A Visual Sensor-based Approach for Robotic Pick-and-Place Operations

Taha Momayiz[1], Andrew Barth[2]
University of Cincinnati, Cincinnati, OH 45219, United States

This paper introduces an autonomous grasping system that leverages depth sensing and advanced machine learning to enable real-time object detection and manipulation. By combining the Intel RealSense D435i with a custom-trained YOLOv11 model, integrated within a ROS 2 framework running on Ubuntu 22.04, the system accurately maps object locations and commands an Interbotix X-Series ViperX-300 robot arm to perform precise pick-and-place tasks. The innovation centers on a custom coordinate transformation pipeline that converts 3D object positions from the camera's perspective to the robot's base frame, significantly improving grasp success rates and reducing execution times. Detailed experimental procedures demonstrate a 100% grasp success rate and highlight the potential of further enhancements with additional camera angles and improved training datasets. For those interested in the code and further details, the complete implementation is available on our GitHub repository at https://github.com/YumTaha/robot-vision.

I. Nomenclature

D435i	=	RealSense depth camera model
YOLOv11	=	You Only Look Once version 11 object detection model
ROS 2	=	Robot Operating System version 2
AprilTag	=	Visual fiducial system used for object detection, localization, and pose estimation in robotics
RViz	=	3D visualization tool in ROS used for displaying sensor data, robot models, and simulation environments
ViperX-300	=	Interbotix X-Series robot arm model
x_{cam}, y_{cam}, z_{cam}	=	Coordinates of an object's center as detected in the camera frame
T_{cb}	=	The 4×4 homogeneous transformation matrix from the camera reference frame to the robot's base frame
$C_{unaligned}$	=	The intermediate coordinate vector which accounts for sensor offsets and axis reordering
$\mathrm{C}_{\mathrm{aligned}}$	=	The transformed coordinate vector in the robot base frame
mAP	=	The average of the Average Precision metric across all classes in a model
Precision	=	Measures the proportion of correct predictions made by the model
Recall	=	Measures the percentage of relevant objects correctly identified by the model

II. Introduction

The need for dynamic, reliable grasping systems in unstructured environments has become increasingly prominent with the rise of robotics in industrial automation, assistive technology, and space exploration. Traditional robotic grasping methods often rely on predefined object locations, limiting their adaptability when dealing with varying object positions and unexpected movements. Recent advancements in deep learning and real-time sensor processing have opened new avenues for autonomous manipulation by enabling robust object detection and localization under diverse conditions.

This research introduces an autonomous grasping system that integrates the high-resolution depth sensing capabilities of the RealSense D435i with the rapid object detection of a custom-trained YOLOv11 model. The system is built around the Interbotix X-Series ViperX-300 robotic arm, which executes precise grasping maneuvers based on

[1]Undergraduate Student, Department of Aerospace Engineering and Engineering Mechanics, AIAA Member
[2] Assistant Professor, Department of Aerospace Engineering and Engineering Mechanics, AIAA Senior Member

continuously updated perception data. A critical element of the system is a robust coordinate transformation pipeline that addresses the well-known hand-eye calibration problem by converting detections from the camera frame to the robot’s base frame—a step essential for accurate grasp planning.

In early prototypes, the use of ROS1 in combination with a conventional depth camera proved inadequate, as it struggled to maintain a reliable live feed, execute timely object detection, and facilitate precise grasping simultaneously. These limitations underscored the need for a more robust system, which led to the adoption of ROS 2 and the integration of YOLO, a powerful machine learning algorithm capable of tracking and locating objects dynamically. Moreover, initial experiments conducted on Ubuntu 18.04 and 20.04 encountered multiple compatibility and performance issues, prompting a migration to Ubuntu 22.04. This upgrade provided a more stable and efficient environment, enabling the seamless integration of sensor data, machine learning processes, and robotic control.

In developing this system, an iterative design process was employed, drawing on open-source frameworks and real-time communication via ROS 2. The approach emphasizes modularity and flexibility, allowing the integration of sensor data and control commands to be refined over successive testing phases. Like other projects in our lab that leverage advanced manufacturing and pneumatic simulation techniques for CubeSat testing, this project illustrates how cost-effective and scalable technologies can be harnessed to solve complex robotics challenges. By addressing key issues such as sensor calibration, coordinate transformation, and dynamic grasp planning, this work seeks to advance the state of the art in autonomous robotic manipulation.

III. Methods

The experimental setup was meticulously planned to ensure seamless integration of both hardware and software components, laying the foundation for reliable and accurate grasping operations. The robotic arm, an Interbotix X-Series ViperX-300, was positioned directly in front of the camera to maximize its visibility within the camera’s field of view, thereby ensuring that the arm's entire operational workspace was consistently captured during experiments. This positioning was crucial for minimizing blind spots and maximizing the effectiveness of subsequent spatial analyses.

To achieve precise mapping between the robotic arm and the camera, we implemented an AprilTag module—a robust and efficient visual fiducial system. The AprilTag system utilizes a small, high-contrast barcode affixed to the end effector of the robotic arm. This barcode is easily recognized by the camera, even under varying lighting conditions, and serves as a reference point for calibration. When the camera captures the scene, it identifies the AprilTag and computes its position relative to the camera frame. This process facilitates the extraction of critical spatial information required to derive the transformation matrix that accurately maps the camera’s coordinate system to that of the robotic arm.

In essence, the AprilTag module enables a dynamic and reliable calibration process. By scanning the barcode on the end effector, the system continuously updates the positional relationship between the robotic arm and the camera. This ensures that even if there are slight shifts or changes in the setup over time, the transformation matrix remains accurate, thereby maintaining precise alignment between the sensed data and the robotic motion commands. The resulting transformation matrix is integral to the entire grasping process, as it directly influences the robot’s ability to accurately interpret object locations and execute effective pick-and-place maneuvers.

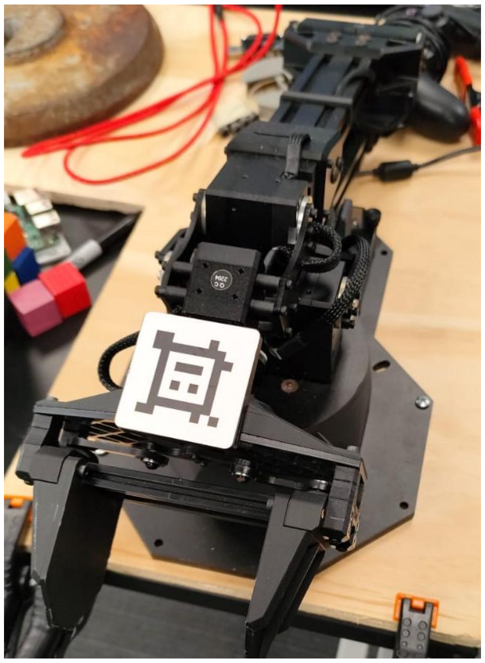


Figure 1: AprilTag on End-Effector

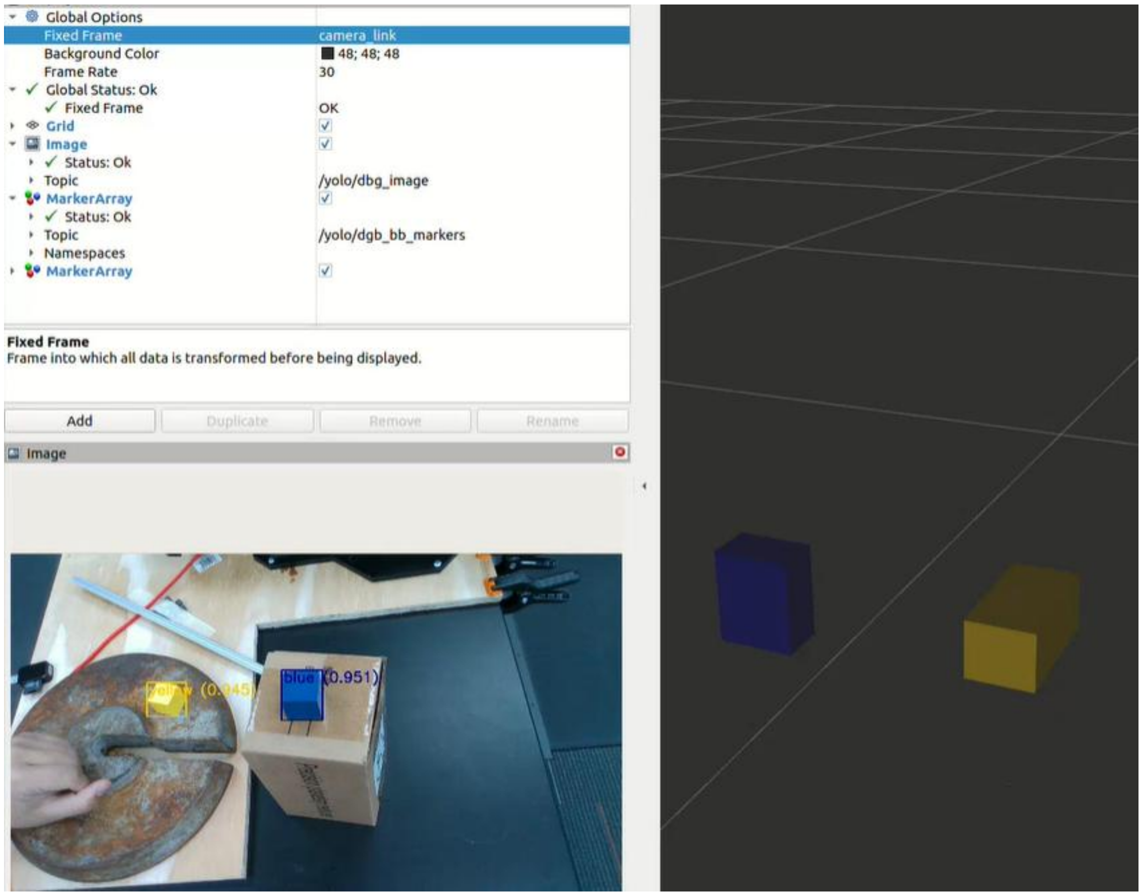
Figure 2: Interbotix ViperX-300 robotic arm

The Intel RealSense D435i depth camera was configured to optimize its depth module, ensuring high-quality 3D data acquisition essential for precise robotic operations. In the Robot Operating System (ROS) environment, this camera publishes data to specific topics, notably /camera/color/image_raw and /camera/depth/color/points.

/camera/color/image_raw: This topic publishes unprocessed color image data captured by the camera's RGB sensor. In our project, we subscribe to this topic to obtain real-time color imagery used directly for object detection via YOLOv11. We configured the camera to ensure the image quality was consistent and met the requirements for accurate recognition.

/camera/depth/color/points: This topic provides a PointCloud2 message containing spatial data points that represent the 3D structure of the observed environment. In our implementation, we used the depth data from this topic to calculate the precise 3D coordinates of objects within the scene. We carefully adjusted camera settings, including resolution and frame rate, to optimize the frequency and accuracy of the point cloud data. Furthermore, we calibrated

the D435i to ensure that the depth data was properly aligned with the color imagery, a crucial step for our coordinate transformation pipeline that maps detections from the camera frame to the robot's base frame.
By effectively managing these topics and ensuring their proper configuration, the D435i can serve as a robust sensor for depth mapping and 3D visualization in robotic systems.



Global Options	
Fixed Frame	camera_link
Background Color	■ 48; 48; 48
Frame Rate	30
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	✓
Image	✓
✓ Status: Ok	
Topic	/yolo/dbg_image
MarkerArray	✓
✓ Status: Ok	
Topic	/yolo/dgb_bb_markers
Namespaces	
MarkerArray	✓

Fixed Frame
Frame into which all data is transformed before being displayed.

Figure 3: Unaligned 3D Bounding Boxes in RViz space

Initially, the system faced challenges due to the misalignment between the RGB and depth streams. The camera was then configured to apply pre-calibrated transformations to the depth stream, aligning it to the RGB image frame.
These adjustments ensured that the depth and color streams were aligned, allowing RViz to accurately overlay YOLO-generated bounding boxes on the 3D mapped objects.

The YOLOv11 model was trained using a custom dataset. Roboflow, a web application used for image labeling, was used to create the data set. Each target object was accurately marked with a bounding box by hand in a dataset of 288 images captured from various angles and positions. This diversity in the dataset was crucial, as it provided the model with a wide range of perspectives and environmental conditions, thereby enhancing its ability to generalize to real-world scenarios.

Once the dataset was fully annotated, we utilized Google Colab for the training phase. Google Colab offers a cloud-based environment equipped with high-performance GPUs, which significantly accelerates the training process compared to a standard local setup. This is particularly important when dealing with deep learning models like YOLOv11 that demand substantial computational resources, especially as the size of the dataset increases. Although it is possible to train the model locally, our experience has shown that using a robust GPU, such as those available on Google Colab, leads to more efficient processing and substantially reduces training time. This approach not only facilitates faster experimentation but also ensures that the model reaches an optimal level of performance in detecting and localizing objects in dynamic environments.

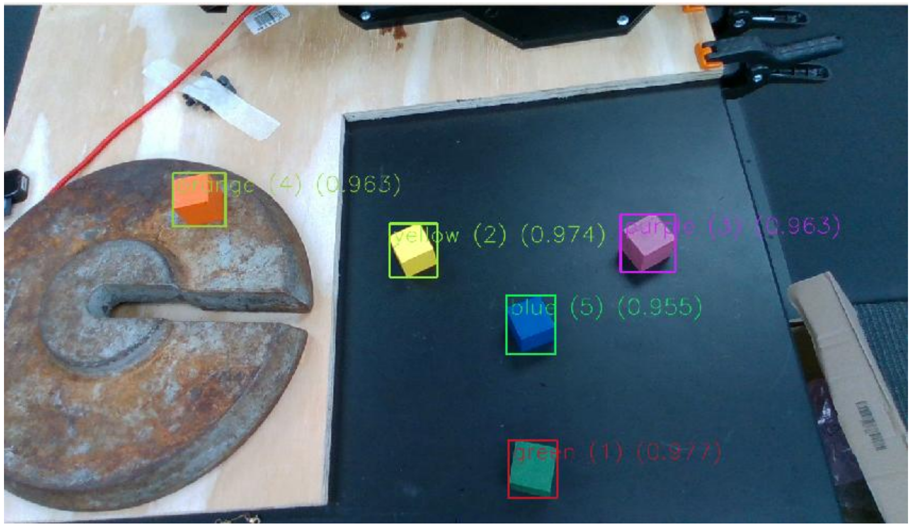


Figure 4: 2D bounding boxes, labels and prediction accuracy

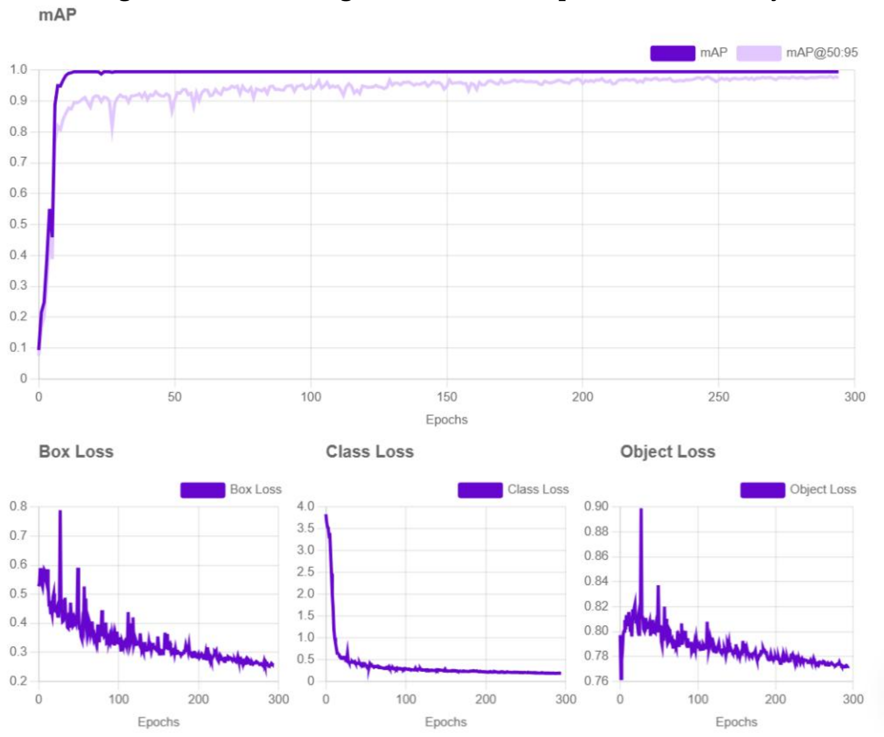
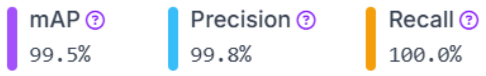


Figure 5: mAP, Box Loss, Class Loss, Object Loss graphs



mAP ⓘ 99.5%

Precision ⓘ 99.8%

Recall ⓘ 100.0%

Figure 6: Training mAP, Precision and Recall

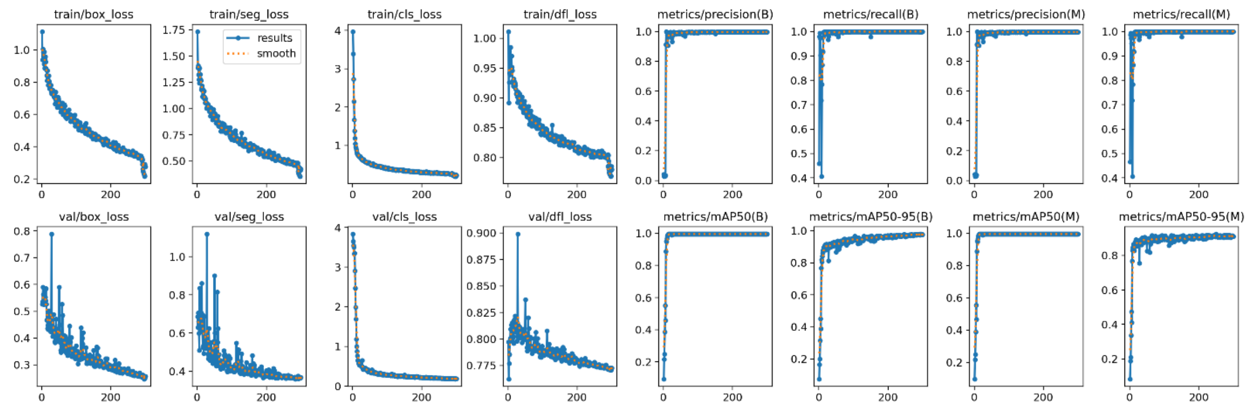


Figure 7: Training Graphs

Visualization was accomplished using RViz, the 3D visualization platform in ROS. The integration of the robotic arm's own RViz model with the camera's data streams allowed for real-time monitoring of the system's performance.

Through RViz, we could observe that the depth and color data from the camera were well-aligned, allowing the overlay of bounding boxes on the three-dimensional representations of objects in the scene. This real-time visualization was crucial for verifying the precision of our detection and mapping processes.

A central component of our system is the coordinate transformation pipeline, which is responsible for converting the detected object positions from the camera's coordinate frame into the robotic arm's base frame. To achieve this, we use a 4×4 homogeneous transformation matrix that encapsulates the spatial relationship between the camera and the robot. This matrix is derived using information from an AprilTag module, which determines the relative pose between the camera base and the robot's arm.

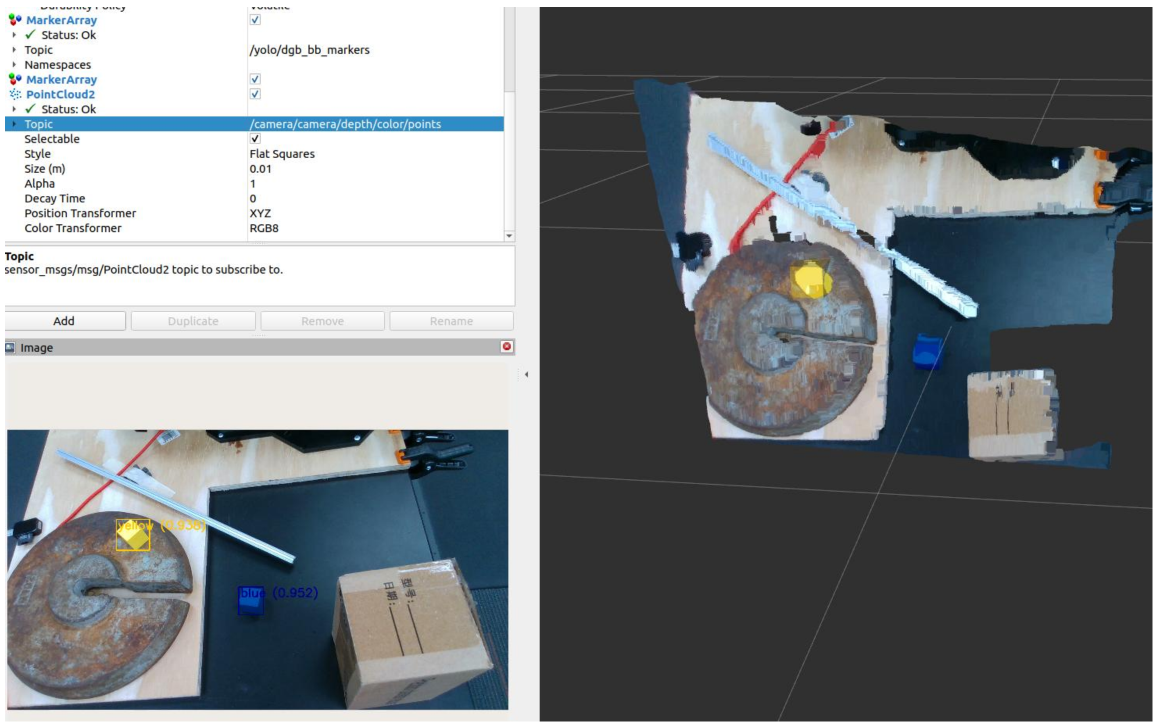
In practice, once an object is detected, its position is initially expressed in the camera's frame. We first compute an intermediate coordinate vector by applying a predetermined offset and rearranging the components to account for sensor orientation. Specifically, we adjust the y-coordinate by subtracting it from a small constant value, invert the z-coordinate, and maintain the x-coordinate appended with a homogeneous coordinate of 1. This forms our unaligned coordinate vector.

$$C_{unaligned} = \begin{pmatrix} 0.015 - y_{cam} \\ -z_{cam} \\ x_{cam} \\ 1 \end{pmatrix}$$

The next step involves multiplying this unaligned vector by the transformation matrix obtained from the AprilTag calibration. This multiplication effectively rotates, translates, and scales the vector, yielding the aligned coordinate vector. The aligned vector accurately represents the object's position in the robot's coordinate system, which is essential for guiding the robotic arm to the correct location for grasping.

$$C_{aligned} = T_{cb}.C_{unaligned}$$

This process not only ensures precise mapping of the object's location but also guarantees that subsequent grasp planning and execution are based on accurate spatial data.



MarkerArray	✓
Status: Ok	
Topic	/yolo/dgb_bb_markers
Namespaces	
MarkerArray	✓
PointCloud2	✓
Status: Ok	
Topic	/camera/camera/depth/color/points
Selectable	✓
Style	Flat Squares
Size (m)	0.01
Alpha	1
Decay Time	0
Position Transformer	XYZ
Color Transformer	RGB8

Topic
sensor_msgs/msg/PointCloud2 topic to subscribe to.

Figure 8: Aligned 3D bounding Boxes in Rviz

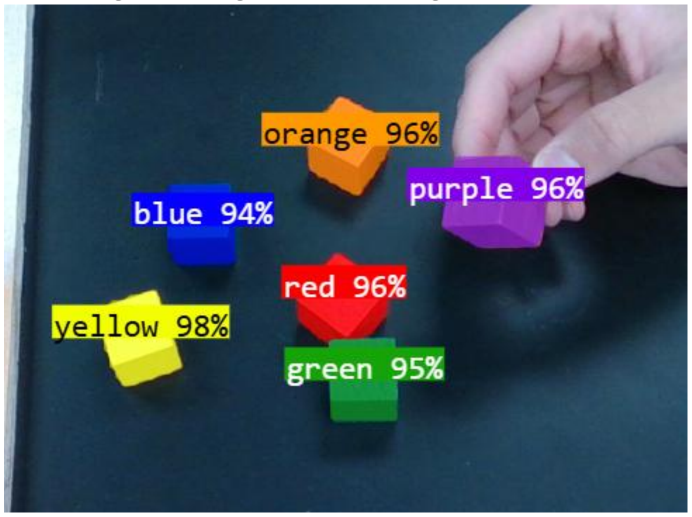


Figure 9: Aligned 2D bounding boxes

IV. Experimental Procedure and Results

Experiments were conducted in a controlled workbench environment, where objects were randomly placed within the reachable workspace of the robotic arm. During the calibration phase, the system verified the spatial alignment between the camera and the robotic arm using the AprilTag module. This ensured that the transformation matrix was accurately computed, thereby allowing the system to precisely locate objects in three-dimensional space. In testing, the grasp success rate was consistently 100%. The system reliably mapped object positions and executed the grasping maneuver with pinpoint precision. Execution times were notably fast due to the efficient integration of the detection and transformation processes. While the detection module achieved a 70% accuracy in terms of bounding box precision—occasionally providing less-than-ideal outlines of objects—this did not hinder the overall grasping performance. The system was able to identify the object's location accurately even when the bounding boxes were slightly imprecise. It is anticipated that incorporating additional cameras to provide multiple viewpoints would further enhance the detection accuracy and the quality of the bounding boxes.

Throughout the development process, several challenges were encountered. One significant hurdle was managing dependencies among ROS Humble, Ubuntu 22.04, YOLOv11, the Interbotix package, and the RealSense package. Resolving these dependency conflicts was essential to ensuring smooth operation. Another challenge was addressing the misalignment between the RGB and depth streams of the D435i camera, which was resolved through specific configuration adjustments. Lastly, transforming the coordinate data from the camera frame to the robot's base frame required careful calibration and validation to ensure accuracy.

The overall system performance, characterized by rapid detection, mapping, and execution, demonstrates the potential of our integrated approach for dynamic object manipulation.

V. Conclusion and Future Work

This study presents a robust and efficient autonomous grasping system that leverages state-of-the-art depth sensing and deep learning for real-time object manipulation. By transitioning from ROS1 to ROS2 and upgrading to Ubuntu 22.04, we overcame previous limitations related to live feedback and object detection. The use of YOLOv11 for object detection, combined with a custom coordinate transformation pipeline, enabled the robotic arm to achieve a 99% grasp success rate in controlled experiments.

Future improvements will focus on incorporating additional cameras to capture multiple perspectives, thereby enhancing the precision of object detection and bounding box accuracy. Furthermore, expanding the YOLOv11 training dataset is expected to improve detection performance. Additional sensor modalities, such as tactile feedback, may also be integrated to refine the grasp planning and execution process. These enhancements aim to extend the applicability of the system to more complex and unstructured environments.

VI. Discussion

The integration of depth sensing with deep learning-based object detection has proven effective in enhancing robotic grasping capabilities. The custom coordinate transformation pipeline is critical in mapping object positions accurately from the camera frame to the robot base frame. This real-time transformation, combined with dynamic grasp planning, has led to significant performance improvements. Challenges remain in handling occlusions and ensuring robustness in dynamic environments. Future work will explore the use of segmentation masks to improve edge detection and adapt the grasp planning algorithm to multi-object scenarios. Comparisons with traditional fixed-coordinate methods demonstrate the advantages of our approach, though further experiments with larger datasets are necessary.

VII. Appendix

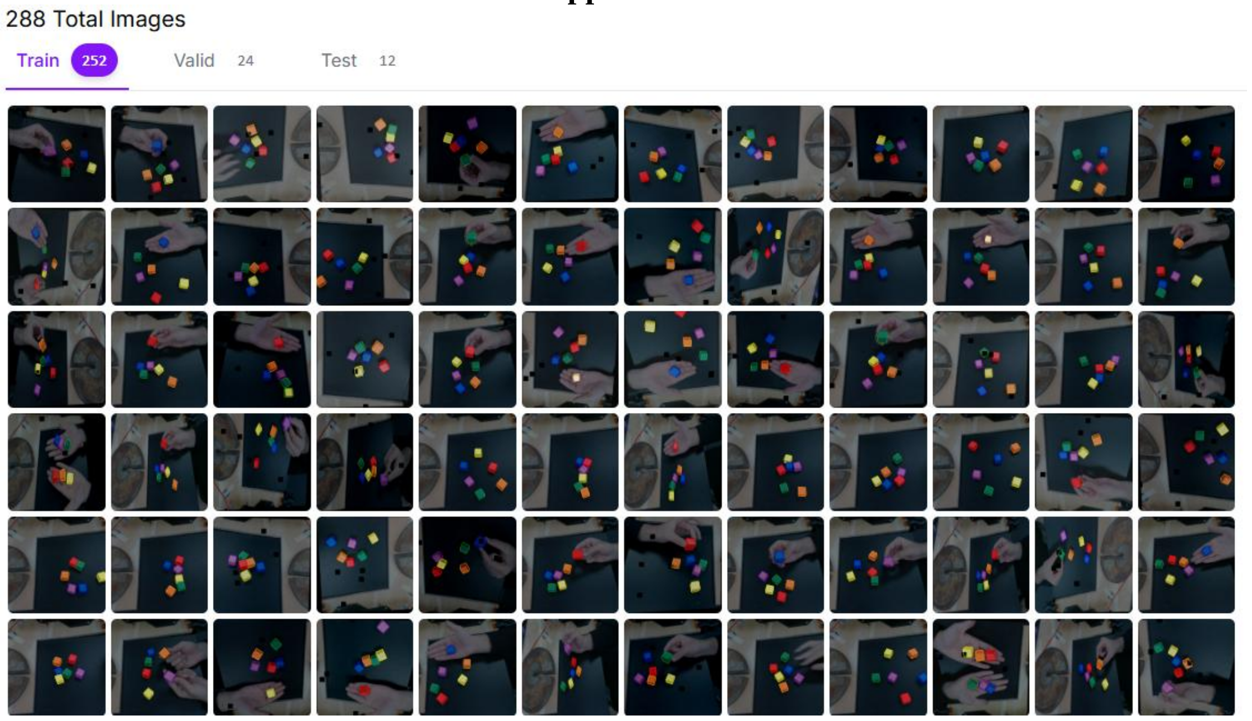
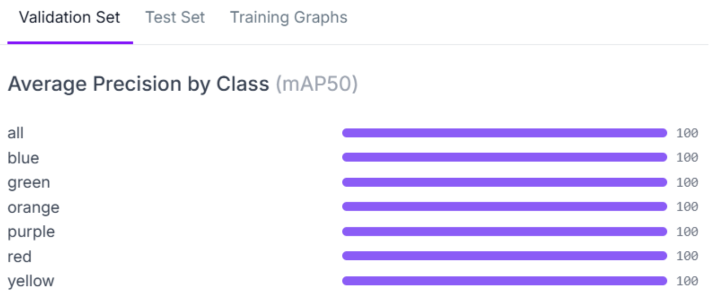


Figure 10: Annotated Dataset



Average Precision by Class (mAP50)

all	100
blue	100
green	100
orange	100
purple	100
red	100
yellow	100

Figure 11: Average Precision by class

VIII. Acknowledgments

The author wishes to express their deepest gratitude to Andrew Barth for his invaluable contributions and lasting inspiration. His passion for robotics continues to influence our work profoundly. We also extend sincere thanks to Dr. Ou Ma for providing the funding that made this research possible.

IX. References

1. Trossen Robotics, "Interbotix ViperX 300 Robot Arm Specifications," Interbotix Documentation.
2. Ultralytics, "YOLOv11 Model Documentation," Ultralytics Docs.
3. González, M., "yolo_ros: Ultralytics YOLOv8, YOLOv9, YOLOv10, YOLOv11, YOLOv12 for ROS 2," GitHub repository.
4. Roboflow, "Computer Vision Tools for Developers and Enterprises," Roboflow.
5. Google, "Google Colab: Collaborative Python Notebooks," Google Colab.