GENG4412 Engineering Research Project Part 2
Final Report

Humanoid Object Recognition, Localisation & Manipulation

Caleb Hodgson-Taylor
23131651
School of Engineering, University of Western Australia

Supervisor: Thomas Braunl
School of Engineering, University of Western Australia

Word count: 7990

School of Engineering
University of Western Australia

Submitted: 20 May 2026

Project Summary

For a human robot to perform everyday physical tasks, it requires the ability to perceive, locate and manipulate small objects. Whilst, object detection, 3D localization and grasp planning exist independently, integrated and open-source combinations for a humanoid platform remain scarce. This project provides a modular ROS2 pipeline, using toned YOLO models for open and closed detection, alongside RGB-D localisation, geometric pose estimate and inverse kinematics; this provides a viable basis for novel object detection, localising and grasping objects for a humanoid robot, completely runnable on just CPU.

Detection and segmentation of known (closed-set) and unknown (open-set) objects was required. Functionality for 3 tuned model types were created for YOLO26, outside the base model that detects 80 COCO objects. YOLO26E text prompt detects novel objects and ignores unspecified ones; many intuitive classes weren't detected, so SAM3's accuracy and scope should be tested in future works. YOLO26E picture prompt detects objects based on reference images and its 2D bounding box showing high accuracy, which diminished for changing orientation and lighting; its 2D mask had spill past object bounds. YOLO26 model was custom trained for toothpaste, cereal box and torch, showing strong detection and mask accuracy with more versatility to environmental change after strong augmentations during the training process.

A 2D mask was back-projected into a 3D point cloud and noise was filtered out; a trade-off existed between noise avoidance and mask quality, causing slightly imperfect masks to be passed. Important as principal component axis (PCA) for orientation and oriented bounding box (OBB) for dimensions could deviate under poor mask quality and untraditional object shapes. Issac Pose estimation models accuracy should be tested in future works.

A right pre-grasp point was chosen with the secondary component being its approach axis and tertiary component being wrist offset; this worked for upright objects. To extend to any object shape, grasping could be with both hands at the right and left most object axes concurrently. The recommended solution would be to use a learned grasp pose detector based on the 3D mask instead. Pinocchio inverse kinematics solved joint positions to arrive at grasp point with safe poses to avoid collisions and grasping based on predefined position or pressure threshold. An 83.3% grasp accuracy with 0.1cm mean deviation and 4.5cm max deviation was found for 3 objects tested 10 times each for unique positions. This pipeline should be combined with SLAM map navigation to explore its environment, YOLO to detect objects from a distance and then walk towards the object before enacting the grasping pipeline. This would create a robot capable of advanced practical tasks, like grocery shopping.

Table of Contents

Contents

List of Figures

List of Tables

Nomenclature

Abbreviation	Definition
YOLO	You Only Look Once
VLA	Vision-Language-Action
IK	Inverse Kinematics
PCA	Principal Component Analysis
OBB	Oriented Bounding Box
PCA	Principal Component Analysis
ROS	Robert Operating system
DoF	Degree of freedom
COCO	Common objects in context
RGB	Red Green Blue
RGB-D	Red Green Blue Depth
SAM	Segment anything model
JSON	Javascript object notation
URDF	Unified robot description format
GPD	Grasp pose detection
COCO	Common objects in context

1. Introduction

Humanoid robots can complete any human task that is undesirable for the user to complete. For example, Elon Musk stated, “It should be able to go to the store and get the groceries. I think we can do that. If we don’t make it, someone else will.” [1]. But the major difficulty in tasks — such as picking up groceries, cleaning, bartending — is that it requires object and scene detection to act correctly within a task. Traditional methods in autonomous vehicles focus on delineating human, vehicle, and infrastructure detection to dictate a binary go or stop; humanoid robots deal with small objects which require precise information on their shape, orientation and location in 3D space relative to the robot. Current attempts usually use teleoperated robots or propositioned objects that can’t respond to a dynamic environment with small objects that are harder to segment than buildings or humans. Whilst methods for closed-set or open-set detection, localisation and orientation exist independently, combinations of all are sparse, non-open source, to non-existent. This report hypothesizes that a modular ROS2 pipeline with tuned object detection models, RGB-D localisation, geometric pose estimation and inverse kinematics provides a basis for detecting, localising and grasping selected objects with a humanoid robot.

The project’s stack provides a hybrid closed/open detector that works on both known and never-seen objects, creating a bridge for future complex robotic tasks. The underlying system can benefit the economy through increased business efficiency in countless industries and benefit the public through more free time from undesired activities; for example, it could be used in grocery shopping, collecting packages or selecting the correct tool. This benefits future researchers by providing a modular ROS 2 GitHub application that allows for the selection of a detection model for known or unknown objects, alongside the retrieval of a desired object’s 3D mask and pose for grasping; sections of this can be used based on the researcher’s requirements with the ability to make future alterations based on their own needs.

1.1 Objectives

To achieve this end requires the fulfilment of the following objectives:

- Capacity to detect and segment all objects of a known (closed set) classification in the robot’s vision with high accuracy and low-speed requirements
- Detect and segment all objects of an unknown (open set) classification in the robot’s vision with moderate accuracy and moderate speed requirements
- Create an accurate 3D mask for the detected object that minimizes noise.
- Find the object’s general 3D orientation and dimensions.
- Calculate an effective grasp position for the object, relative to the robot’s coordinates.
- Grasp the object in an accurate and safe manner.
- All nodes must run concurrently without computationally exceeding hardware limits.

1.2 Literature Review

Early object detection was based of colour thresholds in HSV or background subtraction (anything deviating from a learned background was defined as a foreground object) [2, 3]. Shape analysis followed extracting features from multiple window scales to be compared with an object's features (ex. HOG + SIFT) [4]. Deep learning revolutionized detection, using convolutional networks that are trained to detect specific objects through back-propagating the difference between prediction and desired outcome; YOLO1 released in 2015, providing bounding boxes and confidence intervals [5]. COCO is a large-scale dataset of 80 object classes commonly used for benchmarking computer vision models; it forms the detectable classes for the base YOLO model [6]. Most recently, open-vocabulary models have allowed open set detection through aligning images and text prompts in a vast training database creating matching image-text pairs that are closer in the shared embedded space; YOLOE computes cosine similarity between region features against prompts [7].

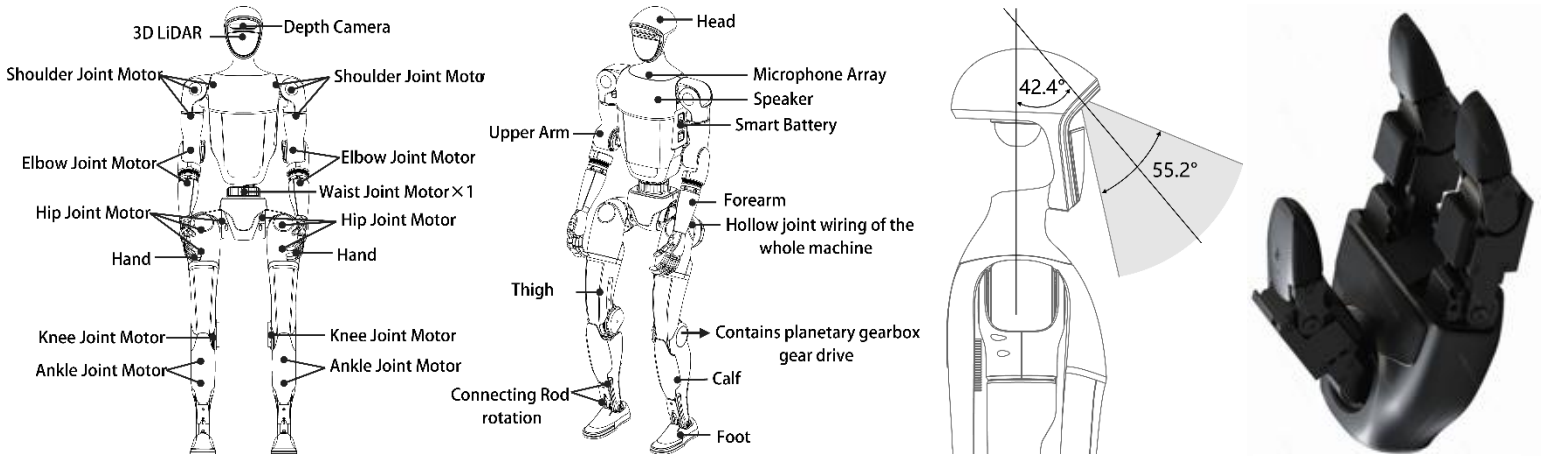
1.2.1 Real world approaches

Telsa Optimus has used top-down visual segmentation (model from Telsa cars), and Boston dynamics has used closed set depth-based segmentation but is transitioning to a large behaviour model, although both have nice demos the utility for this project is limited, as due to the proprietary nature they don't release pipelines or models [8, 9]. RT-2 comes from a large vision-language model applied to web image-text pairs and then is co-fine-tuned on robot demonstrations allowing it to generate sequences of action tokens from camera images and a text goal [10].Amazon sparrows use barcodes and suction devices to package warehouse goods; this exemplifies a limitation in using the unitree G1 Dex3-1 hands [11]. Agility Robotics' Digit uses RealSense depth cameras for object localisation proving their utility. The combination with lidar shows future combinations of sensors for more complex tasks [12]. The most relevant example to this project is the Toyota research institute, who used YOLO5 to detect and segment objects within a shopping centre, they then get a 2D segmentation (using U-Net style model) before back-projecting with depth to achieve an object point cloud; This was used to estimate ideal grasp position (using PointNet Embedding network) and whether suction or gripper should be used, before uses inverse kinematic to plot a collision free path to that position [13]. This is practically identical to the ideal pipeline specified in the project process with the major difference being the extensive new models through every step that make the potential results much more accurate and dynamic. The best existing approaches for closed/open set detection, 3D pose and grasping will be examined in 2.3 ideation, along with conclusions of which ones the project will go for, based on their strengths and weaknesses to the project's objectives and constraints.

2. Design Process

2.1 Constraints

This project worked on the Unitree G1 humanoid robot with 29 joints; this adds greater complexity than traditional industrial robot manipulators, as it accounts for balance, joint limits, besides collisions with itself and the environment, such as the table that holds the object. It has a D435i depth camera, angled downwards 47.4° by default and a depth sensing range of 0.3-5m [14]. This can be angled to face straightforward, but the default angle is better for observing objects below. Unlike overhead or eye-in-hand camera angles, it requires a complex transform from camera to torso, which can add inaccuracies. It has Dex3-1 three-fingered hands. This differs from a traditional parallel-jaw gripper, 5 finger anthropomorphic hand or suction end effector. Adding novel limitations and complexity to how an object can be gripped [15].



(a) Unitree G1 Important Points (B) D435i Angle (C) Dex3-1 Hand

Figure 2.1: Unitree G1 Physical Parts [18]

The Unitree G1's PC1 was inaccessible; problematic, as that's where camera data travels internally. To access camera data, the robot's head case must be removed, connecting to laptop by USB-C. To SSH into PC2 wasn't achievable on the researcher's laptop, meaning ROS nodes couldn't be run locally on G1, thus its powerful GPU couldn't be used for computation. For robot movement, an Ethernet cable was used to read and send robot joint position with wireless being introduced in the last weeks of operation.

Researcher had access to Dell XPS P103G [16], which was used for all computational requirements. Relevant as it lacks a Nvidia GPU, limiting what software can be used. It's lacking computational power limited the number of nodes that could be run concurrently and their duration of running, requiring solutions that limited computations to what's required. The laptop runs on Windows, but robot operation requires Linux thus, the laptop was dual booted with Ubuntu.

The robot was mostly limited to its storage room, which limited environmental variables, such as lighting, background, obstacles that could affect operation. Operation time on robot could be limited by other group members, so other avenues of testing were important to set up.

2.2 Design Criteria

- **Modularity:** The entire system should be split into sections that work independently of each other. One node can be used without another, allowing solitary testing of its function. A node could be expanded or replaced without affecting prior node function. This separation is based on unique functional stages, having distinct inputs and outputs that connect the nodes together into a greater program.
- **Configurability:** The nodes should be able to have their functions changed by many command line parameters, allowing user level change based with no need to change the underlying code.
- **Detection Flexibility:** The system should be able to detect all object classes the user desires, whether they be known or unknown. There should be as many mechanisms as possible to develop a model that satisfies this requirement. This detection should be translatable to many environments.
- **Detection Accuracy**: The detected object should have the correct class, mask and high confidence.
- **Localisation Accuracy:** The 3D mask, PCA, OBB and grasp position should mimic what's expected of reality with flexibility to many types of objects.
- **Grasp Accuracy:** How close the grasp algorithm is to successfully wrapping around object, measured by its offset. Consistency, measured by its successful grasp percentage for a range of objects.
- **Computational Efficiency:** The design should run on existing hardware. Computationally efficient enough to avoid slowdowns or crashes.
- **Speed**: Each step should operate quickly enough for frequent frames of iterations to operate in a real time capacity.
- **Safety**: Operation should pose no danger to the robot itself or that which surrounds it.
- **Debuggability:** Each node should have visualizations and text corresponding to intermediary steps and outputs to compare to desired.

2.3 Ideation Process

2.3.1 Closed set detection

For closed-set detection, there were RT-DETR and YOLO12, but over the course of the project they got outperformed by the new model releases of RF-DETR and YOLO26. RF-DETR reports stronger accuracy benchmarks than YOLO26 [17]. The problem is that its low-latency performance requires an Nvidia GPU, whilst YOLO already has an existing ROS2 [18]. The problem with these closed-set detectors is that only work for the 80 COCO defined object classes [19]; A way to add more classes is to custom train your own models or use open set detection.

2.3.2 Open set detection

For text prompted detection, there is YOLO26E and SAM3. Sam3 is more accurate than YOLOE, but it's much slower on CPU, taking many seconds for a single image detection, requiring a Nvidia GPU [20]. For image prompted detection, the alternative to YOLO26E is OWL-Vit/OWLv2, but it doesn't have ROS2 support or provide segmentation masks [21]. Traditional feature mapping is a more basic version that is even more fragile under changing conditions [22].

2.3.3 Pose

For object rotation and size, there is PCA + OBB and Isaac ROS pose-estimation models. Isaac Pose has higher accuracy but requires a Nvidia GPU, in addition to an object mesh or trained model for the target object class [23]. Principal component axis (PCA) gives the object its 3 principal axes, ranked by spread in each direction. Oriented bounding box (OBB) gives a 3D bounding box, which dictates object size relative to PCA's new axes. They work for novel objects, whilst remaining computationally cheap and easily visualised.

2.3.4 Grasping

Vision-language-action models (VLA) are the other major manipulation pipeline, taking camera data, text commands and/or robot state to predict next robot action, traditionally trained through imitation or reinforcement learning [24]. VLAs weren't chosen as they struggle heavily to generalize for new objects, robots or scenarios outside their training data, whilst being harder to debug and requiring huge datasets [25].

For grasping based on object localisation and geometry, there are learned grasp pose detection methods or geometric grasping approaches. For learned, it traditionally predicts 6-DoF grasp poses based on the object's 3D point cloud. Grasp Pose Detection (GPD) [26] would be most feasible, because of its ROS wrapper and ability to run on CPU; its problematic being for two-finger parallel-jaw gripper on stationary industrial robot manipulator. Ideally requires two cameras from different angles for better 3D point cloud, whilst computationally heavy and only confirmed to support Ubuntu 16.04, which clashes with yolo_ros's 22.04 requirements. In comparison, geometric grasp point uses object main directions and size, adding offset along grasp direction. This is easily integrable, flexible to tuning and computationally cheap.

2.4 Software Tools

Ubuntu (22.04): a Linux based development environment with the Ubuntu distribution was used, because of its easy integration and proven compatibility in previous projects [25, 27, 28].

ROS2 (Humble): Robert Operating System (ROS) is a robotic middleware that helps different robot programs communicate. It provides nodes to separate program tasks, topics to communicate between nodes, debugging tools and existing packages to use. ROS2 is the newer generation with better functionality and compatibility with new packages [29]. ROS2 has 3 main distribution choices, Foxy for ubuntu 20.04, Humble for 22.04, Jazzy for 24.04; foxy isn't supported by newer packagves, whilst Jazzy isn't supported by older packages, so long-term supported humble was chosen [30].

Real Sense ROS Camera Wrapper: Gets D435i RGB camera and depth data, outputting as ROS nodes [31]. **yolo_ros:** a ROS2 wrapper for YOLO models from Ultralytics [32]. This allows YOLO models to run within the ROS2 pipeline, subscribing to camera topics and publishing detection information. It provides extensive parameter configurability. The debug image with object class, 2D mask, detection confidence is especially useful for testing. It provides tracking for detected object across frames and rudimentary 3D information. This provides utility that saves development time and forms a good base for modifications.

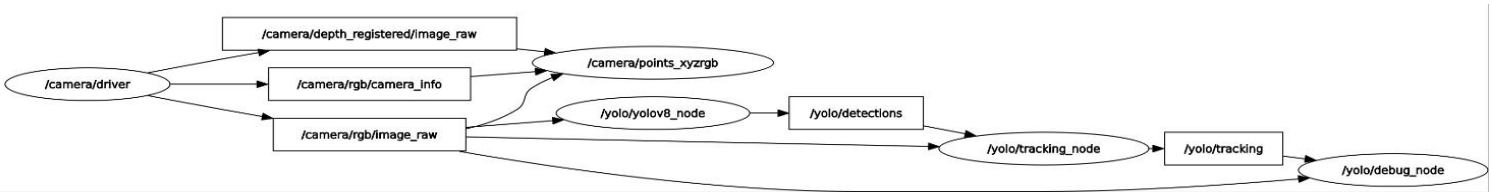
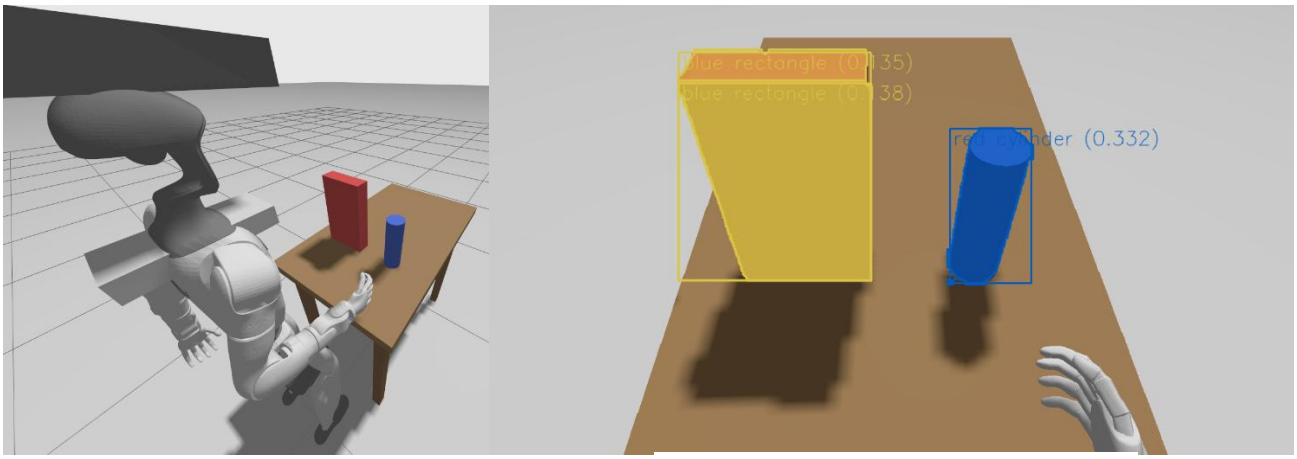


Figure 2.2: yolo_ros node graph [1]

SDK2_Python: Sdk2 is Unitree's official software development kit for communicating with and controlling unitree robots. SDK2_Python was chosen to match the pipeline code, which is python based. It provides faster development speed and modification during debugging than the C++ implementation [33].

Gazebo: Is a robotics simulation environment with other main options being Mujoco and Issac lab. Unitree_MuJoCo has greater joint accuracy and translatability to the real world G1, but it has limited camera and ROS2 support [34]. Issac Lab is more powerful but requires a Nvidia GPU. Gazebo had direct ROS2 support [35], linking to yolo_ros outputs. To set up the gazebo environment, a URDF for the Unitree G1 29 DoF model [36] was downloaded. A harness was created to suspend the robot. A table and basic object shapes were created, as more complex object meshes couldn't load in. An RGB-D camera was spawned with its position and rotation, mimicking the actual D435i and added to G1's URDF model. The G1's movement was controlled by gz_ros2_control [37]. The simulation was key to ensuring movements were safe and generally correct based on translatable inputs and outputs, but it had many problems that forced a quicker move to tests in reality. The model didn't come with collision geometry and instead of 3 fingers, it was 5 that couldn't close. Color information for detection was flipped and camera to torso transform was consistently off.



(a) Unitree G1 & Objects (b) YOLO Detected Objects

Figure 2.3: Gazebo Simulation Environment

Pinocchio: Inverse kinematics (IK) calculates the joint positions for a robot end-effector to reach a desired position, and sometimes orientation [38]. Moveit2 is the most complex and complete method providing IK, collision checking, motion planning, scene management and trajectory managements [39]; the problem was the setup assistant consistently failed, crashing for G1's URDF. Even if working, it adds much more complexity, whilst being traditionally designed for fixed-base industrial arms. Pinocchio was chosen for IK, as its computationally cheaper and easily integrates into Python movement scripts [40]; this requires collisions to be manually avoided through pre-determined poses.

2.5 Standards & Compliance

ISO 10218-1:2025 informed the design because it addresses hazards associated with robot motion and provides requirements for reducing risk during robot operation [41]. These principles informed slow movement with minimal force, robot suspension, localised arm activation, emergency damping procedures, workspace bounds checking, simulation first testing and confirmation prompts before execution. Formal certification would require formal force measurement, safety-rated emergency-stop validation, and full industrial safety certification, which were outside the project's scope.

2.6 Tests & Experiments Undertaken

ROS2 comes with RVIZ2 and RQT used to see ROS topic outputs. RVIZ2 provides visualization of video streams and points on a 3D graph. RQT shows the topic's text or number output. They allowed examination of ROS topic outputs to ensure compliance with expectations and the creation of debugging ROS topics purely for visualization, which are more intuitive than a spreadsheet of numbers, making verification easier.

RVIZ visualised the YOLO debug image, ensuring the detection class, localisation, and 2D mask were accurate for each designed YOLO model in different environments. The 3D point cloud was visualized in RVIZ's 3D graph with coloured markers indicating PCA, OBB and pre-grasp points being shown alongside, allowing quick comparison with desired. The final pipeline was qualitatively tested for 3 objects in multiple workspace positions, with observed grasp success, palm-to-object offset and recorded failure reasons being examined.

2.7 Health & Safety

Safety was prioritized during robot operation. The humanoid was kept suspended during grasping to ensure balance was maintained; actions were minimized to right arm only, reducing potential risks from other joints. The controller's damping button was always ready to be pressed by the operator. If pressed, the robot's joints go slack. All joints' commanded forces and velocities were minimised to reduce risk severity. Workspace bounds and clearance poses were added to ensure invalid positions couldn't be reached and object collision could be avoided. Before any real-world test, it was tested within the simulation to ensure correct behaviour. On program start, it shows the estimated joint position and transitions through each stage, requiring clicking Enter for the operation to begin. Tests were done in stages, from testing joint angles to setting ready pose to quarter/half movements before IK was tested before full. The robot operation was limited to a controlled workspace at a table, away from any people or non-placed objects. Grasping stops either at pressure or predetermined grasp limit, ensuring hand closure at the target doesn't damage grasped object.

3. Final Design, Results and Discussion

3.1 System Overview

The final design of the project was through ROS2 nodes. Their topics were connected to each other as shown in Figure 3.1. RealSense camera node extracted D435i information into ROS topics, the RBG and color aligned depth was passed to YOLO ROS. Different types of Yolo segmentation models could be passed, the node had existing functionality for YOLO26 with its 80 COCO objects and YOLO26E unprompted. Functionality was added for YOLO26E text prompts with specified objects classes to detect, YOLO26E picture prompts with a PNG containing the specified object and a user defined 2D bounding box of its pixel position in said photo; in addition, a custom trained YOLO model for 3 objects toothpaste, torch and cereal box could be passed.

The YOLO ROS node detected all objects with the highest confidence object's 2D mask transforming to the 3D point cloud. The 2D point cloud was back-projecting to 3D and the mask was cleaned, before passing to PCA + OBB. PCA calculates principal, secondary and tertiary axes, along with the oriented bounding box along these coordinates being passed to the right pre-grasp point. Right pre-grasp point, creates two points offset along the secondary axis from the OBB edge, choosing the right-most point, before adding a wrist offset (10cm) for the tertiary axis. This point is transformed from camera coordinates to torso-link coordinates, feeding into the movement script. This moves to desk clearance, then ready joint positions. It then calculates inverse kinematics, finding joint positions required to reach passed right pre-grasp position. Once, there it closes Dex3-1 fingers until pressure threshold is reached, or the predefined grasp limit is reached. Each movement command is being published to SKD2_python for Unitree G1 Movement.

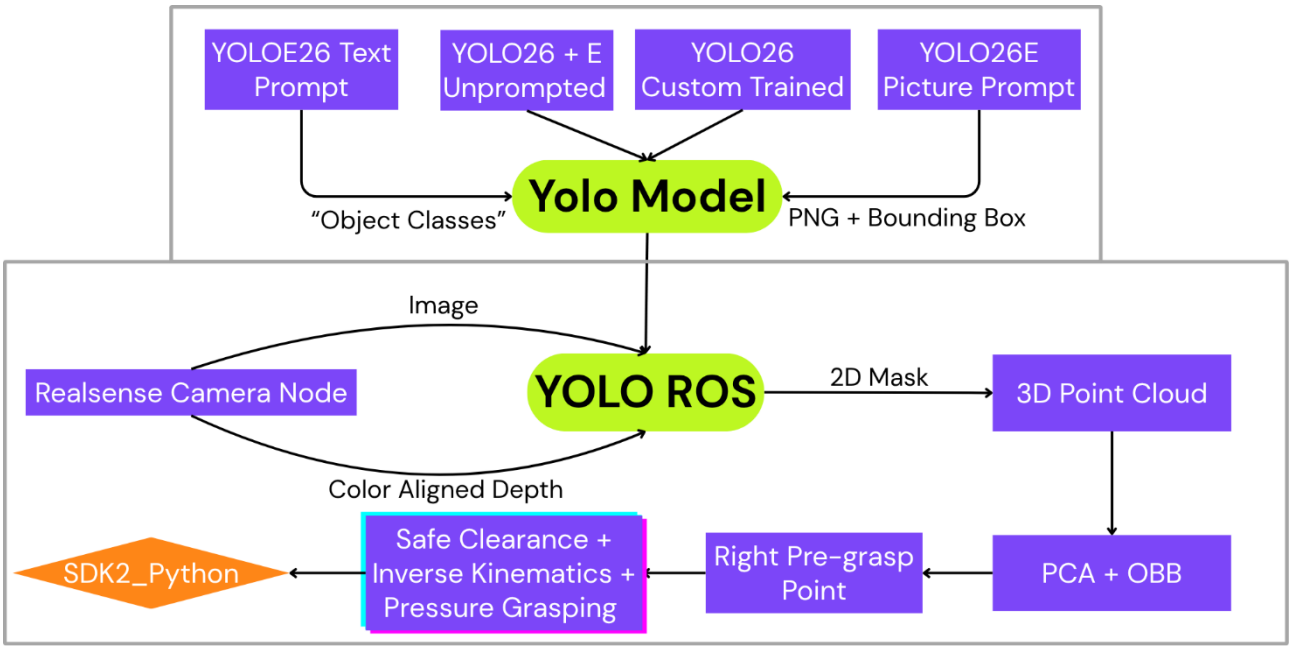


Figure 3.1: Flowchart of system architecture

3.2 YOLO ROS Parameter Tuning

The yolo_ros node had many parameters that needed tuning for optimal performance. Image width and height defaults to 640 x 480, an aspect ratio of 4:3, whilst D435i is 1920 x 1080, an aspect ratio of 16:9 [42]; This letter-boxes window, adding padding to top and bottom of the image. Unused resolution means detail quality is being missed, but there is a corresponding increase in computational cost, so 960 x 544 was selected.

Similar trade-offs were considered for YOLO mode size, with larger models having greater accuracy but larger computational load and latency. Small was chosen for its high detection frequency, seen in Table 3.1, whilst it still maintained high quality detectability. Even for the small model, CPU usage was 620%, meaning 6.2 logical CPU cores out of 8 total were used for operation, showing the importance of limiting computational power. Larger models could be for stationary objects and robot grasping, as frequent positional updates aren't required and CPU usage of even the largest model is 682%, but any real-time operation would require smooth updating.

Table 3.1: Detection Frequency of YOLO model sizes at 960 x 544 resolution

	N	S	M	L	X
Detection Frequency (Hz)	10.0	4.7	1.8	1.5	0.79

The node defaults to the raw depth topic, but aligned_depth_to_color was used instead. The raw depth may have different resolution, field of view and pixel alignments from the RGB stream. This ensures each color pixel of the mask has the corresponding depth value, ensuring mismatch avoidance.

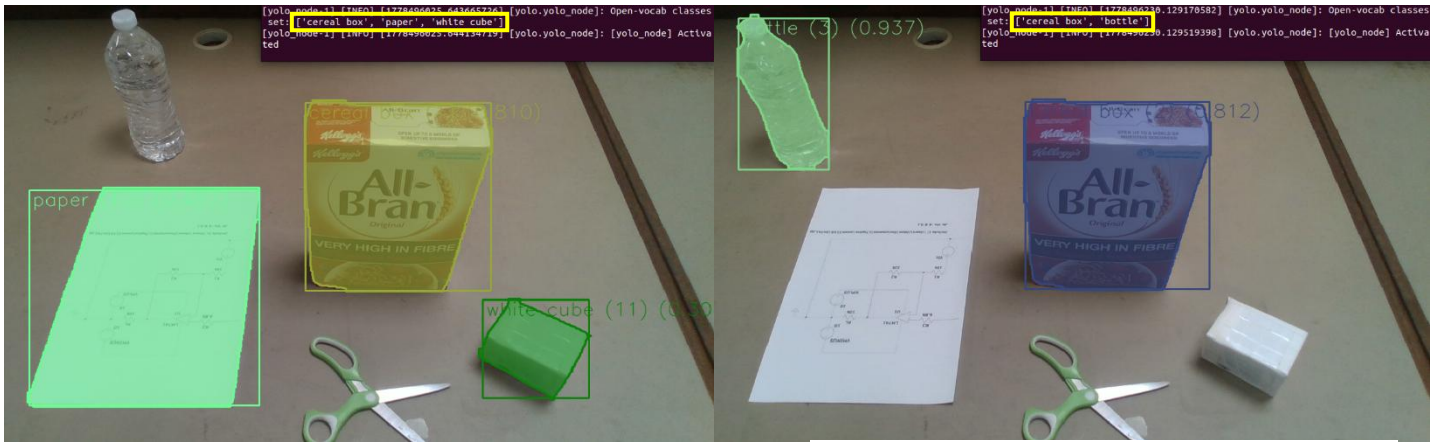
The target frame defaults to base_link, using basic ROS transforms for torso position; this transform didn't match the camera position and rotation relative to torso, leading to detected objects near the edge of the screen crashing because they're seen as out of bounds. The target frame was instead set to camera_color_optical_frame, which keeps detections in the camera's frame and transformed for the singular pre-grasp point later. The YOLO encoding parameter defaults to bgr8, but rgb8 was used instead to match RealSense color stream, avoiding potential color-channel mismatch.

3.3 Text Prompt

The yolo_ros node had functionality for base YOLO26, which is closed-set detection for the 80 COCO objects it was trained on, and YOLO26E unprompted. Both have limited utility for real-world object selection tasks. Almost all small objects fall outside of the 80 COCO classes, with some classes being too generic for users' wants; they may want to identify Coke Zero, not a bottle. Unprompted is highly inaccurate and detection needs to be for a specified target. An open-set detector with a text prompt can increase its accuracy, as the detection can be informed by what the model is looking for, whilst theoretically being able to detect any object class. It added object specification, as only passed classes get detected, thus only detecting user desired objects.

YOLOE text prompt functionality was added to yolo ros by introducing a new classes parameter. This was fowarded to the yolo_node where the list was parsed and then the prompts were applied to model using set_classes yolo function.

Figure 3.2 (a) shows working functionality. COCO objects, such as scissors and bottle aren't detected because they weren't prompted. Paper, cereal box and white cube were prompted thus detected and segmented, despite not being COCO objects. Figure 4.3 (b) supports this, as the previously detected paper and white cube aren't anymore, since there not prompted. Cereal box remains detected and bottle, a COCO object, is detected, since both were prompted. This demonstrates that the prompt model can detect novel objects and restrict detections to only prompted objects.



(a) cereal box, paper, white cube prompted & detected (a) cereal box, bottle prompted & detected

Figure 3.2: YOLO26E-S text prompt detections with prompted classes

Although confidence values for detected objects were relatively low and many objects expected to be detected weren't, such as hammer, stapler and glasses. SAM3 online was tested showing greater variety of objects it could segment but needs Nvidia GPU and performance is much slowly, even still it hits the same walls [43]. Text prompts are limited to general classes and attributes, if you want specific object detection, such as a specific brand of chips that can't be achieved through text, but it can be achieved through picture prompts or custom training, which also provide higher accuracy. Text prompts still have utility in quickly changing desired model detection classes.

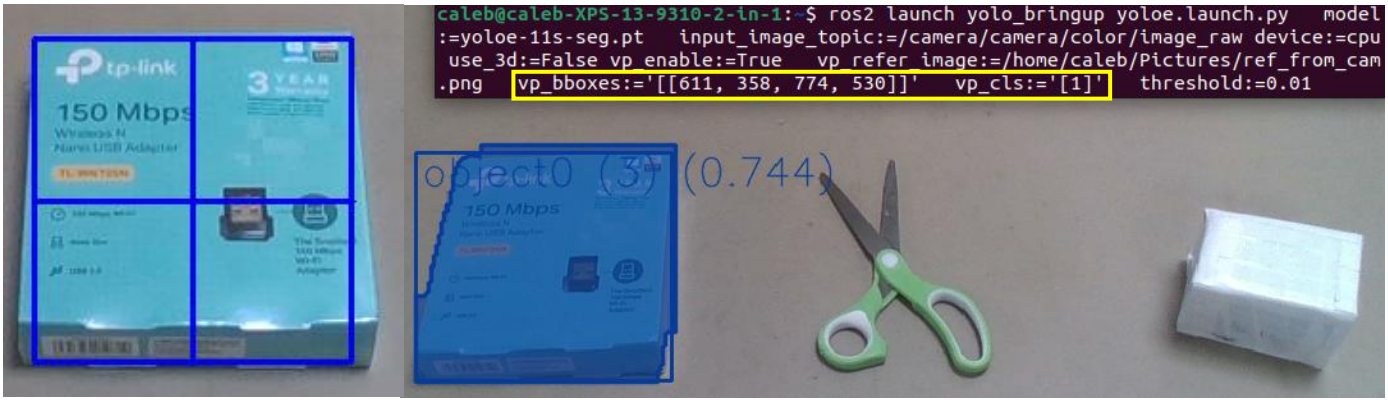
3.4 Picture Prompt

In picture prompt detection, an image containing a desired object and its 2D bounding box are passed to YOLOE to detect said object for a novel scenario.

Picture prompt functionality was added to yolo ROS by introducing new parameters for reference image, bounding-box coordinates and class ID. This was forwarded to yolo ros node where these values were passed to the yolo.predict() call alongside the YOLOEVPSegPredictor.

A script was made to save one camera frame from the RealSense camera topic. This image was passed to another script which opens the image, allowing the user to draw a rectangle around the desired object, as seen in figure 3.3 (a). This returns the rectangle's coordinates and class ID to be added to the YOLO parameters, alongside vp_enable true to ensure visual prompting is enabled.

Figure 3.3 (b) shows the resultant detection with high confidence, even with distracter objects and a different object position. There are issues with the mask spreading past the object actual dimensions and struggling for non-rectangular objects. The main issue with picture prompt is under different lighting conditions, object size, camera or object rotations, the ability to detect decreases severely. Whilst the setup procedure isn't instantaneous, it provides the potential to identify textually indescribable objects.



(a) 2D bounding box tool (b) YOLO11E-S picture prompt detection

Figure 3.3: Picture prompt workflow

3.5 Custom Trained Model

A custom YOLO model is the way to add new closed-set objects with higher accuracy and more unique cases than text prompts, furthermore greater adaptability to environmental variability than picture prompts.

To create a custom YOLO segmentation model, a dataset is required with 2D masks for each object class. Pre-existing datasets exist but usually have inferior quality and aren't available for desired object. A custom created dataset mimics the robot's camera, angle and environment for the specific desired objects, so one was created for a torch, cereal box and toothpaste; normal everyday objects of different dimensions. 266 photos were taken with half being individual shots of each desired object in different rotations and positions, thus sizes. Others being multiple desired objects in the same shot. Cylindrical and rectangular distraction objects, alongside negatives images with no objects were added to increase sturdiness.

Each photo required a 2D mask for each desired object, so LabelMe and its automatic polygon tool were used to create JSON files of segmentation coordinates, which were converted to txt files structured by YOLO's standards [44]. Google Colab was used to rent GPU space for training. Additional training graphs, batches and configuration details are provided in appendix B-D.

The first training run provided decent accuracy for validation set, but it consistently failed in reality for changed lighting, object pose, rotation, scale and background being over-specified to training data. The second training run vastly improved it, partially shown in Table 3.2. Mask precision increased by 0.042, indicating fewer false positives. Mask recall increased by 0.044, indicating fewer missed object. Mask mAP50-95 increased by 0.048, indicating improved segmentation quality; relevant, so background pixels aren't being back-projected into object's 3D mask. Mask mAP50-95 for normal YOLO26x-seg, pre-trained on 80 COCO objects, is 47.0 compared to 95.4, showing extreme segmentation accuracy; helped by less comparative objects and different environments [45]. More importantly, detection flexibility based on environment and pose in reality was massively improved.

Table 3.2: Comparison between first and second custom YOLO segmentation models

Metric	Epochs	Image Size	Mask Precision (%)	Mask Recall (%)	Mask mAP50-95 (%)
First Model	150	640	92.4	90.8	90.6
Second Model	180	960	96.6	95.2	95.4

This was achieved by running off existing model weights for more epochs, increasing the image size, but most importantly adding augmentations for flexibility. Random variations in lighting and color by changing hue, saturation, value, brightness and contrast. Variations in camera angle through degree shift, shear or perspective distortion to account for different object angles. Position and size alterations through translate, scale and flip. Background changes by adding mosaic to combine training images, mix-up to blend two images together transparently and copy-paste to cut a labelled object from one image to another. Camera noise and blur was also simulated. An example of these augmentations is seen in figure 3.4 (a) with massive variations that still maintain accuracy.

Figure 3.4 (b) shows the model working through the yolo_ros node to detect the 3 trained objects with higher confidence and quality segmentation masks, avoiding accessory objects. Although false positives still occur, the most frequent being with darker cylindrical objects incorrectly being detected as the torch, perhaps causes by its lacking detail, metal light refraction and smaller size. Novel environments and camera angles still reduce detection accuracy.

(a) Example training batch (b) Custom trained YOLO26 model detecting torch, cereal box and

Figure 3.4: Custom yolo model training and detection

3.6 3D Point Cloud

3D object position and size isn't enough information to create grasping points that generalizes outside of one object type. A 3D point cloud can provide geometric object information, useful for generalizing to different object rotations and dimensions. The node back projects 2D mask into 3D, using color aligned depth values per pixel, using the following equation to get pixels position in metres, relative to the camera's frame.

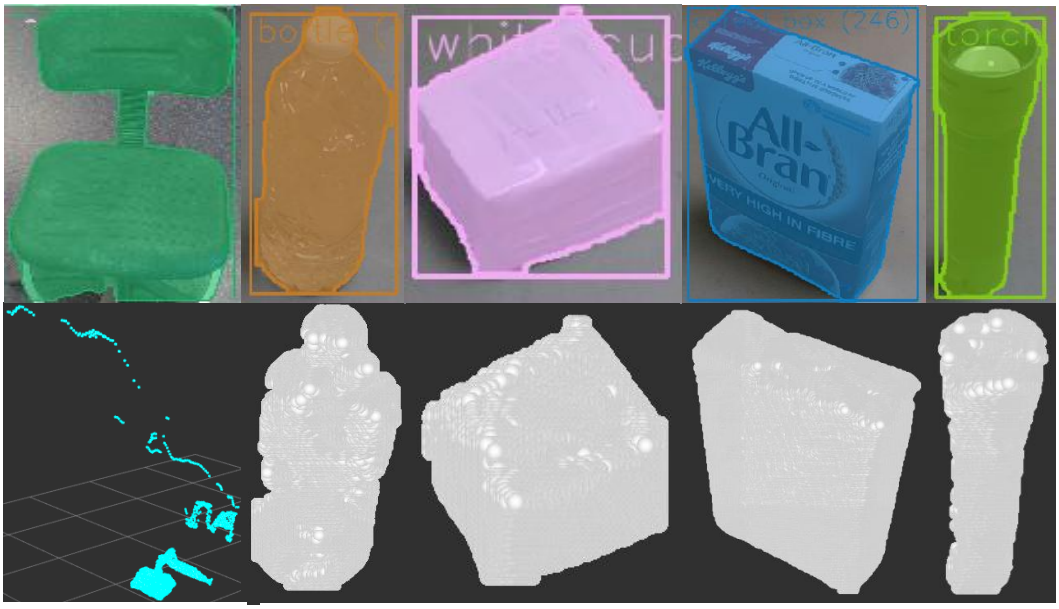
$$Z = Depth$$

$$X = (u - C_x)/f_x \cdot z$$

$$Y = (v - C_y)/f_y \cdot z$$

Where u = pixel x-position; v = pixel y-position; f_x,f_y = camera focal lengths; C_x, C_y = camera centre point.

This was done only for the highest confidence object, as object grasping occurs one at a time. The 2D mask could include background pixels, an early poor example can be seen in figure 4.6 (a). An accurate OBB and PCA require no far-away background pixels, which extends 3D point cloud spread incorrectly, thus filtering to ensure clean output was added. The mask was shrunk by a couple pixels to avoid 2D mask spill. Only the main competent, meaning the largest connected region of valid pixels is kept, as objects are whole, meaning non-touching pixels aren't part of the object. A depth band was added, keeping points close to the object's median depth, as background pixels usually have different depths. Depths outside valid range were removed. The better results can be seen in figure 3.5 (B-E) and massively increase spread accuracy for PCA and OBB. Each filter is a parameter that can be adjusted with a trade-off between object shape accuracy and noise avoidance, an in-between is required as both are essential for PCA and OBB.



(a) Chair (b) Bottle (c) White Cube (d) Cereal Box (e) Torch

Figure 3.5: YOLO 2D segmentation masks and corresponding 3D object point clouds

To save computation, samples were taken every few pixels, as the overall mask shape is relevant, not its resolution; a rectangular region of interest around the mask was examined instead of the full image. Realsense depth quality can produce holes or noise around reflective, dark, thin or edge regions; That and poor mask quality from YOLO can distort 3D cloud accuracy, requiring heavier filtering.

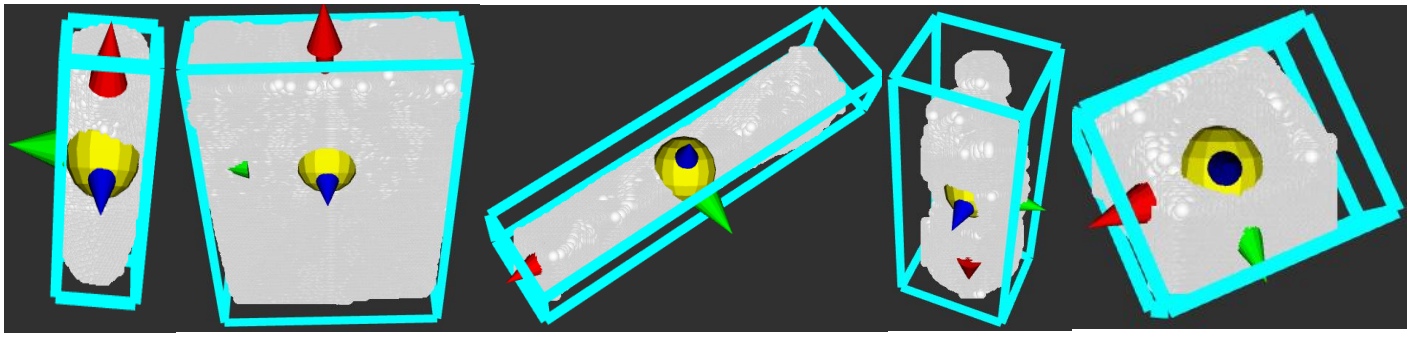
3.7 Principal Component Analysis & Oriented Bounding Box

To use 3D point cloud, general information on its rotation and dimensions needs to be analysed, this is achieved through PCA and OBB. PCA estimates rotation through ordering directions of spread and OBB gets 3D bounding box in the PCA axes.

Achieved by calculating the centroid, which is the average position of all 3D points. The point cloud is centred by subtracting the centroid from every point. The covariance matrix is calculated, measuring how points are spread in 3D. Eigen calculations provide eigen vectors which become PCA axes and eigenvalues which tell how much spread exists across that axis, thus used to order axis spread by principal, secondary and tertiary from most to least spread, thus length. The point cloud is then measured relative to these PCA axes, finding minimum and maximum values for each axis, becoming corner values for the 3D bounding box, which is oriented relative to PCA not camera or torso. OBB is then related back to camera frame and visualisation markers added.

Figure 4.7 shows strong PCA and OBB adherence to different object types and poses. Relevant as PCA direction and OBB tightness to the object are crucial for accurate grasping. Protrusions in the object's rectangular symmetry could dominate spread, becoming the main direction, thus not aligning with ideal grasp directions. A noisy mask can lead to completely incorrect OBB, since it takes max and min points which are usually noise. The axis direction has sign ambiguity, meaning it can flip between frames. Symmetric objects, such as cubes, are problematic in that they don't have a clear unique orientation.

The camera has one angle, meaning the mask only represents front facing depth. This has the effect of making the tertiary axis, usually the object's side that faces the camera, as depth range is usually less than the objects; this means even for cylindrical objects where usually secondary and tertiary axis would usually flip, secondary axis stays constant to the side of the object which has the side effect of being better for consistent grasping.



(a) Toothpaste (b) Cereal Box (c) Torch Lying Sideways (d) Bottle (e) White cube

Figure 3.6: PCA axes and OBB for object's 3D point cloud. Red arrow is principal component axis, green is secondary axis, blue is tertiary axis. Light blue box is oriented bounding box.

3.8 Right Pre-grasp Point

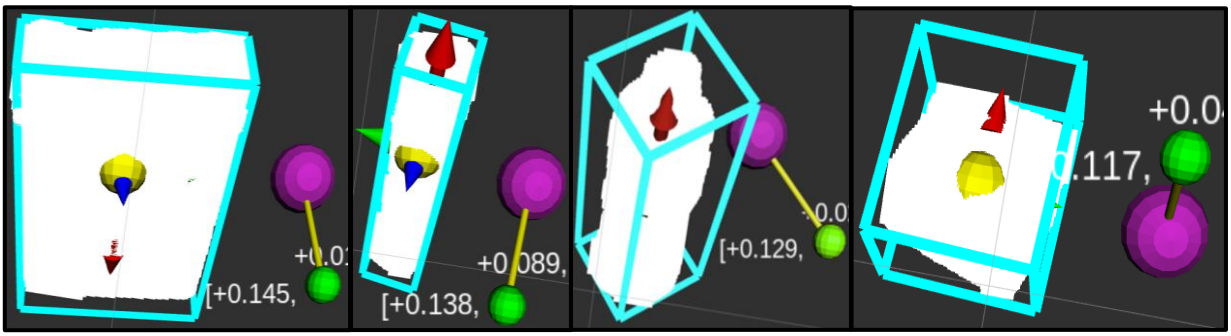
Object geometry is important, but IK requires a single target position for the manipulator to reach; this is the pre-grasp point for the right hand. An approach axis is chosen. For upright objects, its secondary axis, because tertiary is usually facing the camera, making it harder to grasp and primary is the longest, meaning you want to grasp perpendicular to increase grasp surface area. Counterexamples exist, so the approach axis can be altered by changing parameter values.

The node reads the OBB corners of that chosen axis to get its half extent, the distance from centroid to object edge. This half extent, plus clearance is used to create two possible pre-grasp point on either side of the object, at the midpoint of its principal axis height; because of PCA direction flipping, the point with the larger x, therefore right-most point is selected since grasping occurs with the right hand.

This point was converted from camera frame to torso-link, the coordinate frame expected by IK node. Pinocchio extracts the transform from Unitree G1's URDF model, which was imported for the simulation [36] and edited to add a camera with its position and orientation ideally matching reality. The rigid transform applied both rotation and translation that converted camera into torso_link values, meaning forward, left and up from robot torso.

IK expects wrist position, but this point is for hand centre position, therefore a wrist offset is required. The hand and wrist are parallel to each with the hand being perpendicular to secondary axis, so the tertiary axis is the offset vector's direction between hand and wrist; this is normally true, as teritary is the closest to facing the robot; instead, trying to offset wrist position by relative angle to robot won't work because it doesn't match arm angle. The tertiary axis can flip, but we now have values in torso-link, allowing the program to reverse the tertiary axis if it's pointed away from the robot.

Figure 3.7 shows these hand pre-grasp points in purple, all to the right side of the object, regardless of secondary axis polarity; there a clearance offset of 6cm was used, but its modular parameter with 4cm offset seeing more success. The yellow line shows that the wrist offset direction matches the tertiary direction shown by the blue arrow with a wrist offset of 10cms being applied. The green sphere represents the wrist position that will be passed for IK. Figure 3.7 (d) shows the problem with objects whose principal axis isn't up.



(a) Cereal (b) Torch (c) Bottle (d) White

Figure 3.7: Right pre-grasp points relative to PCA and OBB. Purple sphere represents hand centre pre-grasp point, green sphere represents wrist target, yellow line shows their offset.

3.9 Movement Script

The robot calculated a grasp point, but it needed to calculate the required joint positions to arrive at that point. Whilst moving safely to a point, grasping an object and then safely returning.

3.9.1 Inverse kinematics

To solve for the joint's positions, inverse kinematics using Pinocchio was calculated for the right_wrist_yaw_link as the end effector. The G1 URDF model was loaded and reduced, making only right-arm joints free and locking all others. Torso frame target was converted to Pinocchio's internal model frame. The wrist position error from targeted to actual is calculated. Jacobian maps how minor changes in right arm joints would move the wrist, and its damped pseudoinverse calculates the required joint change for desired wrist movement to eliminate error between current and desired position. This gives small joint changes, ensuring joints don't exceed limits, until wrist error is below tolerance or its iteration limit reached. A ready pose near the object was added to decrease the error between desired and actual wrist position, making iterations a lot more likely to ideally converge.

Position only IK was chosen for its lower complexity, which reduced IK failures and unsafe arm movements, but it doesn't guarantee correct wrist orientation and can't adapt to novel rotation requirements. For upright objects, the ready pose orientation and added posture bias were enough for pickups to consistently work. The posture bias allowed any leftover kinematic freedom to make the shoulder and elbow more 90 degrees°.

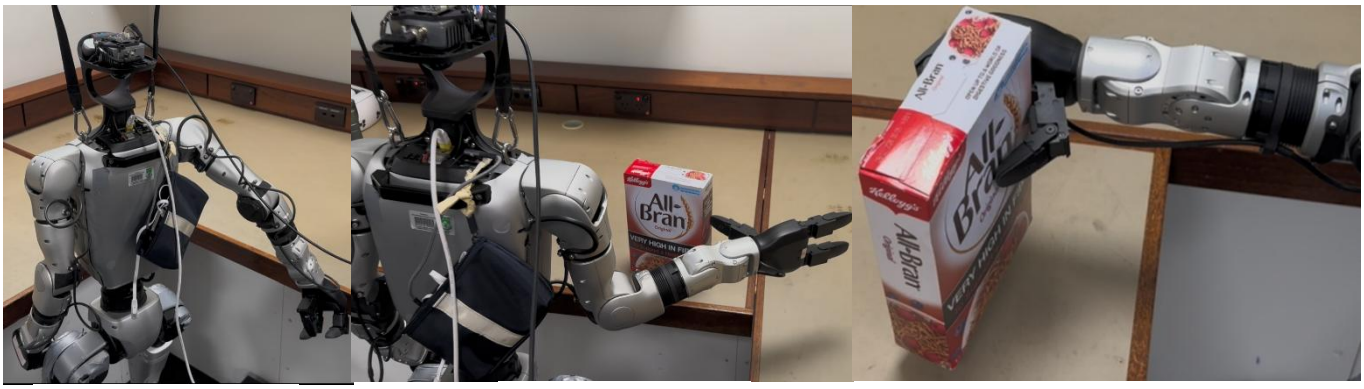
3.9.2 Pressure based grasping

At ready pose, Dex3-1 fingers are opened. After IK position is reached, the fingers close until a maximum pressure change from baseline from any pressure sensor supersedes its threshold or a predefined safety pose for finger's positions is reached to ensure the object can't be crushed. Fine-tuning pressure threshold is important, as too little finishes closing before fully enveloping object and too much could crush the object. A pressure average from baseline was added, as a secondary option, but certain pressure sensors aren't pressed during grasping, limiting its effectiveness.

3.9.3 Safe Sequence

Sending robot directly to final joint poses in IK could lead to sudden movements and collisions, so a safe workflow was designed to grasp the object and come back. Any IK values out of workspace boundary were rejected, as points behind the robot, under the desk, too far to side could enable dangerous collisions. Before running, IK is calculated and a dry run is presented, showing all joint angle changes through each stage, requiring a confirmation button. Arm_sdk through unitree_SDK2_python gradually increases from 0 to 1 and pins the left arm to its current joint positions. The right arm right shoulder is rolled to avoid colliding with G1's leg. For desk clearance, the shoulder is pulled back before elbow is bent, avoiding desk collisions, seen in figure 3.8 (a). Then, G1 moves to ready pose with a 90-degree elbow, above the table and to the very right to avoid collisions, providing straight line to right pre-grasp point. Dex3-1 fingers are opened at ready. Moves to calculated IK target joint values and closes dex3-1 using pressure-based grasping or predefined grasp pose. Lifts object up through its elbow before returning it back down and releasing. For desk

exiting, elbow moves up first and then shoulder moves back. Hand is closed, before the arm_sdk weight are released. If script is interrupted is set arm weight to 0.



(a) Desk avoidance (b) Ready pose (c) Grasped object lifted

Figure 3.8: Key Movement Positions during object grasping

3.9.4 Grasping Results

Videos showing successful grasping are available on the researcher's GitHub homepage, linked in appendix A and reference list [46]. They show 3 separate grasps for 3 different objects cereal box, water bottle and toothpaste in 3 different positions, this exemplifies the veracity and versatility of the pipeline.

A quantitative analysis of performance is shown in table 3.3 with raw run information being available in appendix D. The objects were placed in different positions over IK range 0.1973 to 0.3387 forward, left -0.2868 to -0.01347 and 0.0365 to 0.1471; this testing range was limited by the overlap between camera detection and workspace areas reachable by IK. The grasping was 83.3% successful. One failure occurred by pressure threshold stopping its grasp too early; if the predefined grasp pose was used, success rate would be 86.7%. Two failures occurred for the water bottle when positioned to the very left with just the fingertips touching. Two failures occurred for toothpaste being knocked over; one being hand going too far, the other its thumb clashing during rotation after ready pose. Mean offset was 2.9 cm, closely matching the 3 cm programmed palm-edge offset (4 cm palm-centre offset minus 1 cm palm radius). This makes max absolute offset error 4.5cm not far enough. 3cm deviations could be handled by finger length with fingertips being used to pick up some deviated objects. Cereal box grasps were more effective than toothpaste, because of its larger size and weight keeping its place for displacement; the rectangular cereal box also had more definite principal axes than the cylinders, aligning with the box's faces.

Table 3.3: Quantitative grasping performance across 3 objects. Offset was distance between objects edge and palm edge during grasping.

	Attempts	Successful grasps	Success rate	Mean offset (cm)	Max offset (cm)	Min offset (cm)
Toothpaste	**10**	**7**	**70%**	**3**	**7.5**	**-0.5**
Water Bottle	**10**	**8**	**80%**	**3.5**	**7**	**0.5**
Cereal Box	**10**	**10**	**100%**	**2.4**	**5**	**-0.5**
Total	**30**	**25**	**83.33%**	**2.9**	**7.5**	**-0.5**

3.9.5 Grasping Discussion

The largest offset inaccuracies came from misleading OBB and PCA originating from a poor mask, in addition to some far-left objects, indicating potential IK joint deviations over longer distances; position errors accounted for 3 unsuccessful grasps. The overall success satisfies the hypothesis, as it demonstrates the accuracy of the entire detection and localisation pathway, showing it provides a basis for grasping.

This success rate is below popular alternatives, such as GPD with 93%, Dex-net 4.0 at 95%, contract-grasp net 95% but these are fixed industrial arms with parallel/suction gripper close to the object generally with an external camera and greater funding [26, 47, 48]. Humanoid robots represent better comparisons. The best comparison is with HERO, which used a Unitree G1 to grasp objects using open vocabulary detection (SAM-3), grasp pose detector and IK. It had an 83.8% average success rate for picking up novel object and 2.44cm manipulator tracking error, very close to observed grasping performance. Although HERO's task was harder, requiring whole-body manipulation with simpler performance of 10 daily objects on table, reporting a 90% success rate [49].

A major limitation is that many IK positions can't be seen by the camera, thus objects are't detected or clip out of frame; this is caused by camera not being angled downwards enough. A camera and Ethernet connection are required, limiting robot's movements relative to the laptops. The dex3-1 joints can't release, like arm weights until the robot's power is turned off; this requires finger position to be set with complete closing, introducing risks, so a half close state was introduced. The biggest limiting factor for success was PCA and OBB accuracy; higher success rate implementations relied on learned grasp pose detectors, like anygrasp, but they require a Nvidia GPU unavailable to the project [50].

4. Conclusions And Future work

4.1 Conclusions

The project achieved its primary objectives. The capacity to detect all objects of known classification with high accuracy and low speed through effective YOLO ROS parameter tuning. A YOLO26 custom training workflow was created, allowing accurate and variable detection of any added closed set objects; a model was trained for a cereal box, a toothpaste container and a torch, showing high detection and mask accuracy over greater environmental variability through harsher training augmentations. Although the process is lengthy and false positives can occur.

All objects of an unknown classification could be detected with moderate accuracy and low speed requirements, after YOLO26E prompt functionality was added. YOLO26E text prompt could detect non-COCO objects and only detected prompted classes, although many intuitive objects weren't detected. YOLO26E picture prompt could accurately detect novel objects through an example photo with drawn bounding box coordinates, although segmentation quality can be poor and detection doesn't adapt to different poses and lighting.

An accurate 3D mask was created that used many filters to reduce noise. A general object orientation was achieved through PCA and dimension through OBB. This struggled with incomplete masks, protrusions and noise.

A grasp position was created, effective for upright objects as the principal component is up, tertiary towards robots' camera, secondary to objects right side. The hand offset is to object's right side and wrist offset towards robot camera, using secondary and tertiary directions by default. Whilst these directions could be altered more flat or square objects would struggle.

Using Pinocchio IK and SDK2 Python workflow, successful and safe object grasps were demonstrated for different object types and positions with an 83.3% accuracy and 4.5cm maximum deviation observed for 30 tests. The intention behind grasping was to be a proof of concept that validates the vision workflow, rather than a fully general grasping system capable of handling any object. Clearance poses, pressure and pre-defined grasping limits allowed all operations to occur in a safe manner. All of this was achieved, whilst remaining computationally cheap and runnable on a laptop's CPU.

4.2 Future Work

Much potential future work grows off this program's findings. The modularity of the ROS nodes allows any nodes to be replaced without destroying other functionality. Sam 3 should be examined for open set detection, comparing its accuracy to yolo26e text prompt. Many more objects could be trained with higher quality datasets, containing more pictures, distraction objects and environments. For very small object detection SALI tiling [51] could be added, as it creates smaller overlapping crops of frames to run yolo on them individually. Add the ability to tilt the camera downwards to see objects closer to the robot that can be reached by IK.

A two-camera setup could be used to get a full 3D point cloud. Iterative Closest Point (ICP) could increase 3D point mask quality by minimizing point-to-point distances, aligning the source to the target point cloud. This target may be a CAD mesh modelled for the object or against the previous masked point cloud to reduce jitter and maintain smooth pose after slight changes in view. With a Nvidia GPU, Isaac ROS pose-estimation model could be used bringing greater accuracy. Rather than OBB taking minimum and maximum it could take the percentiles to ignore the most extreme points, eliminating noise.

With faster computation, a learned grasp pose detection [26] should be tested to see how well it adapts to G1's gripper potential bypassing need for orientation and dimension estimation; If it struggles, a new GPD could be trained on for unitree's hand specifications and object types, especially for 5 fingered hands.

For grasping to respond to all object types, IK that considers wrist position and orientation should be implemented. IK should move to offset point in the approach axis's orientation. Moveit2 should be used instead of Pinocchio for collision checking, motion planning, scene and trajectory management.

A simple implementation would be for 3 different object types. If an object is upright, it follows traditional rightmost grasping. If an object is flat or square, the approach axis should be the tertiary component, as the upward face is pointing towards the camera; the wrist offset should be in the secondary axis because it's perpendicular to the principal axis, thus greatest grasping surface area. The palm should point downwards, rotating to match secondary axis direction.

Some objects are too big or thin across certain axes that one hand grasping isn't feasible, instead, both hands could be used. Principal axes order would be ignored, simpley create offsets in all 9 object directions and pick the rightmost point for the right hand and leftmost point for the left. Pick the axis rotation that is closest to the object to torso offset rotation and use that for both wrist offsets. IK could be run concurrently for both arms with joint timers, ensuring they reach and grasp at the same time.

For true shopping functionality, the robot needs to detect and go to objects with its movements. A LIDAR slam map could provide scene navigation with robot exploring, until its desired yolo object is detected and then navigates to stand in front of object, before stationary grasping. A shops catalogue of items could be detected by a YOLO26 model custom trained using the project's workflow or a catalogue of product pictures with their items bounding box using YOLO26E picture prompt. These possibilities show the utility of this detection, localisation and manipulation pipeline for the tasks of the future.

References

[1] B. Schmidt, "Teslabot: Tesla's stunning decision to make its own robot, to get the groceries," The Driven, Aug. 20, 2021. [Online]. Available: https://thedriven.io/2021/08/20/teslabot-tesla-stunning-decision-to-make-its-own-robot-to-get-the-groceries/

[2] C. Stauffer and W. E. L. Grimson, "Adaptive background mixture models for real-time tracking," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Fort Collins, CO, USA, 1999, pp. 246–252, doi: 10.1109/CVPR.1999.784637.

[3] G. R. Bradski, "Computer vision face tracking for use in a perceptual user interface," Intel Technol. J., Q2 1998.

[4] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), San Diego, CA, USA, 2005, pp. 886–893, doi: 10.1109/CVPR.2005.177.

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, real-time object detection," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Las Vegas, NV, USA, 2016, pp. 779–788, doi: 10.1109/CVPR.2016.91.

[6] Ultralytics, "COCO dataset," Ultralytics Docs. [Online]. Available: https://docs.ultralytics.com/datasets/detect/coco

[7] R. Sapkota and M. Karkee, "YOLOE-26: Integrating YOLO26 with YOLOE for Real-Time Open-Vocabulary Instance Segmentation," arXiv.org, 2026. https://arxiv.org/abs/2602.00168 (accessed May 19, 2026).

[8] Tesla, "Tesla AI Day 2022," YouTube, Oct. 1, 2022. [Online]. Available: https://www.youtube.com/watch?v=ODS97aZfGO0

[9] Boston Dynamics, "Making Atlas see the world," Boston Dynamics Blog, Apr. 2024. [Online]. Available: https://bostondynamics.com/blog/making-atlas-see-the-world/

[10] A. Brohan et al., "RT-2: Vision-language-action models transfer web knowledge to robotic control," arXiv:2307.15818, 2023. [Online]. Available: https://arxiv.org/abs/2307.15818

[11] M. Wiggers, "Amazon unveils Sparrow robot arm that can pick items using suction," The Robot Report, Nov. 10, 2022. [Online]. Available: https://www.therobotreport.com/amazon-unveils-sparrow-robot-arm-that-can-pick-items-using-suction/

[12] E. Ackerman, "Digit is now shipping, and we spent an afternoon with it," IEEE Spectrum, Apr. 26, 2023. [Online]. Available: https://spectrum.ieee.org/agility-robotics-digit-ships

[13] M. Bajracharya et al., "Demonstrating mobile manipulation in the wild: A metrics-driven approach," arXiv:2401.01474, 2024. [Online]. Available: https://arxiv.org/abs/2401.01474

[14] Unitree Robotics, "About G1," Unitree G1 Developer Documentation. [Online]. Available: https://support.unitree.com/home/en/G1_developer/about_G1 (accessed May 19, 2026).

[15] "Unitree G1 overview," Quadruped Documentation. [Online]. Available: https://www.docs.quadruped.de/projects/g1/html/g1_overview.html (accessed May 19, 2026).

[16] Dell, "XPS 13 7390 2-in-1 setup and specifications guide," Dell Technologies, 2019. [Online]. Available: https://dl.dell.com/topicspdf/xps-13-7390-2-in-1-laptop_setup-guide_en-us.pdf

[17] Roboflow, "RF-DETR benchmarks," RF-DETR Documentation. [Online]. Available: https://roboflow.github.io/rf-detr/learn/benchmarks/

[18] Ultralytics, "Ultralytics YOLO26," Ultralytics Docs. [Online]. Available: https://docs.ultralytics.com/models/yolo26/

[19] T.-Y. Lin et al., "Microsoft COCO: Common objects in context," in Proc. Eur. Conf. Comput. Vis. (ECCV), Zurich, Switzerland, 2014, pp. 740–755.

[20] Ultralytics, "SAM 3," Ultralytics Docs. [Online]. Available: https://docs.ultralytics.com/models/sam-3

[21] Hugging Face, "Zero-shot object detection," Hugging Face Transformers Documentation. [Online]. Available: https://huggingface.co/docs/transformers/en/tasks/zero_shot_object_detection

[22] ApX Machine Learning, "Limitations of template matching," Introduction to Computer Vision. [Online]. Available: https://apxml.com/courses/introduction-to-computer-vision/chapter-5-introduction-object-recognition/limitations-template-matching

[23] NVIDIA, "Isaac ROS pose estimation," GitHub repository. [Online]. Available: https://github.com/NVIDIA-ISAAC-ROS/isaac_ros_pose_estimation

[24] Y. Ma, Z. Song, Y. Zhuang, J. Hao, and I. King, “A Survey on Vision-Language-Action Models for Embodied AI,” arXiv.org, 2024. https://arxiv.org/abs/2405.14093

[25] J. Inayat-Hussain, “Vision Language Action Models for Humanoid Robotics: Evaluating Capabilities, Limitations, and Future Directions,” The University of Western Australia, Oct. 2025.

[26] A. ten Pas, "GPD: Grasp Pose Detection," GitHub repository. [Online]. Available: https://github.com/atenpas/gpd

[27] J. Smith, “Developing Humanoid Robotics: Human Pose Mapping, SLAM and Path Planning for Real World Applications,” University of Western Australia, Oct. 2025.

[28] T. Ryan, “Depth-Camera-Based Real-Time Human-to-Humanoid Pose Retargeting,” University of Western Australia, Oct. 2025.

[29] Open Robotics, "ROS — Robot Operating System." [Online]. Available: https://www.ros.org/

[30] Open Robotics, "ROS 2 Humble Hawksbill release," ROS 2 Documentation. [Online]. Available: https://docs.ros.org/en/rolling/Releases/Release-Humble-Hawksbill.html

[31] Intel RealSense, "realsense-ros," GitHub repository. [Online]. Available: https://github.com/realsenseai/realsense-ros

[32] M. González-Santamarta, "yolo_ros: ROS 2 wrapper for Ultralytics YOLO," GitHub repository. [Online]. Available: https://github.com/mgonzs13/yolo_ros

[33] Unitree Robotics, "unitree_sdk2_python," GitHub repository. [Online]. Available: https://github.com/unitreerobotics/unitree_sdk2_python

[34] Unitree Robotics, "unitree_mujoco," GitHub repository. [Online]. Available: https://github.com/unitreerobotics/unitree_mujoco

[35] NVIDIA, "Isaac Lab," GitHub repository. [Online]. Available: https://github.com/isaac-sim/IsaacLab

[36] AIST, "g1_description: URDF description for the Unitree G1," GitHub repository. [Online]. Available: https://github.com/isri-aist/g1_description

[37] ros-controls, "gz_ros2_control," GitHub repository. [Online]. Available: https://github.com/ros-controls/gz_ros2_control

[38] Y. Chen, "Inverse manipulator kinematics," Robotics ENGN4627 lecture notes, Australian National Univ., Canberra, Australia, 2005. [Online]. Available: https://users.cecs.anu.edu.au/~chen/teaching/Robotics_ENGN4627_2005/lectureNotes/engn4627-Part04.pdf

[39] MoveIt, "moveit2," GitHub repository. [Online]. Available: https://github.com/moveit/moveit2

[40] stack-of-tasks, "Pinocchio," GitHub repository. [Online]. Available: https://github.com/stack-of-tasks/pinocchio

[41] International Organization for Standardization, ISO 10218-1:2025 — Robotics — Safety requirements — Part 1: Industrial robots, ISO, Geneva, Switzerland, 2025.

[42] Intel, "Intel RealSense D400 series product family datasheet," Datasheet 337029-005, Intel Corp. [Online]. Available: https://cdrdv2-public.intel.com/841984/Intel-RealSense-D400-Series-Datasheet.pdf

[43] prithivMLmods, "SAM3 demo," Hugging Face Spaces. [Online]. Available: https://huggingface.co/spaces/prithivMLmods/SAM3-Demo

[44] K. Wada, "labelme: Image polygonal annotation with Python," GitHub repository. [Online]. Available: https://github.com/wkentaro/labelme

[45] Ultralytics, "Segmentation," Ultralytics Docs. [Online]. Available: https://docs.ultralytics.com/tasks/segment

[46] C. Hodgson-Taylor, "ros_ws," GitHub repository. [Online]. Available: https://github.com/CalebAndChill/ros_ws

[47] J. Mahler et al., "Learning ambidextrous robot grasping policies," Sci. Robot., vol. 4, no. 26, art. no. eaau4984, 2019, doi: 10.1126/scirobotics.aau4984.

[48] M. Sundermeyer, A. Mousavian, R. Triebel, and D. Fox, "Contact-GraspNet: Efficient 6-DoF grasp generation in cluttered scenes," in Proc. IEEE Int. Conf. Robot. Autom. (ICRA), 2021. [Online]. Available: https://arxiv.org/abs/2103.14127

[49] R. Dong, Z. Li, X. He, and S. Gupta, "Learning humanoid end-effector control for open-vocabulary visual loco-manipulation," arXiv:2602.16705, 2026. [Online]. Available: https://arxiv.org/abs/2602.16705

[50] graspnet, "AnyGrasp SDK," GitHub repository. [Online]. Available: https://github.com/graspnet/anygrasp_sdk

[51] Ultralytics, "SAHI tiled inference," Ultralytics Docs. [Online]. Available: https://docs.ultralytics.com/guides/sahi-tiled-inference

Appendix

Appendix A. Researcher's project GitHub with videos of successful grasping on the homepage

https://github.com/CalebAndChill/ros_ws

Appendix B. Custom training plots for first iteration

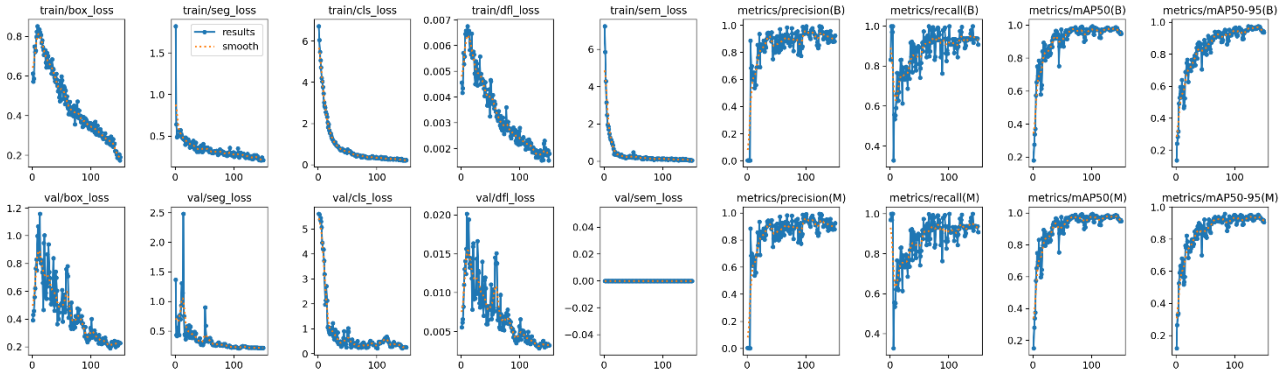


Figure B1: Training curves for first iteration

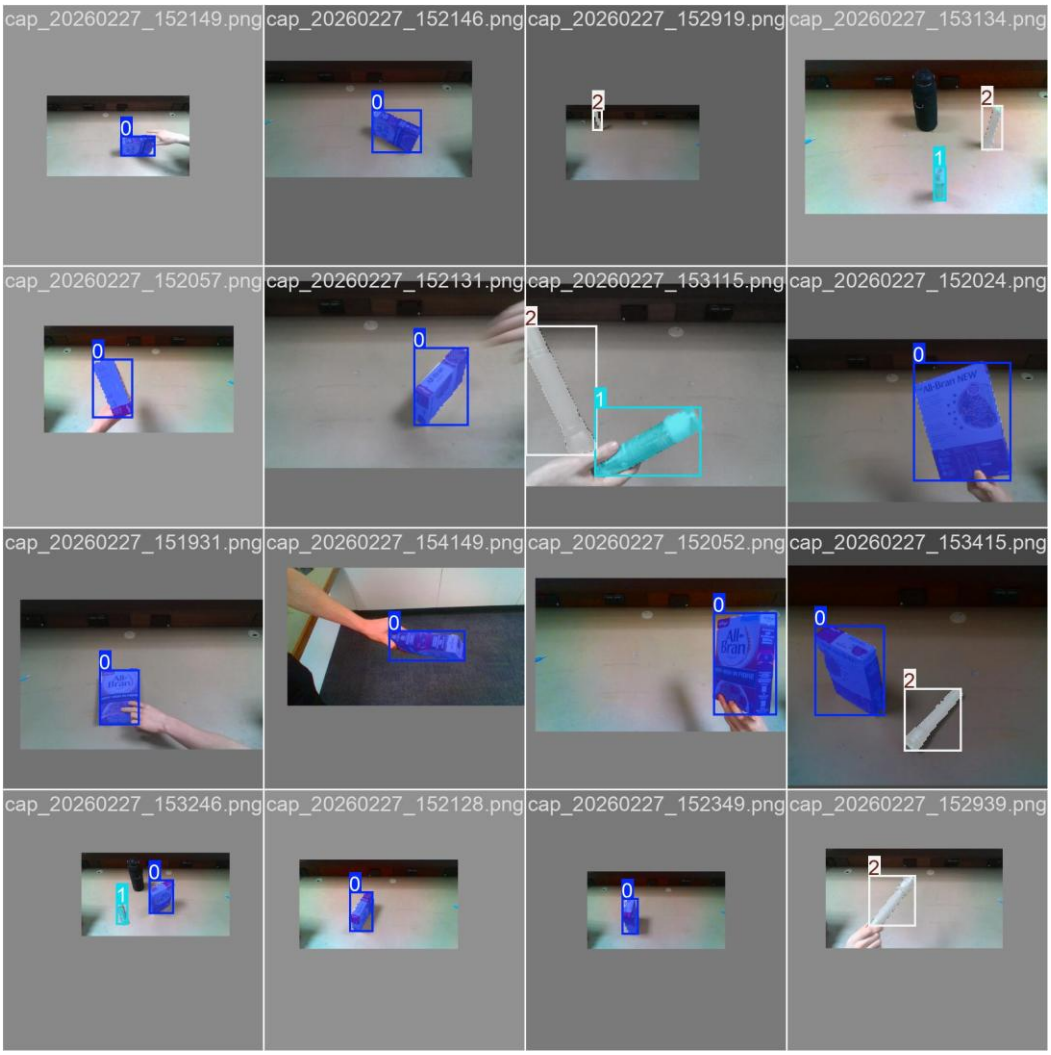


Figure B2: Example batch for first iteration

Appendix C. Custom training plots for second iteration

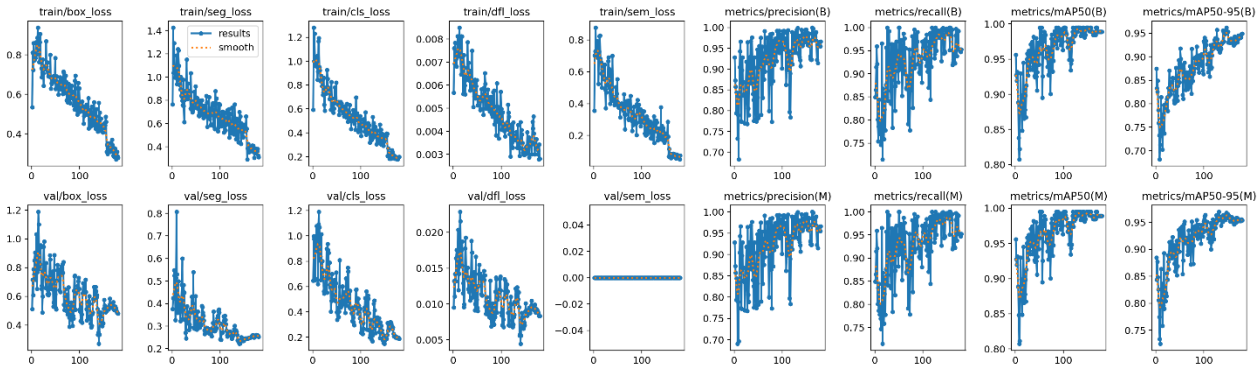


Figure C1: Training curves for second iteration

Figure C2: Validation batch0 for second iteration

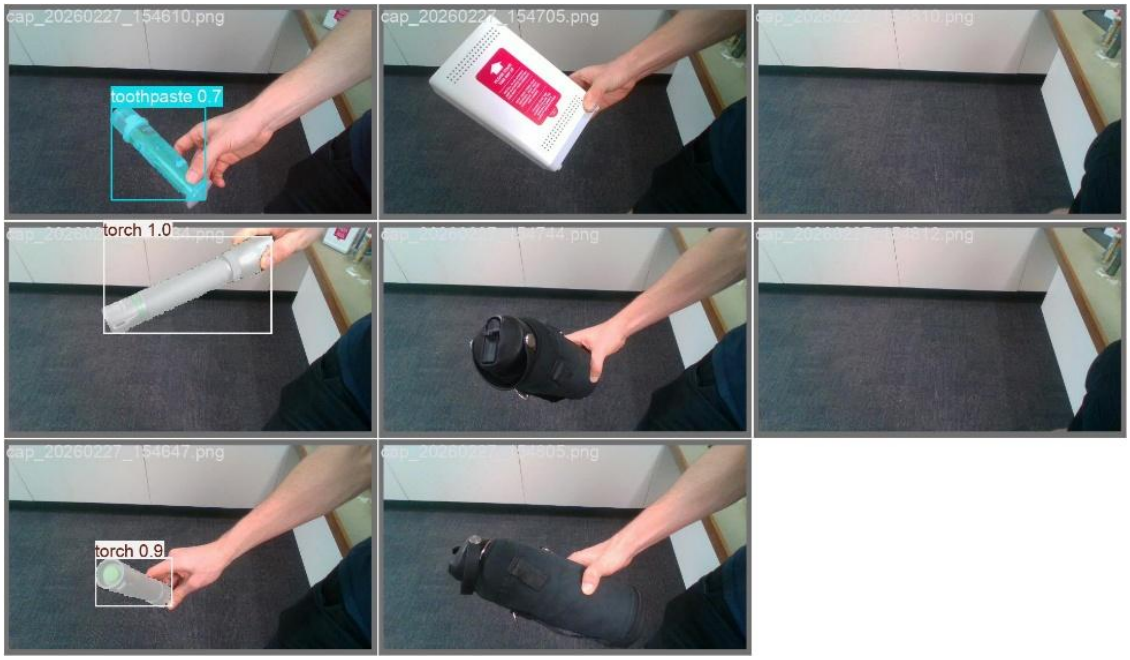


Figure C3: Validation batch1 for second iteration

Figure C4: Example batch for second iteration

Appendix D. Training script for second iteration of custom trained model

```
!pip install -U ultralytics
from ultralytics import YOLO
import albumentations as A

model = YOLO("/content/drive/MyDrive/best.pt")

custom_transforms = [
    A.RandomBrightnessContrast(p=0.4),
    A.GaussNoise(p=0.2),
    A.MotionBlur(p=0.15),
    A.CLAHE(p=0.15),
]

model.train(
    data="/content/Dataset_YOLO/data.yaml,
    epochs=180,
    imgsz=960,
    hsv_h=0.03,
    hsv_s=0.80,
    hsv_v=0.60,
    degrees=8,
    translate=0.15,
    scale=0.65,
    shear=2,
    perspective=0.0005,
    fliplr=0.5,
    bgr=0.15,
    mosaic=0.7,
    mixup=0.10,
    copy_paste=0.25,   # remove if detection-only
    close_mosaic=25,
    augmentations=custom_transforms,
    plots=True,
)
```

Appendix E. 30 Grasping trials with detected IK position, object offset from palm, whether they succeeded or failed with failure reasons.

Object Grasp Attempt	IK forward	IK left	IK up	Approximate Object Offset	Success/Failure
Toothpaste	0.3105	-0.1698	0.1407	- 0.5cm (palm passed object edge)	Failure (Knocked over going too far)
Toothpaste (2)	0.3387	-0.1314	0.1407	6cm	Success (although grasped by fingertips)
Toothpaste (3)	0.2763	-0.2028	0.1471	0.5cm	Success
Toothpaste (4)	0.3263	-0.1848	0.1471	0.5cm	Success
Toothpaste (5)	0.3044	-0.1384	0.1468	0cm	Success
Toothpaste (6)	0.2157	-0.1268	0.0442	5cm	Success (although grasped by fingertips)
Toothpaste (7)	0.2423	-0.1357	0.0374	4cm	Success
Toothpaste (8)	0.2729	-0.0916	0.0442	3cm	Success
Toothpaste (9)	0.2205	-0.0717	0.0468	7.5 cm	Failure (skimmed fingertips, but enough contact for lift)
Toothpaste (10)	0.2049	-0.2793	0.0451	~	Failure (thumb knocked over, during rotation after ready pose)
Water bottle (1)	0.2636	-0.0935	0.0555	5cm	Success (although grasped by fingertips)
Water bottle (2)	0.1973	-0.1238	0.0460	1cm	Success
Water bottle (3)	0.2345	-0.1944	0.0653	2cm	Failure (correct grasp position, but pressure threshold stopped grasp early)
Water bottle (4)	0.2561	-0.2004	0.0387	6cm	Success (although grasped by fingertips)
Water bottle (5)	0.3061	-0.1813	0.0412	2cm	Success
Water bottle (6)	0.2561	-0.01347	0.0379	0.5cm	Success
Water bottle (7)	0.2700	-0.0732	0.0391	7cm	Failure (skimmed fingertips, but enough contact for lift)
Water bottle (8)	0.2678	-0.0611	0.0365	6cm	Success (although grasped by fingertips)
Water bottle (9)	0.2679	-0.2311	0.0365	3 cm	Success
Water bottle (10)	0.2049	-0.0939	0.0459	2 cm	Sucess
Cereal box (1)	0.2762	-0.1891	0.1002	3cm	Success
Cereal box (2)	0.2678	-0.2781	0.1234	0cm	Success
Cereal box (3)	0.3017	-0.2868	0.1134	-0.5 cm	Success (pushed cereal box before grasping)
Cereal box (4)	0.2344	-0.2261	0.1106	3 cm	Success
Cereal box (5)	0.3047	-0.2513	0.0985	2 cm	Success
Cereal box (6)	0.3162	-0.1752	0.1078	2cm	Success
Cereal box (7)	0.3211	-0.2011	0.0985	1cm	Success
Cereal box (8)	0.2547	-0.1056	0.1023	5cm	Success (although grasped by fingertips)
Cereal box (9)	0.2744	-0.1447	0.1113	3 cm	Success
Cereal box (10)	0.2324	-0.1229	0.0994	4 cm	Success