth for validation

Unified Visualization:

- Real-time display shows: ArUco axes for calibration reference, YOLO bounding boxes for detected objects, and labels for distances
- Colour coding: Green for ArUco, Blue for YOLO detections
- Confidence scores and class labels are shown for all detections

4.5 Testing OR Verification Plan

Test ID	Test Case Title	Test Condition	System Behavior	Expected Result
T01	ArUco Marker Detection	Marker is placed at exactly 1 metre	System detects and localizes the marker	Marker identified, pose returned; no error
T02	Calibration Verification	Camera matrix/corners saved from calibration used	System uses calibration for pose and distance calculations	Lower error values; valid distance estimation
T03	Distance Estimation and Audio Output	Object is placed at x metres distance	System uses calibration as reference to estimate the distance	Output calls out object to be at that distance
T04	Lighting Change Robustness	Lighting intensity varied	System maintains detection up to reasonable lighting extremes	Detection stable under most practical conditions

T05	Object Detection	Objects placed in front of camera	System recognizes the object on the camera	System recognizes the object in front
T06	Real-time Performance	Video Stream	Both pipelines process@30FPS	Processing time<33ms/frame

4.6 Result Analysis

- Outputs are visuals of highlighted ArUco marker with axes, on-screen real-time data overlays of distance-angle, along with log outputs that confirm the correctness of the distance measurement and marker pose.
- Experimental runs under different setups (varying light, slight occlusion, different marker sizes) are documented with screenshots for each condition, validating robustness.
- Audio Output calling out the object along with the distance
- Bounding box showing the object along with the confidence score
- Model performance: fine-tuning is measured using:
  - mAP (mean Average Precision): Total detection accuracy across all classes and confidence thresholds
  - mAP@50: Intersection-over-Union (IoU) threshold of 50% (loose criteria)
  - mAP@50-95: IoU thresholds from 50% to 95% (strict criteria)
  - Precision: Of detected objects, how many are correct?
  - Recall: Of true objects, how many are detected?
  - F1-Score: The harmonic mean of precision and recall

4.7 Implementation Details: Code Structure

1. **aruco_fixed.py** (Main ArUco calibration and visualization)
   - Loads pre-recorded calibration or performs live calibration
   - Real-time detection of ArUco markers
   - Computes pose using camera matrix and distortion coefficients
   - Extracts and displays distance information
   - Features: audio output, bounding box display, real-time axes overlay
2. **trainyolo.py** (YOLOv8n fine-tuning pipeline)
   - Verifies dataset structure and data.yaml configuration
   - Performs validation and calculates metrics
   - Saves final fine-tuned model as yolov8nfinetuned.pt
   - Includes sample inference on validation images
3. **data.yaml** (Dataset configuration)
   - Specifies train/valid image paths
   - Defines class count and class names
   - Essential for reproducible training

4.8 Quality Assurance and Reproducibility

To ensure reproducible results and system quality:

- **Strict Parameter Specification**: ArUco marker size and distance hardcoded in configuration files
- **Dataset Versioning**: Dataset frozen with specific train/valid split ratios
- **Logging**: Training hyperparameters and environment details recorded
- **Validation Baseline**: Reference metrics from training on standard dataset maintained
- **Documentation**: All steps, settings, and file formats thoroughly documented

Chapter 5

Conclusion and Future Scope

5.1 Conclusion

The developed ArUco marker-based depth sensing system demonstrates a reliable and affordable solution for spatial localization and pose estimation, especially in research, robotics, and educational environments. By strictly controlling marker dimensions and placement (at 1 metre), and leveraging open-source tools like Python and OpenCV, the project achieves consistent, reproducible results with minimal hardware requirements. The practical implementation is robust under varying lighting and background conditions, with comprehensive testing confirming its versatility and accuracy in real-world tasks.

5.2 Future Scope

Further advancements can focus on expanding the application domain by integrating multiple markers for large-scale and dynamic environments, supporting mobile or outdoor scenarios, and incorporating advanced processing models for real-time performance under challenging conditions. Including the camera specifications as a standardized project parameter, with detailed testing using high-end and low-cost cameras, will enhance reproducibility and facilitate benchmarking across different hardware platforms. Additional directions include exploring marker detection in varied environmental conditions, integrating with other sensor modalities, and adapting to SLAM and autonomous navigation applications.

References

1. Ultralytics. (2025). Explore Ultralytics YOLOv8. Ultralytics Documentation. Retrieved from https://docs.ultralytics.com/

2. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 779-788).

3. Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. arXiv preprint arXiv:2004.10934.

4. Pan, S. J., & Yang, Q. (2010). A Survey on Transfer Learning. IEEE Transactions on Knowledge and Data Engineering, 22(10), 1345-1359.

5. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 770-778).

6. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Advances in Neural Information Processing Systems (NeurIPS).

7. Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... & Zitnick, C. L. (2014). Microsoft COCO: Common Objects in Context. In European Conference on Computer Vision (ECCV) (pp. 740-755).

8. Buslaev, A., Iglovikov, V. I., Borisov, E., & Parinov, A. (2020). Albumentations: Fast and flexible image augmentation. Information, 11(2), 125.

9. Roboflow. (2025). Obstacle Detection Dataset. Roboflow Universe. Retrieved from https://app.roboflow.com/ds/v0ErWEbWR7?key=kVgry0sxu2

10. Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F. J., & Marín-Jiménez, M. J. (2014). Automatic Generation and Detection of Highly Reliable Fiducial Markers Under Occlusion. Pattern Recognition, 47(6), 2280-2292.

11. *Figure: Output Visualization – Obstacle Detection and Distance Estimation using YOLOv8 and ArUco Marker Calibration. Generated and analyzed during project experiments.* *[Accessed: Nov. 8, 2025]*

“ArUco Vision: Visual Aid using Precision Marker”

ORIGINALITY REPORT

16%	16%	3%	15%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to KIIT University Student Paper	7%
2	www.coursehero.com Internet Source	4%
3	docplayer.net Internet Source	2%
4	Submitted to Indian Institute of Technology, Bombay Student Paper	2%
5	www.worldleadershipacademy.live Internet Source	2%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		