Autonomous Robotic Dishwasher Arm

MIE1075 AI Applications in Robotics

By:

Noman Ali 1003602657

Mireille Baher Gendy 1000640041

Alexis Bruneau 1008704270

Ryo Teraoka 1008187515

Presented to Dr.Andrew Goldenberg

December 31st, 2022

Contents

Table of Figures

Table of Tables

Abstract

With the advances in technology, grasping objects using a robotic arm with the help of AI has been on the rise. In our project, we propose a method to have an autonomous robotic arm for dishes. Our process follows these main steps: object localization, object pose estimation, grasp estimation, and motion planning of the robotic arm. Each of those main steps is enhanced with AI tools and methods such as YOLOv7 for object detection, MiDaS for depth estimation, Canny method for edge detection, Gen6D for pose estimation, and a neural network and numerical optimization technique for the motion planning of the robotic arm. The main advantage of our proposed method is that it only requires a monocular RGB camera. Multiple solutions exist for grasping complex object but often requires different hardware such as lidar, stereovision camera, and RGB-D camera. Our approach aims at reducing the cost of the overall robot and reducing the dependencies among hardware components.

1 Introduction

Incorporating routine housework chores into an individual's daily schedule can be a tedious task. Professionals everywhere are required to complete several tasks after a workday. As of today, most of these tasks are too complex to be fully automated such as cooking and washing dishes. According to a survey in the United States, more than 25 % of respondents said that washing dishes is their least favorite household chore (DishFish, 2019). Thus, by automating the dishwashing process, we will no longer have to do this tedious task after a meal. However, there is no commercialized robotic arm or related system that picks up dirty dishes and places them into a dishwasher. With new emerging technology, grasping objects using AI has been a hot topic recently (Andras, 2020). Most robotic arm-based systems use lidar, depth cameras, and stereo cameras to achieve object picking and placement (Akhilesh Kumar Mishra, 2015), (C. -H. Chen, 2011). In this project, we proposed an autonomous robotic dishwasher arm with an RGB camera. It simplifies the system by only using one component, reducing the dependency between different sensors. It would also help in keeping the cost down. This is a big advantage since it is one of the barriers to introducing current solutions to the market.

In our solution, we proposed six algorithms to accomplish picking up dirty dishes to place in the dishwasher. First, YOLOv7 is used to detect and classify six types of objects: plates, bowls, cups, forks, knives, and spoons. Second, the highest object is detected by monocular camera depth estimation. Third, we find the best grasping spot for each object. Fourth, the 6-D pose of the object is calculated to manipulate the joint angles of the robot arm. Fifth, the pixel information from the RGB camera is converted to real-world coordinates. Finally, using a neural network, inverse kinematics calculation and joint angle optimization were performed to control the robot arm. We set up the assumed situation for the project as shown in Figure 1 below. We put the robotic arm between the sink and the dishwasher. This robotics arm consists of three links and a two fingers gripper. The focus of this project was the first five components. Due to the unavailability of hardware and limited simulation software at our disposal, not a lot of emphasis was done on the final step where the object is placed in the dishwasher. This component doesn't involve AI and will be dealt with in a brute-force manner.

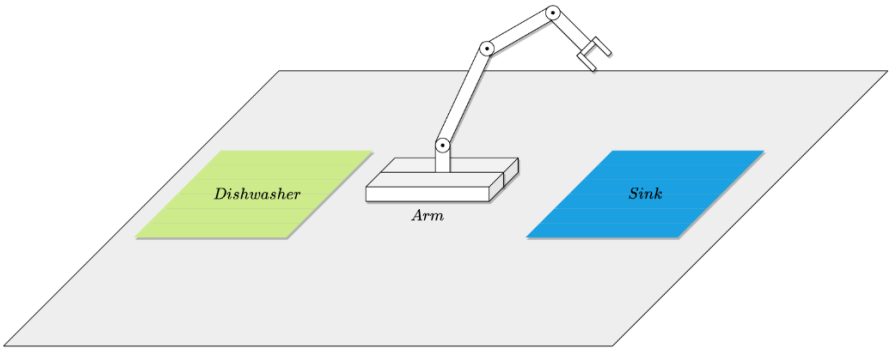


Figure 1 Setup Environment

2 Problem Definition

2.1 Current Problem

The objective of this project is to automate the process of loading a standard dishwasher. This problem is significant as solving it can be further generalized to solve other problems into the field of domestic robots. Solving this problem will give us significant insight in the fields of robotic planning and perception. The problem consists of having a robotic arm placed between a sink filled with dishes and a dishwasher. The robotic arm is fixed and can reach any position of the sink and dishwasher. The sink can contain six different types of dishes: fork, knife, spoon, plate, bowl, and mug. Any combination of those dishes is possible, and they can be placed in any orientation in the sink.

2.2 Proposed Solution

The main problem can be broken down into multiple smaller problems. The first step to be addressed is to classify the objects in the sink. The second step is to identify which of those objects should be dealt with first by finding the highest object. The third step is to identify a grasping point on the object. The fourth step is to orient the robotic gripper to efficiently handle the target object. The fifth step is to locate the object in the real-world coordinate system. The sixth step is to grasp the object and perform the motion planning of the robotic arm. Finally, the last step is to place the object in the dishwasher. The process repeats until there are no dishes in the sink. Figure 2 showcases a high-level view of our proposed method.

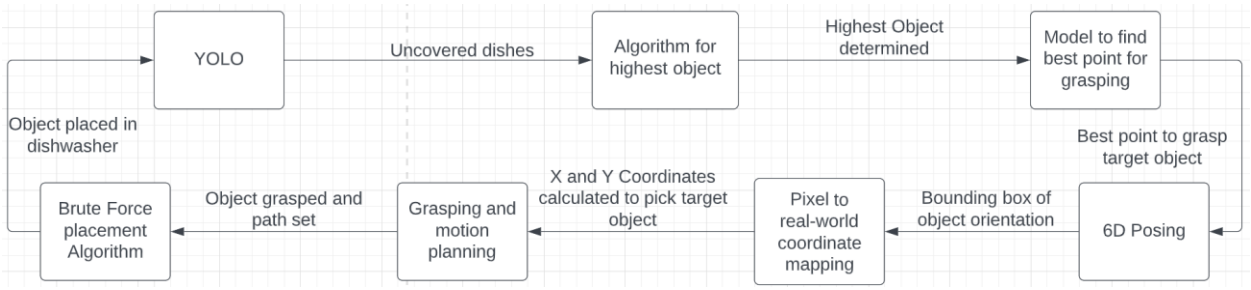


Figure 2 Flowchart of proposed solution High Level

2.3 Project Assumptions

The autonomous dishwasher arm is designed with the following assumptions in mind. First, the arm can handle a total of six different objects, which include a fork, knife, spoon, mug, plate, and bowl. Second, the objects that the arm encounters will always be the same type, such as metal utensils, regular mugs, white circular plates, and small circular bowls. Third, the objects can be placed randomly in the sink and do not need to be arranged in any particular order. Fourth, the sink is large and clear of any obstacles, allowing the robot hand to move freely without colliding with the walls of the sink.

2.4 Composition of Robotic Arm (Hardware)

To design a robotic arm that can autonomously load dishwashers, certain physical requirements must be considered. Generally, the robotic arm should have different actuators such as motors or servos to control its movement, along with power sources and appropriate controllers. Additionally, the inclusion of motion sensors such as inertial measurement units, odometers, force/torque sensors, and accelerometers is necessary. Since these are typically present in most commercially available robotic arms to control their movement, we opted to base our design on a readily available model rather than creating our own.

As our design heavily relies on image processing, classification, and machine learning models, we have also included an RGB camera and an embedded system in the design. The choice of embedded system depends on factors such as software dependency, pricing, and the intended monetization of the final product.

For the robotic arm itself, we have selected an open-source three-link 5-DOF 3D printed arm by BCN3D (BCN3D MOVEO: A fully Open Source 3D printed robot arm, 2016). This allows for customization of the link setup based on the size of the sink and dishwasher, adjustable gripper configurations, and affordability in terms of materials used. A monocular RGB Camera will be used just like (Kaimeng Wang, 2022), (Francisco Gomez-Donoso, 2019).

Finally, we have identified three potential options for the embedded system. One option is to use a Raspberry Pi 4 in conjunction with a Google Coral Accelerator, which would offer a portable, fully functional arm. Another option is to use an NVIDIA Jetson, which would provide faster computation at the expense of higher cost, greater size, and fewer online resources compared to the Raspberry Pi.

A third option is to utilize cloud computing, allowing the robot to access data centers via a wireless internet connection. This could be more suitable for a subscription-based model, in which multiple customers could connect to the data center where calculations are carried out and use the arm's features. This approach would also reduce hardware requirements, as a smaller microprocessor may be sufficient if it is able to transmit information to the data center at the required speed. However, this option would require an internet connection and should be explored after the final product has been developed.

3 Methodology

3.1 Image Processing

Using a monocular camera to capture the dishes, YOLOv7 is used to classify the objects in the sink. YOLOv7 is a state-of-the-art real-time object detection used in many applications such as autonomous driving, robotics, and medical image analysis. This deep learning model is faster and more accurate than other models that perform real-time object detection (Wang, 2022).

In the process, the highest object in the pile of dishes is the desired target. To recognize the highest object from the dishes, depth estimation is done using MiDaS (Rene Ranftl, 2020). MiDaS is a deep learning model that can estimate the depth of objects using a monocular camera.

3.2 Pose and Grasp Estimation

To grasp the objects, a two-finger grasping hand is used. The robotic hand used in our proposed method is one used in another robotic arm that can pick dishes (Hiroki Suzuki, 2021). This hand is discussed more in detail in section 4.5. To locate the best grasping spot, different approaches are used depending on the objects. For the utensils and mugs, predetermined spots are used to locate the best grabbing spot. The mug handle and the neck of utensils are chosen as grabbing spots. YOLOv7 is used to locate those areas.

For the bowls and plates, their outside edges are considered the best grappling spot. To locate the desired grasping spot, depth estimation is used in parallel to an edge detection method. The Canny Edge Detection is used and returns the outside edge. Then the edge that is the highest in the sink is considered the best grasping spot. The chosen grasping spot for those dishes was inspired by the following paper (Ashutosh Saxena, 2010), discussed in further detail in 4.2.

To grasp the desired location correctly, the end effector needs to orient itself accordingly to the object's orientation. Gen6D was used to perform the pose estimation of objects. It is a generalizable model-free 6-DoF object pose estimator that only requires an RGB image input (Yuan, 2022). With the obtained pose estimation and object classification, the 6DoF is obtained via offline pre-computation.

3.3 Motion Planning

Upon determining the location of the object and the designated grasping location, the robotic hand must reach that position. Since the movement of a robotic arm is restricted to its joint angles, it must compute the required joint parameters to reach the desired position. This mathematical process, referred to as inverse kinematics, is accomplished using numerical optimization methods, with the potential for assistance from an artificial neural network if the aforementioned method is unable to yield a solution. The arm subsequently attains the desired joint angles, grasps the object using a two-finger grasping hand, and places it inside the dishwasher using a brute-force algorithm

4 Related Work

This section is used to showcase some similar work as our proposed solution. This is used to get further information on some of our methods and showcase the feasibility of our method.

4.1 Object Detection

To achieve autonomous grasping, the robot needs to recognize the target. Then, it needs to determine the location to grasp the object. A paper was published for grasping of irregular objects using YOLO. The proposed work uses a two-finger gripper to grasp irregular objects (Shitian Zhang, 2020).

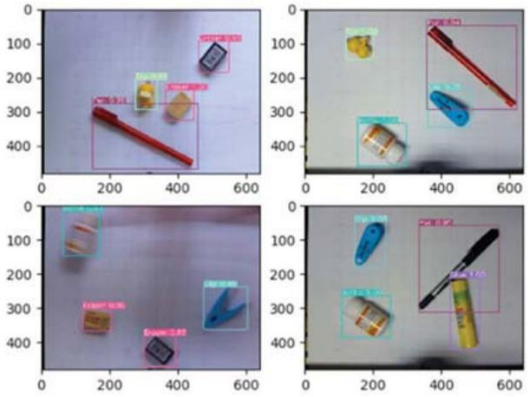
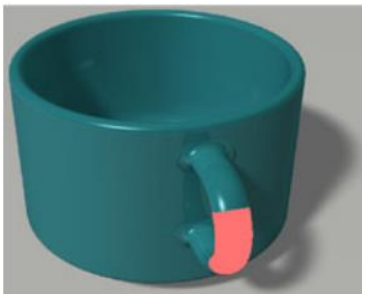


Figure 3 YOLO Detection (Shitian Zhang, 2020)

4.2 Grasping Spot

Ashutosh Saxena Et Al has proposed a vision-based system for grasping novel objects in clustered environments (Ashutosh Saxena, 2010). They tested their scenario to grasp objects in a normal kitchen environment. In their research, they showcased that a good grasping point for cups is their "handles" and that for bowls and plates the "lips" or edges of them are a good point (Ashutosh Saxena, 2010).



(a) Mug Grasping Point

(b) Plates Grasping points

Figure 4 Grasping Points of Objects (Ashutosh Saxena, 2010)

4.3 Depth Estimation

With the development of computer vision, grasping objects using 3D machine vision is becoming more popular since it does not require calculating 3D models. A proposed work was done to use CNN based method for monocular vision in robotic grasping (Xiaojun Wu, 2022). Their method

is divided into three steps: object detection, monocular depth estimation, and key point estimation. One big contribution from this paper is they showed that with monocular depth estimation, it was not necessary to use stereo vision or other sensors method to calculate distances. They were able to achieve an 80% grasping rate using their approach. Figure 5 illustrates how the robotic arm gets an object. It detects it, then a key point is computed from the depth estimation for each part. Figure 6 illustrates the grasping algorithm from (Xiaojun Wu, 2022)

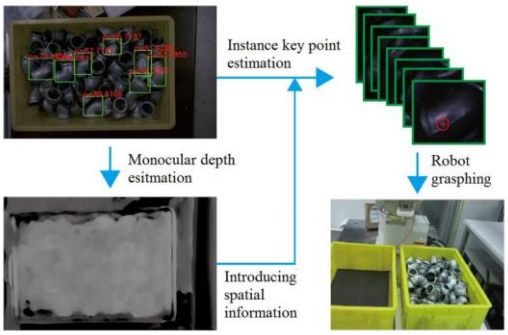


Figure 5 Monocular vision-based robotic grasping (Xiaojun Wu, 2022)

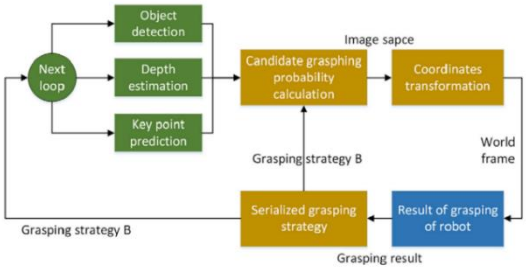


Figure 6 Grasping Algorithm (Xiaojun Wu, 2022)

4.4 Pose Estimation

Recently, there's more research on 6DoF poses for unseen objects without training or finetuning on test objects. Before, 6DoF object pose estimators can only target a specific object or a specific object category. Generalizable 6DoF object estimators were presented recently in ECCV2022 or CVPR2022. Most of the following method works by using a CNN to train a model using a template and a real image and finding their similarity. Then, at runtime, the network is applied to images of the unseen object to predict the pose (Van Nguyrn, 2022) shown in Figure 7. Some of the proposed methods at the ECCV2022 or CVPR2022 was able to achieve pose estimation with limited hardware using just an RGB camera and some did not require a 3D model (Jiaming SUn, 2022), (Yisheng He, 2022), (Muhammad Zubair Irshad, 2022).

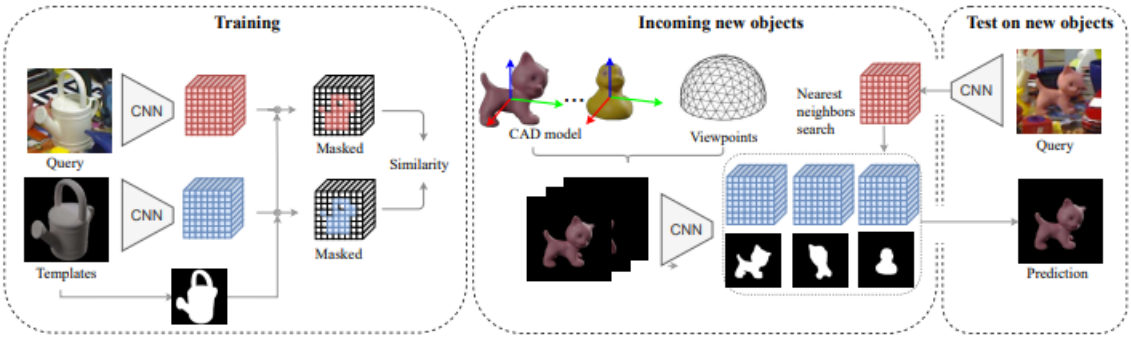


Figure 7 Generalizable object pose estimator steps (Van Nguyrn, 2022)

4.5 Robotic Hand

To accomplish the dishwasher robot arm system proposed in this report, it is necessary to use a robot hand that can grasp tableware stably. Therefore, the robot hand proposed by Suzuki et al. is applicable (Hiroki Suzuki, 2021). The component of the two-finger hand is shown in Figure 8.

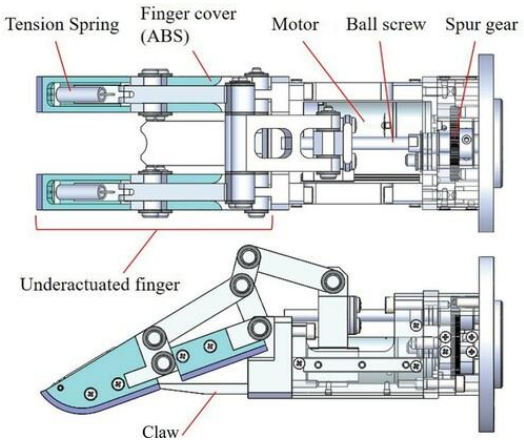


Figure 8 Components of the two-finger hand (Hiroki Suzuki, 2021)

The proposed robotic hand has been developed to grasp a variety of tableware. As shown in Figure 9 below, it can pick up dishes even when they are stacked on top of each other. We chose this robotic hand to pick the objects in our proposed method.

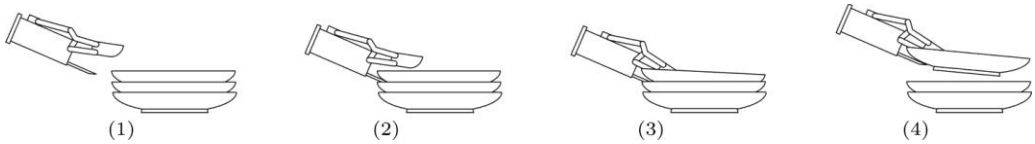


Figure 9 Grasping Stacked Dishes (Hiroki Suzuki, 2021)

4.6 General Design

To successfully have an autonomous dishwasher robot arm, it needs to perform grasping of objects correctly. The following paper goes in-depth into different methods to grasp objects using vision-based using parallel grippers. The three main components they mention for efficient grasping are the following: object localization, object pose estimation, and grasp estimation. The methods reviewed for object localization consist of object localization without classification, object detection, and instance segmentation. This step provides the region of the target object. The object poses part is used to find the 6D object pose. The reviewed methods were correspondence-based, template-based, and voting based. This helps orient the hand accordingly to the pose of the object. Finally, the grasp estimation is used to constrain the grasp from one direction. The reviewed methods were the 2D planar grasp and the 6DoF grasp method. The proposed process uses an RGB-D camera and one parallel gripper. The following figure illustrates a high-level view of their process. This paper goes over different combinations of these three tasks to accomplish robot grasping (Guoguang Du, 2020).

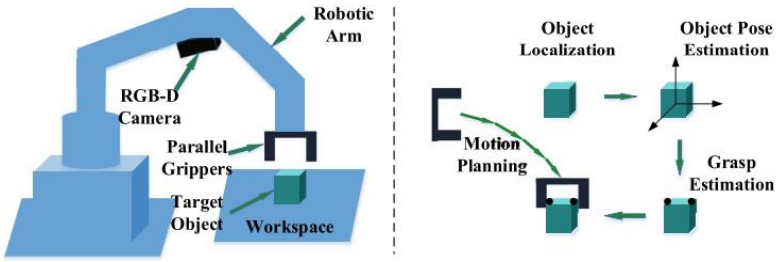


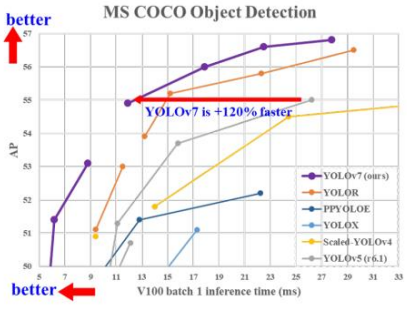
Figure 10 High-Level View Process - Related Work (Guoguang Du, 2020)

In this paper, they tested different 2D object detection methods for the object localization part. They reviewed two-stage methods and one-stage methods. The one-stage methods achieved higher accuracy and high speed. One of the proposed methods was YOLO. The paper reviewed mostly 6D object poses. One of the methods reviewed is the templated-based method. This method consists of finding the most similar template from the templates that are labeled with Ground Truth 6D object poses. Finally, one proposed method for the 6DoF Grasp was a method based on the shape of the object. As mentioned, the 6D pose of the object can be estimated. The 6DoF grasp can be obtained via offline pre-computation. It is one of the most popular methods used for grasping system (Guoguang Du, 2020). The 6DoF grasp poses can be stored in a database. If the 6DoF grasp pose does not exist, analytical methods considering kinematics and dynamics formulation can be used. Many analytical methods exist for a known 3D object, but Force-closure is one method (Guoguang Du, 2020).

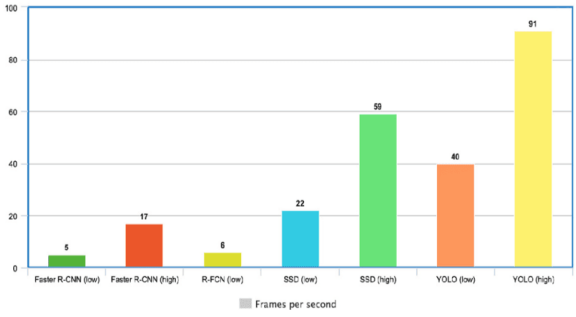
5 Literature Review

5.1 YOLOv7

YOLOv7 is a state-of-the-art real-time object detection used in many applications such as autonomous driving, robotics, and medical image analysis. Its accuracy and speed have proven to be up to 120% quicker and more accurate than other methods in Figure 11 (Wang, 2022). The application requires detecting dishes and cutlery in real-time. Therefore, its detection speed is important to accurately detect those objects.



a) Comparison between different version of YOLO (Wang, 2022)



b) Comparison between different object detectors (Hui, 2018)

Figure 11 YOLOv7 vs Other Object Detector

YOLOv7 works in the following four steps. First, it performs a residual block by dividing the image into N grids, each of them divided into SxS equidimensional regions. Next, a bounding box regression is performed. Each of the N grids divided during the residual block process is responsible for detecting and localizing the objects contained within it. However, many duplicate predictions occur because multiple grids predict the same object with different bounding boxes. Therefore, as a third step, non-max suppression is required to suppress all bounding boxes with low probability scores. To further increase the probability, Intersection over Union (IoU) can be applied between duplicated bounding boxes. IoU can be computed with the following equation.

$$IoU = \frac{Area\ of\ overlap}{Area\ of\ Union}$$

Finally, based on the probability, the bounding box that achieves the highest IoU is the area selected. Through the above four steps, YOLOv7 outputs the bounding box coordinates "$(x_{left\_bottom}, y_{right\_top}, width, height)$" as the result of object detection. YOLOv7 provides several pre-trained models as weights for object detection using the COCO's dataset. As shown in Table 1. YOLOv7-E6E provides the highest score of. This model was chosen for its accuracy rather than its detection speed since dishes and cutlery do not move

Table 1 Comparison with different pretrained weights (Wang, 2022)

Model	Test Size	AP^{test}	$AP_{50}{}^{test}$	$AP_{75}{}^{test}$	batch 1 fps	batch 32 average time
YOLOv7	640	51.4%	69.7%	55.9%	161 *fps*	2.8 *ms*
YOLOv7-X	640	53.1%	71.2%	57.8%	114 *fps*	4.3 *ms*
YOLOv7-W6	1280	54.9%	72.6%	60.1%	84 *fps*	7.6 *ms*
YOLOv7-E6	1280	56.0%	73.5%	61.2%	56 *fps*	12.3 *ms*
YOLOv7-D6	1280	56.6%	74.0%	61.8%	44 *fps*	15.0 *ms*
YOLOv7-E6E	1280	56.8%	74.4%	62.1%	36 *fps*	18.7 *ms*

5.2 Highest Object Detection

If we assume that plates would be stacked after a meal, the system needs to know which object, such as dishes or cutlery, is in the highest position to pick up. Depth estimation is necessary for the robot hand to successfully pick up the highest object. Lidar, depth cameras, and stereo cameras are commonly used to obtain depth information. However, these devices are expensive. Therefore, to keep costs down, we introduced a system that uses a monocular camera to perform depth estimation. To achieve monocular depth estimation, we used a robust monocular depth estimation model that has been pre-trained on datasets shown in Table 2. This pre-trained model is called MiDaS (Rene Ranftl, 2020). Many monocular depth estimation models require many training data sets to achieve high accuracy. However, it is difficult to collect many datasets without distinct characteristics and biases for training data. In this model, multiple datasets can be mixed during training, even if the annotations are incompatible. Therefore, a larger number of datasets can be used for training and more accurate monocular depth estimation can be achieved in various environments. Thus, this model could be applied to our situation to estimate to depth to pick up dishes. In this study, they trained deep neural networks on five large datasets with RGB-D, SfM,

and stereo: the ReDWeb, MegaDepth, WSVD, DIML Indoor, and 3D Movie datasets as shown in Table 2. The models trained on these datasets were validated on DIW, ETH3D, KITTI, NYU, and TUM. A dynamic subset featuring humans in indoor environments was also used for testing.

Table 2 Dataset used in the monocular depth estimation (Rene Ranftl, 2020)

Dataset	Indoor	Outdoor	Dynamic	Video	Dense	Accuracy	Diversity	Annotation	Depth	# Images
DIML Indoor [31]	✓			✓	✓	Medium	Medium	RGB-D	**Metric**	220K
MegaDepth [11]		✓	(✓)		(✓)	Medium	Medium	SfM	No scale	130K
ReDWeb [32]	✓	✓	✓		✓	Medium	**High**	Stereo	No scale & shift	3600
WSVD [33]	✓	✓	✓	✓	✓	Medium	**High**	Stereo	No scale & shift	1.5M
3D Movies	✓	✓	✓	✓	✓	Medium	**High**	Stereo	No scale & shift	75K
DIW [34]	✓	✓	✓			Low	**High**	User clicks	Ordinal pair	496K
ETH3D [35]	✓	✓			✓	**High**	Low	Laser	**Metric**	454
Sintel [36]	✓	✓	✓	✓	✓	**High**	Medium	Synthetic	(Metric)	1064
KITTI [28], [29]		✓	(✓)	✓	(✓)	Medium	Low	Laser/Stereo	**Metric**	93K
NYUDv2 [30]	✓		(✓)	✓	✓	Medium	Low	RGB-D	**Metric**	407K
TUM-RGBD [37]	✓		(✓)	✓	✓	Medium	Low	RGB-D	**Metric**	80K

5.3 Edge Detection

Edge Detection is needed in our process to grasp plates and bowls. The Canny Edge Detection technique was used in our case to locate the edge of both objects. Canny Edge Detection is one of the most used image processing tools to detect edges in a robust manner (Poonam Dhankhar, 2013). It is a popular technique as it has adjustable parameters which can affect the computation time and effectiveness of the method. The method recognizes edges by looking for sudden changes in pixel intensity in neighboring pixels. The method consists of four main steps, three steps to extract the edges and one to reduce the noise. The steps consist of noise reduction, calculating the intensity gradient of the image, suppression of false edges, and hysteresis thresholding.

The first step of the process is to reduce noise before computing the edges. Gaussian blur filter is applied to the image to reduce details that could be considered edges. The second step is to calculate the intensity gradient of the image. The Sobel kernel is applied and calculates both the magnitude and direction of each pixel. Then, unwanted pixels are removed by using non-maximum suppression. Each pixel is compared to neighbor pixel. Finally, Hysteresis thresholding is applied. It is a threshold that removes further pixels. If the gradient magnitude value is higher than the threshold, those pixels are kept since they are recognized as strong edges. On the other end, if the gradient magnitude is lower than the smaller threshold, the pixels are suppressed. Finally, any pixels between the two thresholds are a candidate to stay in the final edge map and are considered weak pixels. If those “weak” pixels are connected to a strong edge, they are kept in the final edge map (Poonam Dhankhar, 2013).

Despite the number of detection techniques, this method is popular. It contains several parameters that can be modified to increase its accuracy. The size of the gaussian filter can be modified to control the blurring effect and the threshold hysteresis control what information is kept. It this report, the hysteresis threshold was tuned to obtain the best results.

5.4 6D Pose

Estimating the orientation and location of an object is an important step in our process. It would allow the robot arm to align according to the orientation of the object. Some of the available pose estimators can only be used for a specific object in their training data (Yann Labbé, 2020). Some models can generalize on unseen objects, but it requires high-quality 3D models (Martin Sundermeyer, 2020). The Gen6D is a model that avoids those issues. The model consists of an object detector, a viewpoint selector, and a pose refiner. This model avoids the issues mentioned. This model is generalizable since the pose estimator can be applied to an arbitrary object without training. It is model-free and only needs reference images of an object with known poses. Finally, it can work with simple input and only takes RGB images and does not require additional object masks or depth maps (Yuan Liu, 2022).

Gen6D applies image-matching to estimate an object pose in a coarse-to-fine manner. The model works by first detecting an object by matching the reference image to the query image. Then, a viewpoint selector matches the query images to the reference image to produce an initial pose. Finally, that pose gets refined by a pose refiner to get a more accurate pose. Those steps can be seen in Figure 12.

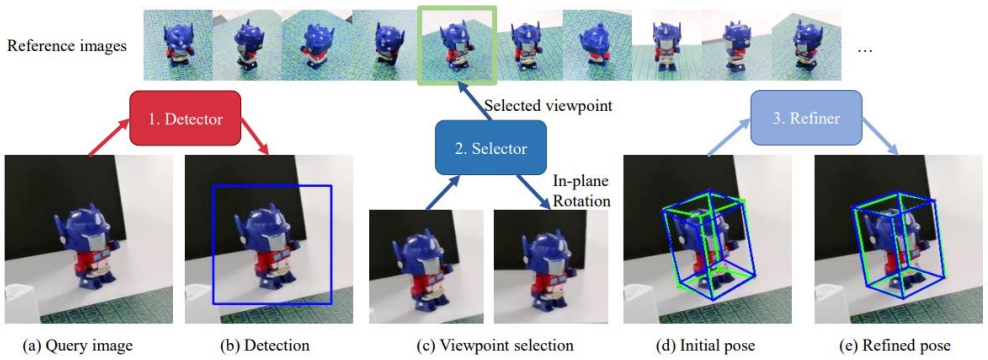


Figure 12 Gen6D Steps (Yuan Liu, 2022)

Having a noisy background is a challenge for getting an efficient viewpoint selector. Also, if there's no perfect match between the reference image and the query image, there is the possibility of multiple matches. To avoid this issue, Gen6D uses a neural network to compare the query image to the reference image and calculate a similarity score. The unavailability of a pose refiner was a challenge for their pose refiner. Usually, pose refiners renders an image on the input pose and then match the rendered image with the query image to refine its pose. However, without a 3D model, this step becomes difficult. To address this issue Gen6D uses a 3D volume-based pose refinement method. Using the query image and the input pose, it finds multiple references that are near the initial estimated pose. After, to refine the input pose, the constructed feature pose is matched against the features projected from the query image by a 3D CNN. Figure 13 and Figure 14 illustrate the Gen6D architecture for the estimator and the pose refiner.

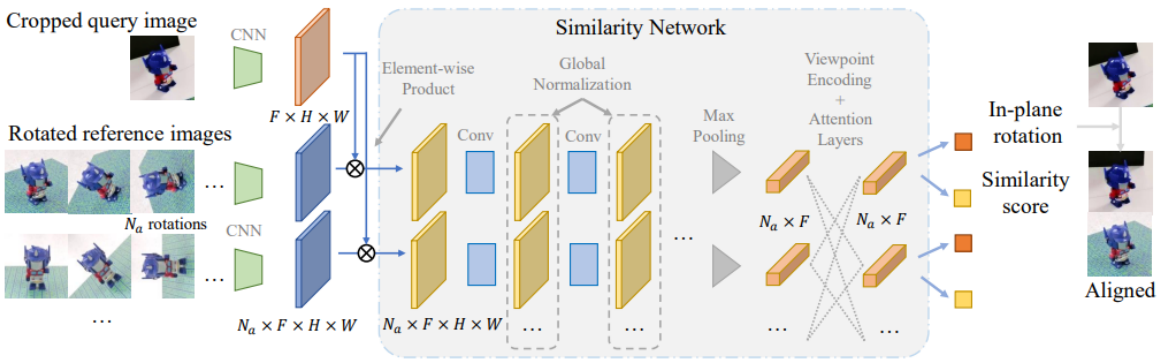


Figure 13 Gen6D Pose Estimator Architecture (Yuan Liu, 2022)

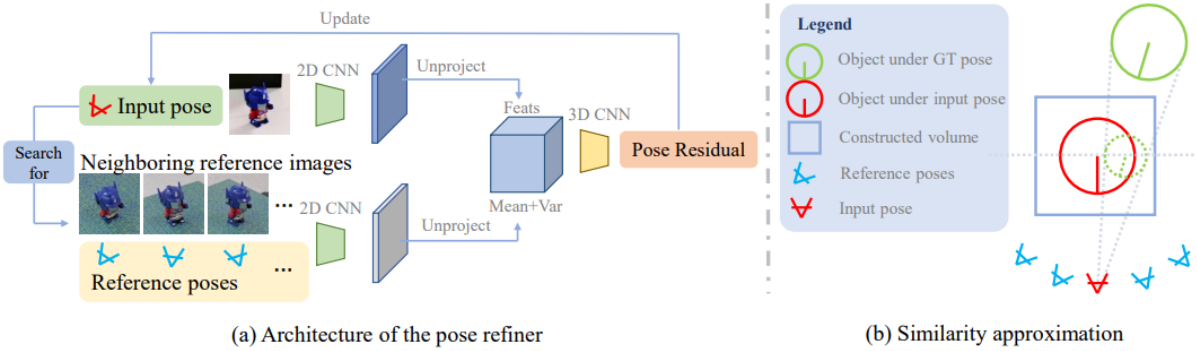


Figure 14 Gen6D Pose Refiner Architecture (Yuan Liu, 2022)

5.5 Pixel to Real-Coordinate Mapping

For the robotic arm to locate the target object, the location of the object should be deduced from the data gathered from the captured image. In other words, the data location of the object should be converted from virtual data to real data useful in the physical world. If the center point of the target object is found to be x,y,z in the pixel form, the output of the mapping algorithm is x',y',z' where each of these coordinates is the distance in the x, y and z axis respectively. In previous work, this was achieved by predetermining the field of view of the camera. The camera was set such that the frame captured was kept constant at for example 480-pixel x 640 pixels. Keeping the frame constant would enable us to measure the frame in centimeters and calculate the ratio by dividing the predetermined distance in pixels and the predetermined distance in centimeters.

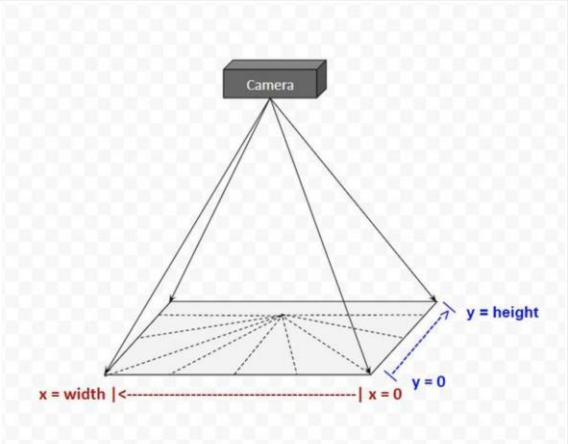


Figure 15 Example of a controlled captured field of view

This method would yield accurate results. However, given the limited tools at our disposal, controlling the frame proved to be tricky. This was the case because a phone camera was used to take the pictures. Thus, a workaround was found by using the Aruco marker, which would find the pixel-to-centimeter conversion ratio without having to control the camera frame. An Aruco marker is a square marker composed of black and white patterns. The borders of the square are black while the inner part is a binary matrix that determines the marker's id. The id helps detect errors and is useful for troubleshooting. The white inner patterns of the marker also dictate the size of the square. This pattern eases the detection of the marker in computer vision applications.

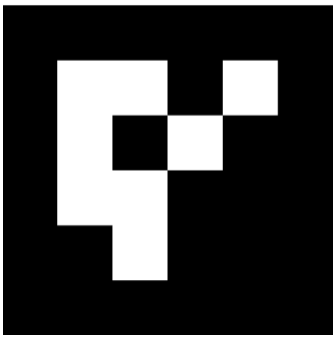


Figure 16 4x4 Aruco marker

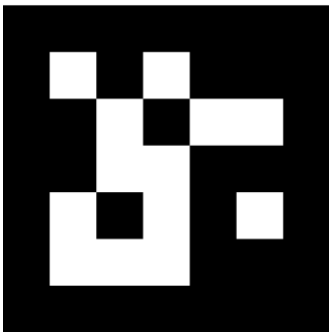


Figure 17 5x5 Aruco marker

Aruco markers are a useful and popular tool in computer vision as they are cost-effective and readily available. Using an Aruco marker will help to find an easy and cost-efficient way to find a conversion from pixel to centimeters. The second step to locating the target object is to mark a point of reference. The location of the point of reference is of no significance to our application. The objective is to mark this point and store it such that the robotic arm can find objects in reference to it. In our application, we used BLOB detection to mark this point and the x and y distances were found from said point. BLOB stands for a binary large object, and it is a group of connected pixels forming a particular shape. In our application, our chosen shape was a black dot on a white background. This shape was chosen for its ease of accessibility and reproducibility.

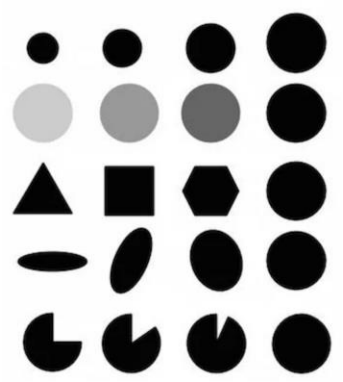


Figure 18 BLOB examples

Due to our setup, the z distance will be constant. This is caused by the fact that the camera will be placed above the sink and will not change throughout the process. Thus, we do not need to calculate the distance in the z-direction as it will remain constant and recomputing it will be a redundant step.

Thus, in conclusion, in this component the mapping algorithm first detects an object, in our final application the object will be detected using YOLO, however for time limitation purposes the object detection in this component was based on pixel color. As a future step this component will be merged with the YOLO component of our algorithm. However due to time limitations we completed each component independently. The target object must be of a dark color placed on a white background. The algorithm will detect the object and draw a bounding box around it. We now have the dimensions of the object in pixels as well as the center point of the object.

The second step of the algorithm is to have a conversion ratio. In other words, we need to know how to convert distances from pixel to cm. This is achieved with the use of an Aruco marker. In our application, we conducted tests with the 5x5 marker. The main objective of using the Aruco marker was that it is readily available online and easy to print and use. It is also efficient as most libraries such as OpenCV have built-in methods that makes using them straight forward in most applications. In addition to that using this approach made the algorithm simpler and more concise as alternative approaches would've been more time consuming and more prawn to error. Thus, by using this marker we managed to find an efficient and fast method to find the conversion ratio from pixel to cm. The idea is simple, since we predetermined the marker's circumference to be 20 cm, we import the dictionary for the 5x5 Aruco marker. We then detect the marker in our input image and measure the circumference. We then divide the pixel circumference by the predetermined circumference for a 5x5 square and this would be our ratio to convert from pixel to cm.

The final component of the algorithm is to detect blobs such that the robotic arm can refer to a known point and locate the target object with reference to that point.

5.6 Inverse Kinematics

Upon acquiring real-world coordinates linked to the grasping point, the robotic arm must autonomously navigate towards the point, grip the object, and place it inside the dishwasher. For this, we shall consider the robotic arm mentioned above, and analyze its kinematics. Consider the design of a three-link planar arm (modelled after the arm) given below.

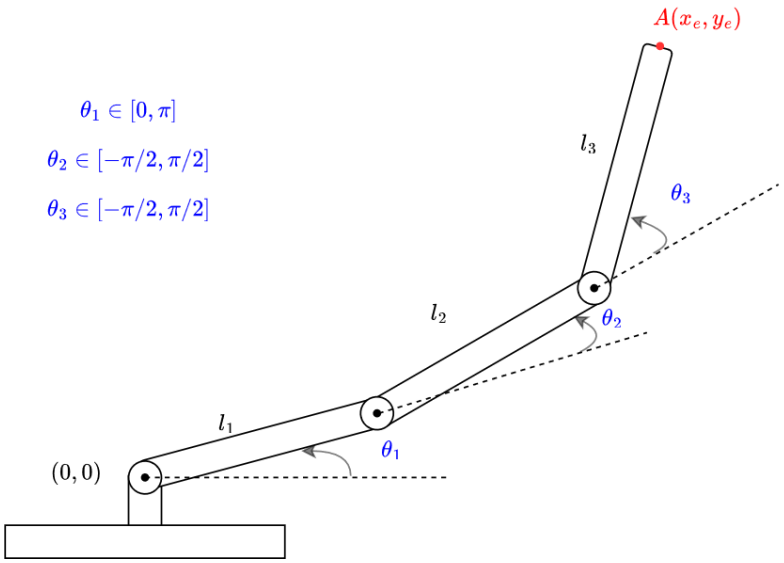


Figure 19 Three-Link Planar Robotic Arm

The variables present in Figure 19 are linked via the following equations:

$$x_e = l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) + l_3 \cos(\theta_1 + \theta_2 + \theta_3)$$

$$y_e = l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2) + l_3 \sin(\theta_1 + \theta_2 + \theta_3)$$

Where (x_e, y_e) are the end-effector coordinates, $(\theta_1, \theta_2, \theta_3)$ are the joint parameters, and (l_1, l_2, l_3) are the lengths of each link. The equations above are independent, non-linear equations. Determining the position of the end-effector (x_e, y_e) via given joint parameters $(\theta_1, \theta_2, \theta_3)$ is termed as the forward kinematics of the arm. In this case, a set of given joint parameters leads to one solution which is usually solved analytically.

On the other hand, determining a set of joint parameters given the end-effector coordinates is quite challenging given the non-linearity of the equations. This is defined as the inverse kinematics problem. The solutions for these depend on the number of constraints (in this case, the end-effector coordinates), and the number of degrees of freedom (in this case, the arm links). Let n be the number of degrees of freedom in a system, and m the number of constraints (Section II. MODELING, 2020).

Table 3 Degrees of Freedom/Constraints Cases

Case	Solutions	System Term
$n < m$	$None$	$Overconstrained$
$n = m$	$None\ or\ Finite$	$-$
$n > m$	$None\ or\ Infinite$	$Underconstrained/Redundant$

There are a few exceptions to these rules but will not be considered in detail. For our three-link planar arm, the number of degrees of freedom outnumbers the number of constraints (redundant), and thus we either obtain infinitely many solutions, or none.

While inverse kinematic problems can be solved analytically, these usually rely on fixing joint parameters, grouping links together, or arbitrarily reducing the number of degrees of freedom, which would also reduce the flexibility of the arm. This project focuses on using machine learning models, as well as numerical methods, to solve this problem and acquire an optimal solution.

5.6.1 Artificial Neural Networks

Solving independent, non-linear equations using neural networks has been widely discussed due to the nature of deep learning activation functions also being non-linear and sinusoidal (Francesco Aggogeri, 2022), (Duka, 2014), (Jiaoyang Lu, 2022). As this project aims to determine joint parameters for a three-link arm without any physical hardware or training data, our approach is based on (Duka, 2014), where training data for the neural network is done via an iterative process of different sets of joint parameters using forward kinematics. The figure below shows this iterative process is carried out for links of equal length, along with random poses for the three links.

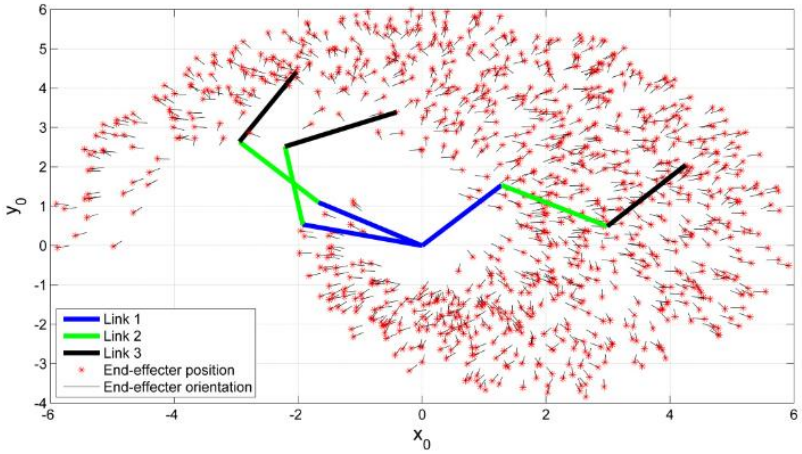


Figure 20 3-Link Planar Manipulator Simulation (Duka, 2014)

After obtaining the training data, the final sets of joint parameters are used as the training outputs, while the end-effector positions and orientations are considered as the inputs (along with the end-effector orientation). The neural network is thus set up in the manner below.

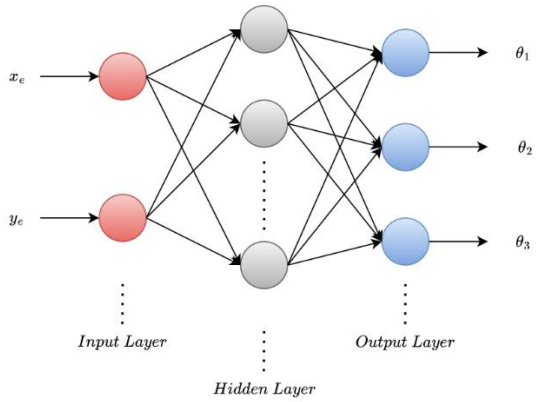
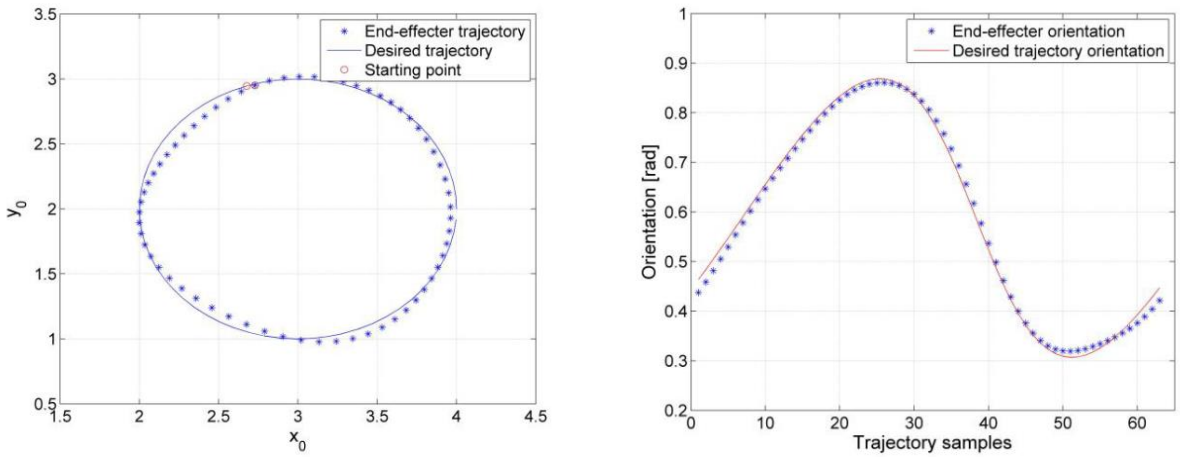


Figure 21 Architecture of Neural Network

With the hidden layer being composed of 100 neurons and the hyperbolic tangent as an activation function. The results, along with the expected values, are plotted below.



a) Trajectory b) Orientation

Figure 22 End-effector results versus desired (Duka, 2014)

5.6.2 Numerical Optimization

Another approach to solving the inverse kinematics problem is through optimization techniques. These usually consist of algorithms that assign a specific cost function, start from an initial guess of what the solution might be, and iterate until a solution is found. This solution is reached when the assigned cost function converges to a global minimum.

Many such algorithms exist, such as the Levenberg-Marquardt algorithm, Broyden-Fletcher-Goldfarb-Shanno (BFGS) gradient projection (Sugihara, 2009), (MathWorks, 2022). However, there are difficulties associated with this approach. For instance, if the initial starting (or guess)

point is inaccurate, the cost function might not converge in a specified number of iterations. Additionally, the algorithm might converge to a local rather than the global minimum, which would not lead to a solution. These drawbacks will be considered in our results and simulation section further ahead.

5.7 Placement Algorithm

A brute force approach was taken to address this step. The idea is that each classified object will have a hard coded location in the dishwasher. For example, plates will occupy the top section of the dishwasher. Depending on the average size of plates that will be used, all the section assigned to plates will be split equally. The coordinates for each plate slot will then be hard coded in a data structure. Once the algorithm identifies a plate, the robotic arm will place the first plate in the first available slot for a plate in the plate section of the dishwasher. The algorithm will then move on to the next available slot. The same process will be done for all other classes: forks, knives, spoons, glasses, and mugs.

6 Simulation Results

6.1 Classification of Objects

The first step of our process is for the robot arm to classify each object in the sink. YOLOv7 was used to detect the object. We used the pre-trained weights YOLOv7 and YOLOv7-E6E (Wang, 2022). YOLOv7-E6E had the highest accuracy, but the detection is slower. However, since the objects are not moving, this is not an issue. Hence, YOLOv7-E6E was used. These model weights are trained using Microsoft's COCO dataset. The COCO dataset on which YOLOv7 trains contains 80 classes. Cup, fork, knife, spoon, and bowl are among the 80 trained classes. However, plates are not among the trained weights. Using the pre-trained weights, we were able to detect with high accuracy the objects we mentioned except plates.

We also labeled 1000 objects for the 6 objects. We compared our train model to the pre-trained one from YOLOv7. For each comparison, we used the same pictures. We tested four cases increasing in difficulty each time. For each scenario, an overhead shot was used as the robot would be placed above the sink. The first scenario consisted of each object being separated by a certain distance and having no occlusion. It is to note that for each test scenario, we put a threshold for confidences under 0.5 and an IoU threshold of 0.45.

a) Manually Label Model

b) Pretrain Model

Figure 23 YOLOv7 Scenario 1

The second case scenario consisted of having a bit of overlap between each item and reducing the distances between each object shown in Figure 24. Each utensil was placed on top of the plate. This case scenario is pretty much like the first scenario Figure 23.

a) Manually Label Model

b) Pretrain Model

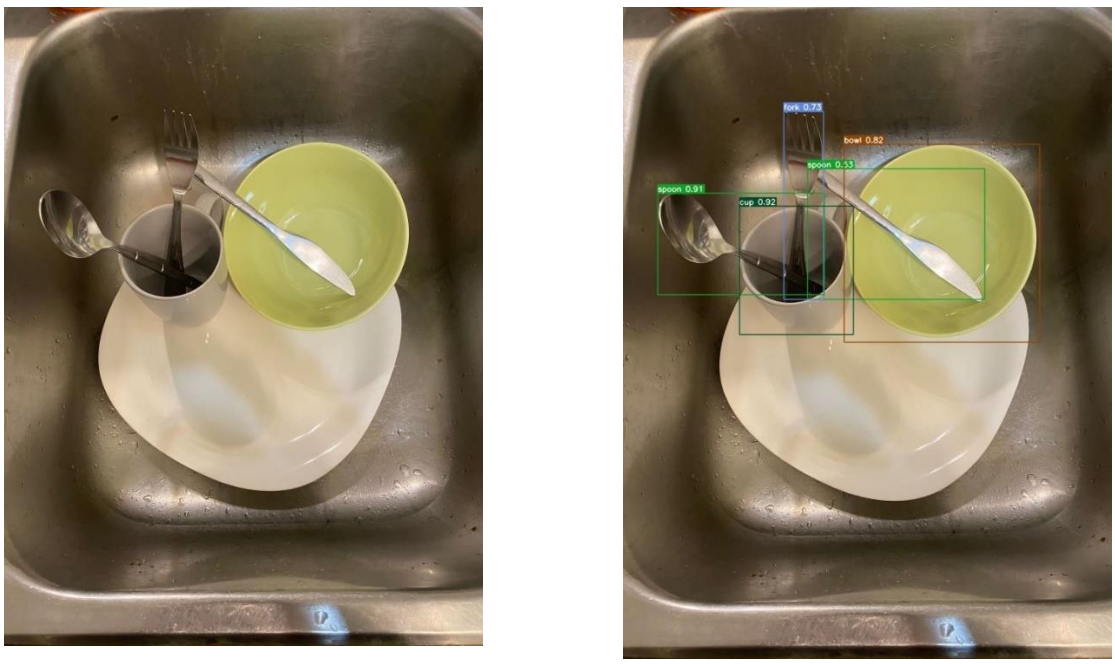
Figure 24 YOLOv7 Scenario 2

In the third case, the objects were closer to each other, more overlap occurred, and the objects were placed in different orientations as shown in Figure 25.

a) Manually Label Model b) Pretrain Model

Figure 25 YOLOv7 Scenario 3

Finally, the last test scenario was supposed to be the hardest one. This test scenario had the most amount of occlusion, and each object is placed in different orientations shown in Figure 26.



a) Manually Label Model b) Pretrain Model

Figure 26 YOLOv7 Scenario 4

From the figures shown above, the YOLOv7 pre-trained model predicted objects much more accurately than our pre-trained model. The only point where our model was better was that it was able to detect the plate in Figure 23, whereas the pre-trained model classified it as a bowl. However, this illustrates that with enough trained images, the model would probably be able to recognize the objects correctly. We can see that our train model was able to predict some objects in Case Scenarios 1 and 2 but could not identify objects in Case Scenarios 3 and 4. The images in which we trained the objects were mostly oriented in the same manner as the two first cases. Hence, the results from scenarios 1 and 2 were better than 3 and 4. The pre-train model that we used illustrated that it can accurately predict most objects even in different orientations (except the plate). The only object it missed was the mug in Figure 25. The confidence level of each bounding box was also higher for the pre-train model. Table 4 underneath illustrates the results throughout each scenario.

Table 4 Classification Results by Scenarios

	Manually Label Model		Pre-trained Model	
Scenarios	Correctly Counted	Average Confidence	Correctly Counted	Average Confidence
1	4/6	76 %	5/6	88%
2	3/6	70%	5/6	85%
3	0/6	0%	4/6	60%
4	0/6	0%	5/6	78%

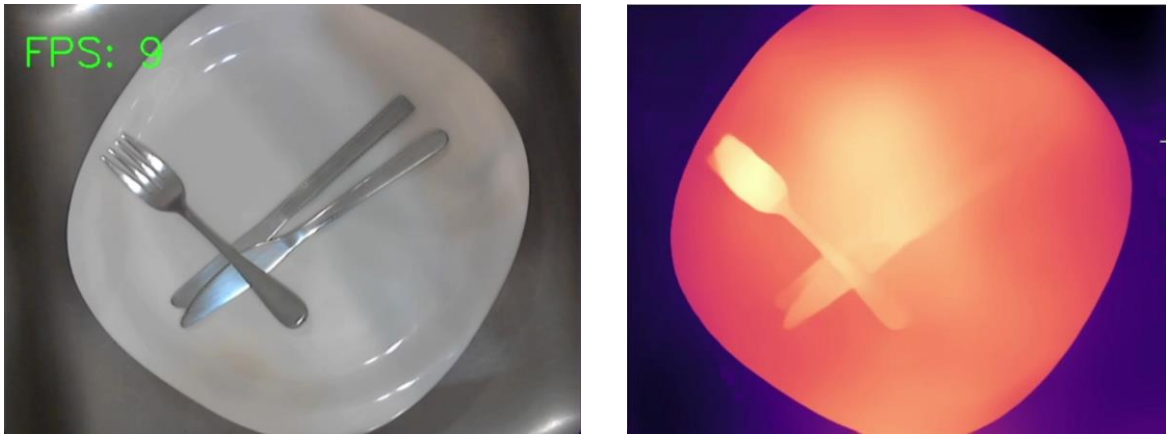
6.2 Detecting Highest Object

Detecting the highest object among the pile of dishes is a crucial step in our process. After classifying each object, the robot arms need to recognize which object to pick. Using the pre-train weights from MiDaS (Rene Ranftl, 2020), we accomplished depth estimation using a simple monocular camera. In real time, the model produces a depth map. Our approach would recognize the highest point in the depth map and map it to the object we classified in the first step of our process. It is to note that MiDaS provides four different pre-trained models. In our case, we used the largest model to achieve the highest accuracy, and the model was running in real time at an average of 8 FPS.

a) Depth Estimation Input b) Depth Estimation Output

Figure 27 Depth Estimation using MiDaS Test 1

Figure 27 illustrates the depth estimation using MiDas. The mug and the white bowl are detected as the highest objects, followed by the black bowl and the plate as the lower object. It is the correct result. After detecting those objects, we removed them from the setting. The only object left were the utensils and the plate shown in Figure 28. After removing those objects, the depth estimation was able to recognize the utensils. and outputs the fork as the highest object. The model accurately predicted the highest object for this scenario. It is one of the hardest case scenarios the model would face as the depth difference between those objects is small.



a) Depth Estimation Input b) Depth Estimation Output

Figure 28 Depth Estimation using MiDaS Test 2

6.3 Object Grasping

6.3.1 YOLOv7 Detection

After finding the highest object, the robot hand needs to detect a grasping point on the object. We followed the same approach as (Ashutosh Saxena, 2010) for the grasping spot of the mug and plates, and bowl. The predetermined grasping spot for the mug is its handlebar, and for the plate and bowl, it is their edges. For the utensils, we want the robot hand to pick them up at their neck, shown in Figure 29. This spot was chosen as it is easier to recognize a pattern in this area. Also, when the utensils are laying down one of the robot's fingers can go under the desired picking point.

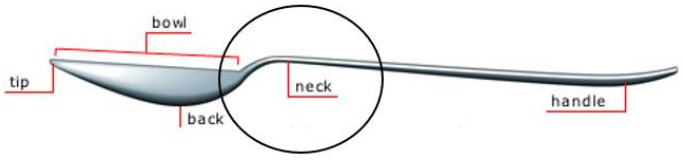


Figure 29 Utensil Neck

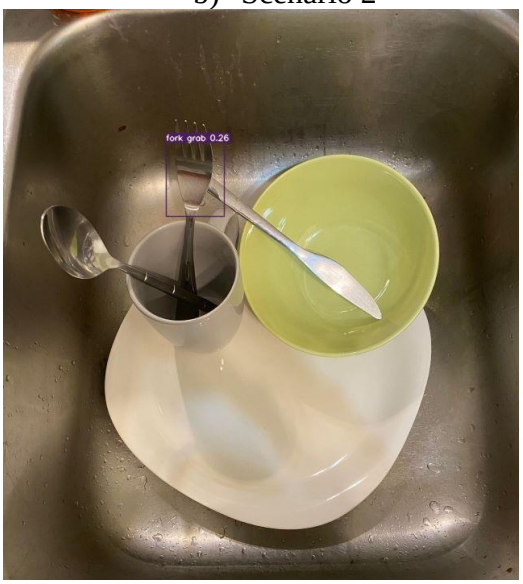
Figure 30 Label Grasping Point

Two methods were used two detect where the grasping points are located. The grasping points for utensils and the mug are found using YOLOv7. We manually labeled pictures and trained a model with YOLOv7 to recognize those spots Figure 30. Our model was able to detect the grasping points for the easier case scenarios but had some False Negatives in Scenario 3 and 4.

a) Scenario 1

b) Scenario 2

c) Scenario 3



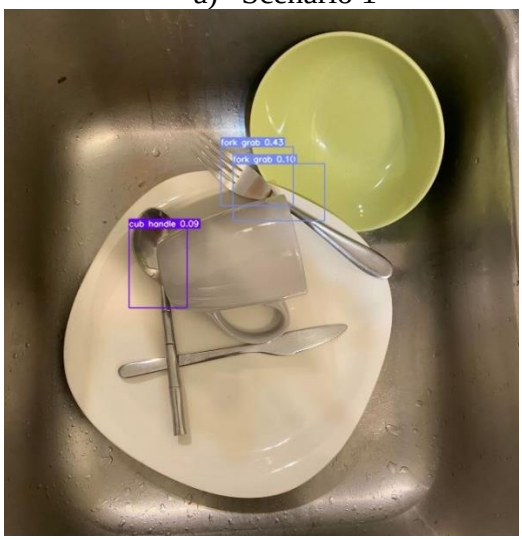
d) Scenario 4

Figure 31 Grasping Spot - Confidence 0.25

A confidence threshold of 0.25 was used for the results shown in Figure 31. Multiple confidence threshold was tested. This confidence level is the one that gave the highest accuracy. When lowering the confidence level, the model was able to detect more grasping locations but also provided a lot of false positives. Figure 32 illustrates the results obtained when a confidence level of 0.075 was used.

a) Scenario 1

b) Scenario 2



c) Scenario 3

d) Scenario 4

Figure 32 Grasping Spot - Confidence 0.075

Table 5 Grasping Spot Results

	Confidence 0.25			Confidence 0.75		
Scenario	TP	FP	FN	TP	FP	FN
1	4	0	0	4	0	0
2	4	0	0	4	1	0
3	1	0	3	1	2	3
4	1	0	3	2	0	2
Total	10	0	6	11	3	5

$$Accuracy_{confidence0.25} = \frac{TP + TN}{TP + FP + FN + TN} = \frac{10}{16} = 62.5\%$$

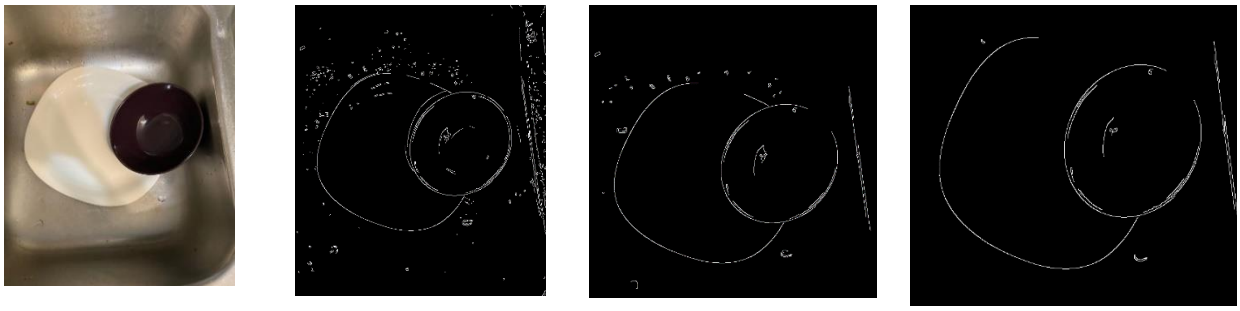
$$Accuracy_{confidence0.075} = \frac{TP + TN}{TP + FP + FN + TN} = \frac{11}{19} = 57.8\%$$

Table 5 and the equation above showcases the difference in results between both confidence level that was tested. We can see that the accuracy for the threshold of 0.25 gave a higher result. We believe that by training a bigger dataset of images, the accuracy would improve drastically, and the optimal confidence level would be higher. The accuracies that we observed were expected. After training our model, we obtained the following for our results.

Table 5 and the equation above showcases the difference in results between both confidence level that was tested. We can see that the accuracy for the threshold of 0.25 gave a higher result. We believe that by training a bigger dataset of images, the accuracy would improve drastically, and the optimal confidence level would be higher. The accuracies that we observed were expected. After training our model, we obtained the following for our results.

6.3.2 Edge Detection

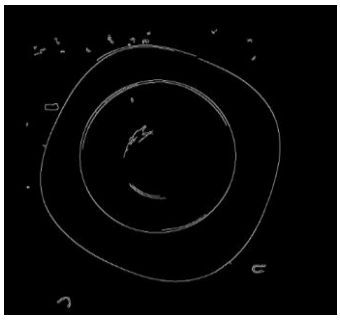
The Canny Edge detection was used to locate the edges of bowls and plates. The hysteresis threshold was tuned to find the best smooth edge detection without too much noise. Figure 33 illustrates some of the threshold used. For a threshold of 75-100, too much noise was included, and for a threshold of 75-400, a bigger part of the plate's edge was not detected. Hence, the threshold of 75-225 was chosen as the best one and was used in the following tests.



a) Input Image b) Threshold (75-100) c) Threshold (75-225) d) Threshold (75-400)

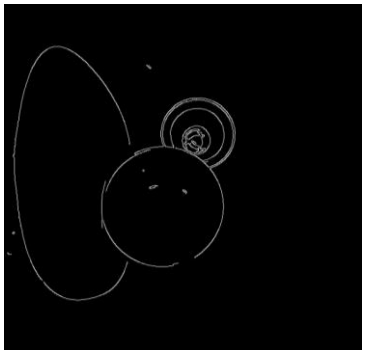
Figure 33 Canny Edge Detection Using Different Threshold Case 1

The following figures illustrate the results on different configurations of the plate and bowl using a threshold of 75-225. The results show the model accurately detects the edges of those objects while minimizing the noise. This approach works for different orientations of the plate and bowl. Combining the depth estimation proposed in 0 with the edge detection would allow the robot's hand to have a grasping point for both objects. For example, in Figure 35, the plate is placed vertically. Canny Detection was able to identify its edge and using the depth estimation, the robot knows which point on that edge is the highest.

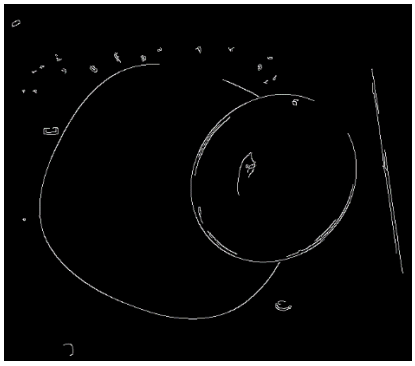
a) Input Image b) Output - Threshold (75-225)

Figure 34 Edge Detection Case 2

a) Input Image b) Output - Threshold (75-225)

Figure 35 Edge Detection Case 3

a) Input Image b) Output - Threshold (75-225)

Figure 36 Edge Detection Case 4

6.3.2.1 Limitations

A limitation for the object grasping section is that this approach would not be able to recognize the edges of a bowl if it is placed upside down. Also, an object might be the highest one, but its grasping spot is blocked. For example, the edge of a mug can be registered as the highest object. Afterward, that object is the object of interest. However, if its handlebar is blocked by another dish, the robot hand won't know what to do. To avoid this situation, we can implement an extra step. If the grasping spot cannot be spotted, the robot hand tries to pick up the second registered highest object.

6.4 6D Posing

Getting the 6D pose of the object which the robot wants to pick is desired in our process. The purpose of this step is to match the orientation of the robot's hand to the object. Gen6D was used to perform pose estimation of each object (Yuan Liu, 2022). It is a model-free 6-DoF object pose estimator. This model only requires posed images of the unseen object. We perform pose estimation for two situations. In the first situation, we used a cup from GenMop (Yuan, 2022). The cup had the point cloud of the object and had a predefined X, Y, and Z plane orientation. Using this object, we trained the model and were able to achieve high accuracy. Figure 37 illustrates the steps to achieve a final pose estimation. First, it detects the mug, then it creates an initial pose estimation. Finally, the pose estimation gets refined to produce. illustrates the final predicted output (blue bounding box) with the ground truth (green box).

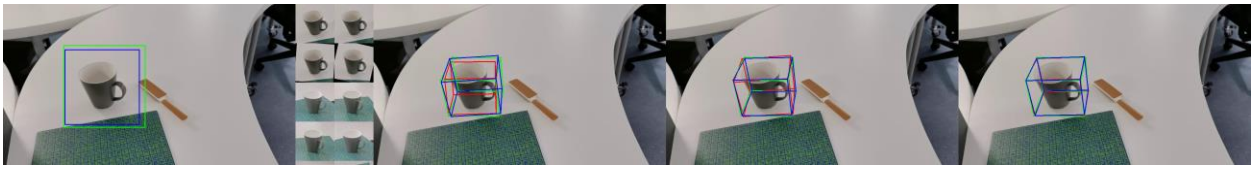


Figure 37 Pose Estimation Steps GenMop

We also applied pose estimation on a custom object. That custom object was our mug. Before applying the Gen6D model, we had to create the coordinate system for our object as the ground truth. First, a 3D point cloud of the object was created using Colmap shown in Figure 39. Then, CloudCompare was used to visualize and process the reconstructed point cloud from COLMAP shown in Figure 40. The positive x and z direction was done manually for the object. Then, the model was trained to recognize the pose of our custom object. The pose estimation on the custom object was alright. However, it was not perfect shown in Figure 41.

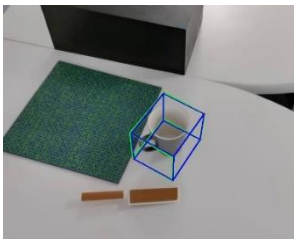


Figure 38 Pose Estimation Output GenMop

Figure 39 3D Point Cloud of Custom Object

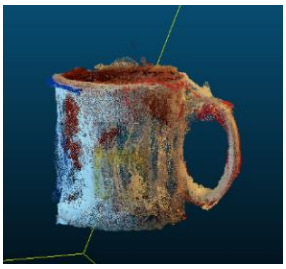
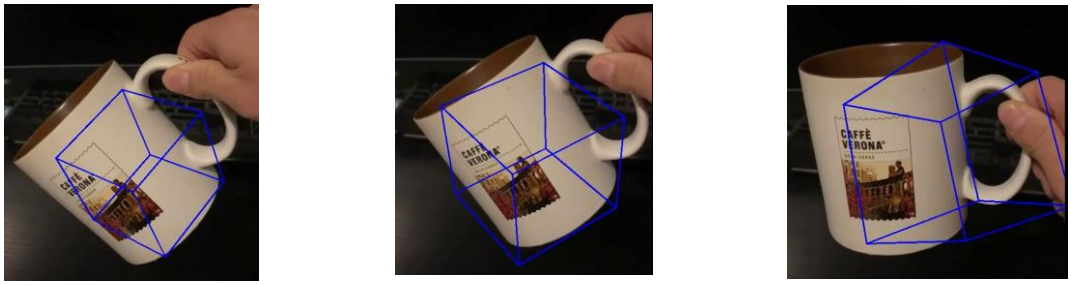


Figure 40 CloudCompare Assign Axis



a) Pose Estimation 1 b) Pose Estimation 2 c) Pose Estimation 3

Figure 41 Pose Estimation on Custom Object

6.4.1 Limitation

One limitation mentioned in Gen6D documentation is that Gen6D requires to be trained on a large amount of data with known ground-truth poses to achieve high performance (Yuan Liu, 2022). However, it was difficult to obtain a high-precision 3D point cloud with accurate known poses. First, to obtain precise 3D Point Cloud using ColMap, the scene must contain enough texture. The objects were placed on a background with different textures, but some points of the background were recognized as the object (COLMAP Tutorial, n.d.). Also, since we are manually assigning the orientation of our custom object, it is not the most accurate method. Hence, the difference in accuracy between the object from GenMop and our custom object.

6.5 Pixel to real-world coordinate mapping

6.5.1 Tests and Results

The objective of this algorithm is for the robotic arm to locate the target object after it detects it. As the robotic arm will only have gathered this information from the image it has captured, the data gathered from the image should be mapped to the data in the real world. Thus, the coordinates from the image in pixels should be transformed into coordinates in the physical world.

Our tests were conducted by capturing multiple images using an iPhone 12. First, the picture with only the blobs was taken and the code was run to detect the reference points which was detected successfully. After that, without changing the location of the blobs the target object and the Aruco marker were placed on the same white background. The code was able to detect the object successfully as well as the marker.

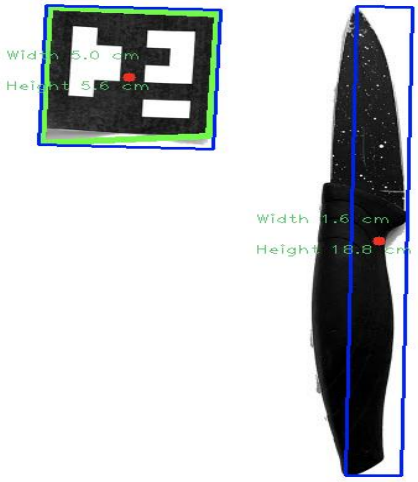


Figure 42 Aruco marker and target object detection

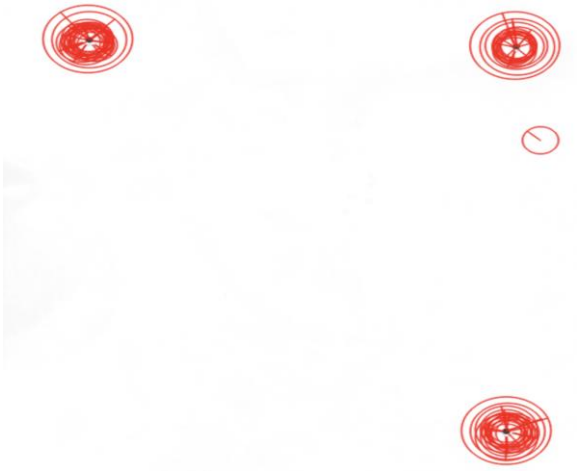


Figure 43 BLOB detection

The x and y distances from the closest reference point were found to be 12.7 and 15.7 respectively. The results are reasonable but not accurate. The actual distances were measured using a ruler and are shown in the figures below. Both the x and y distance from P1 were found to be 12 cm. This gives an error of approximately 25%. The error is quite large specially when it comes to small object. An error this large might cause the robotic arm to miss the object completely.

Figure 44 Actual x distance from P1

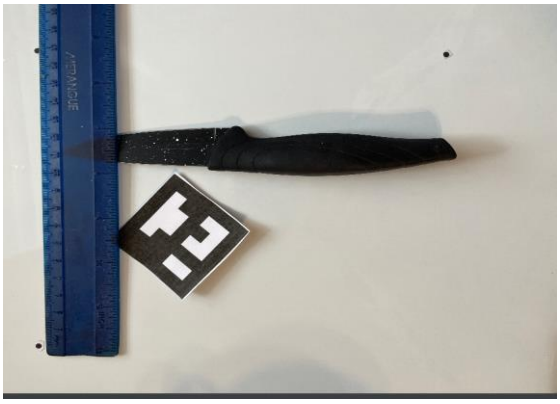


Figure 45 Actual y distance from P1

6.5.2 Limitations and Assumptions

The large range of errors could be caused by many sources. The first source of error is the inconsistent height from which the pictures were taken. As the blob picture and the object and marker pictures are taken in two separate instances, the inconsistent height would cause the size of detections to be different in each picture resulting in a significant error. Another source of error could be due to the inaccurate method used to measure the actual distances. As the tools at our disposal were limited, the standard ruler might not have been the most accurate tool to use. The ruler might have not been straight enough causing the actual distance to be incorrect. Even though the algorithm resulted in a significant error, it did give us an idea of how to perfect it in the future.

For the algorithm to work, it is assumed that the camera is held at a constant distance from the object. Because the distance from the object in the z-axis is held constant, the code is not calculating that distance. The second assumption made in the simulation is that the input object will be one object at a time, thus the distance can only be calculated for an isolated object and not multiple objects in the field of view of the camera. The last and final assumption made is that the object will be placed in the center of a platform with three reference points around the object.

6.6 Inverse Kinematics

6.6.1 Artificial Neural Networks

Using the same methodology as (Duka, 2014), an artificial neural network was setup using Keras in Python (Google Colab). In a similar manner, the network was tested using the movement of the arm (with a total length of unity), along with some constraints such as the training data not having configurations that cannot be achieved by the arm (joint constraints). After obtaining the joint parameters, the final end-effector coordinates were obtained using forward kinematics. The results are shown below.

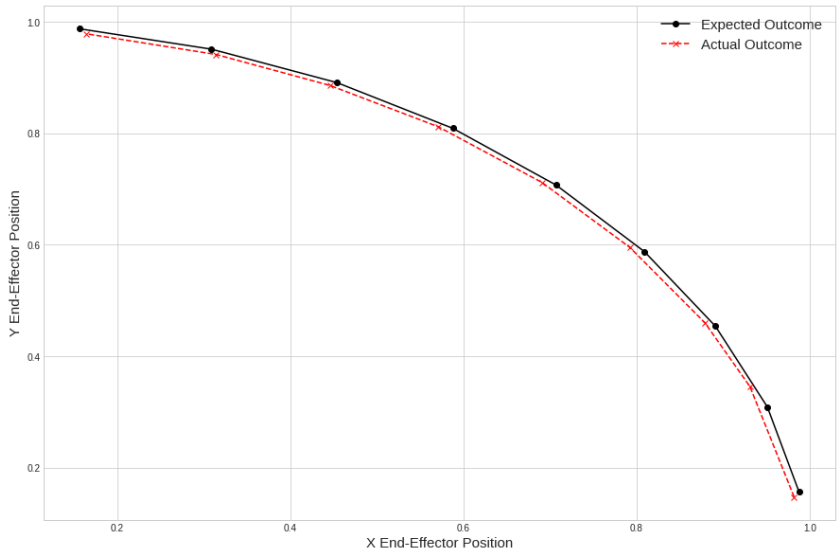


Figure 46 Expected (Input) and Actual End Effector Coordinate Case 1

As expected, the actual outcome is close to the end-effector coordinates, with a maximum error percentage of three percent, in terms of total length. However, these are coordinates near the edge of the arm's reach, and do not require different joint parameter configurations (i.e., the difference between a one-link and three-link arm is negligible). Below is a plot showing the neural networks output to points within the area bound by the arm's length.

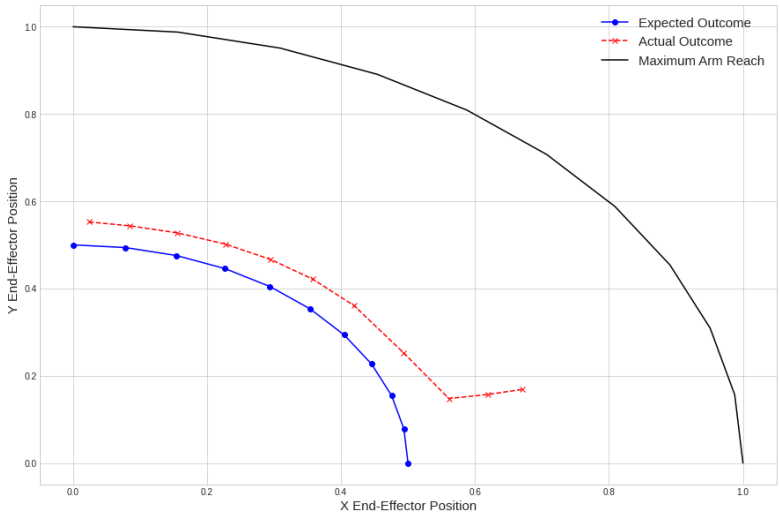


Figure 47 Expected and Actual Coordinates Case 2

In this case, the minimum error is around four percent, while the maximum error reaches around seventeen percent. It also appears that the model starts to give extremely inaccurate results in the last three cases (from left to right). At first glance, this may appear to be due to insufficient training data. However, adding more joint configurations and parameters related to these coordinates (and configurations) still showed significant errors.

To explore this inaccuracy in more detail, we proposed to train another deep learning model using the same network as before, but with a rectified linear activation function (ReLU) instead of the hyperbolic tangent function. The result is shown below.

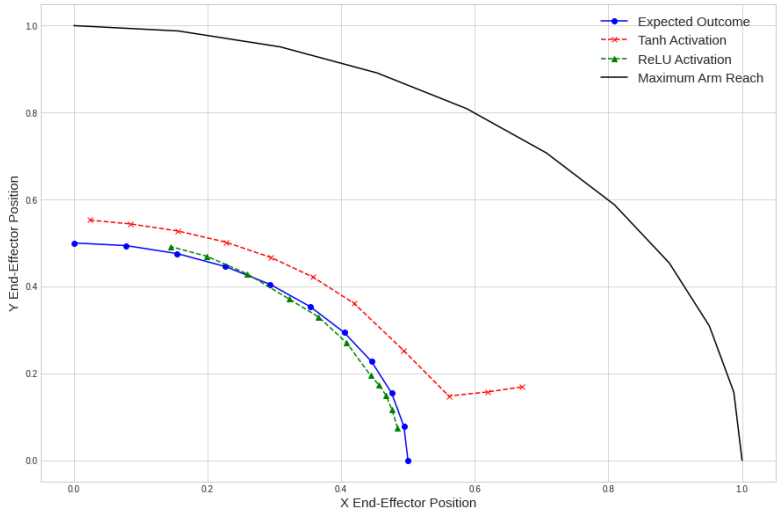


Figure 48 ReLU Output for Case 2

It appears that using the ReLU activation function helps the model to avoid inaccurate behavior. Additionally, using the ReLU activation function instead of the hyperbolic tangent function

significantly reduces computational load (Christianversloot, 2022). However, the ReLU model is considerably more inaccurate in terms of average error (percentage error with respect to length). We further evaluated the two models at random end-effector coordinates, along with a third model (which is a deeper neural network featuring both ReLU and the hyperbolic tangent functions as neural activation methods). The results are shown below.

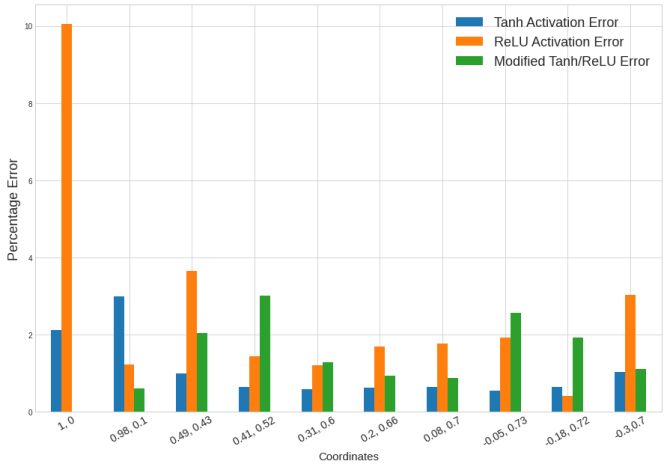


Figure 49 Percentage Error for Random End-Effector Coordinates

These models do have their drawbacks (Christianversloot, 2022), but as we move towards our numerical approach for solving the inverse kinematics problem, we will use the modified model as it does not have substantial inaccuracies in edge cases. This will be important as we try to solve the convergence issues mentioned in section 4.

6.6.2 Numerical Optimization

Unlike the neural network approach, the numerical approach for solving this problem is more straightforward, as solutions are highly accurate. Comparisons between numerical approaches are usually done on how they deal with multiple minima issues (i.e., failure to converge), the number of time/steps it takes to reach a solution, or how they deal with singularities (arm becoming more rigid to find a solution).

As this project has a robotic arm with a relatively smaller number of degrees of freedom, our main aim is to find a solution to the problem, if it is within our arm/joint constraints. Thus, we will mainly focus on the convergence problem.

Using MATLAB and Peter Corke's Robotic Toolbox, a 3-link robotic arm with lengths (1, 0.5, 0.5) and a revolute joint at its base was setup. This can be seen in the image below.

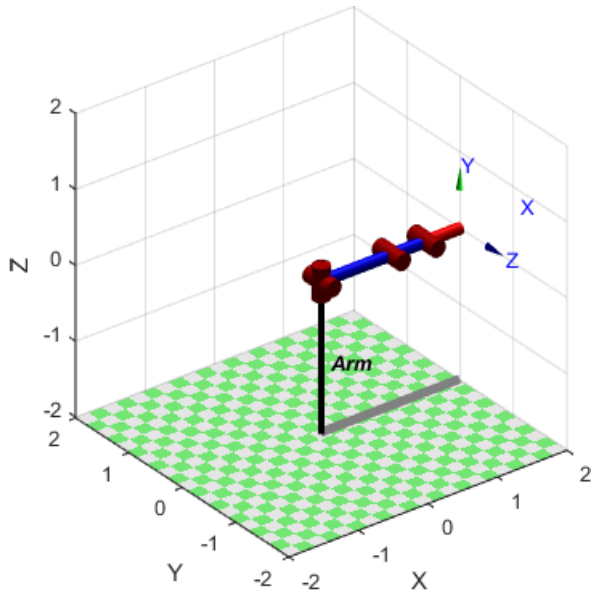
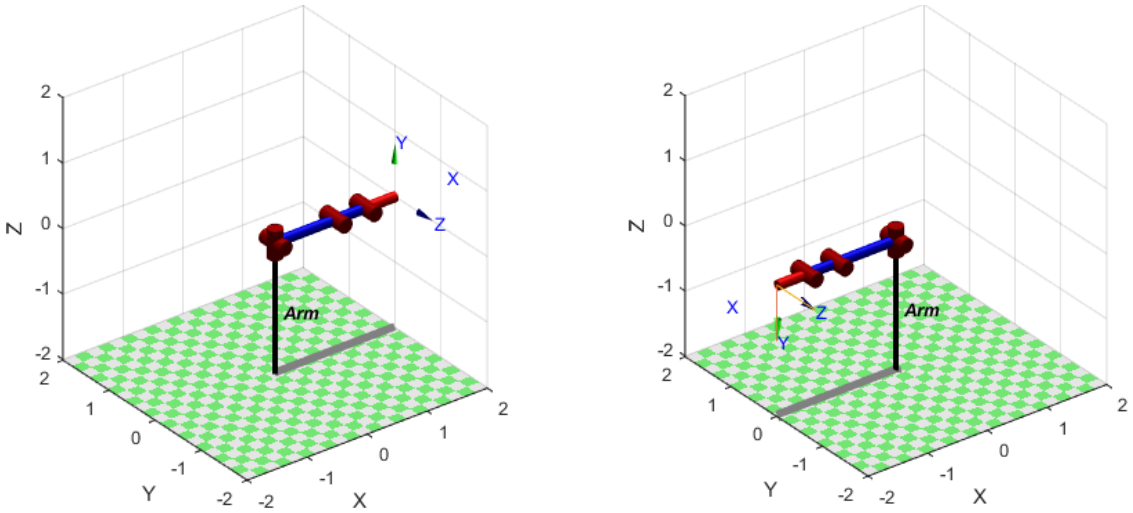


Figure 50 Initial Resting Position for Arm

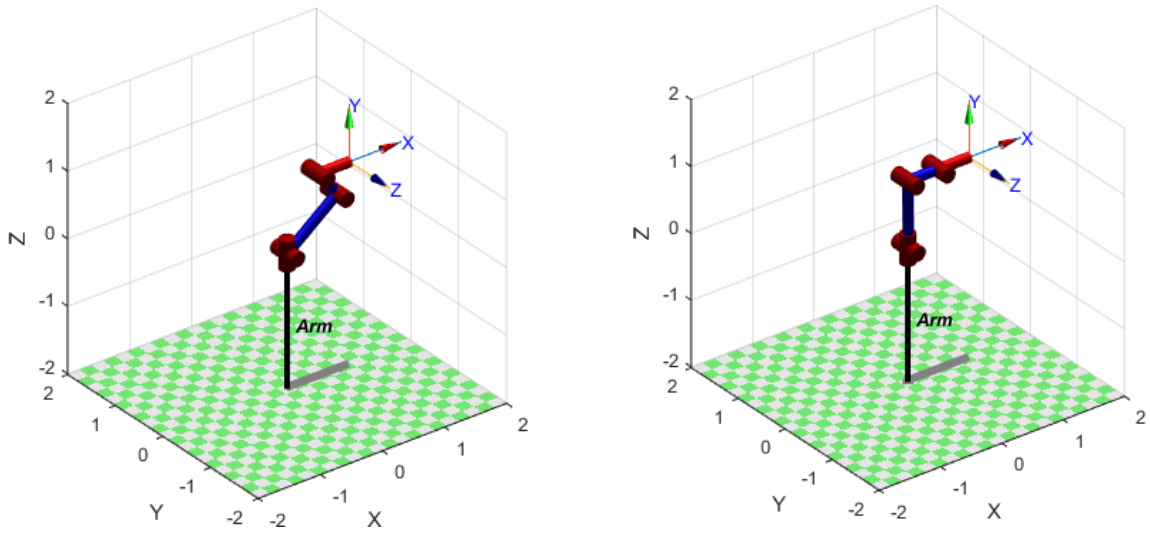
The algorithm primarily used was the Levenberg-Marquardt algorithm. As the algorithm requires an initial guess parameter, we compared the solution to an arbitrary point $A\ (1, 0, 1)$ using two guesses, the robot's initial position I.e. $(\theta_1, \theta_2, \theta_3) = (0, 0, 0)$ and the other extreme position i.e. $(\theta_1, \theta_2, \theta_3) = (180, 0, 0)$



a) Initial Position 1 b) Initial Position 2

Figure 51 Initial Robot Positions for Two Cases

The final solutions, respectively, are shown below.



c) Final Position 1 d) Final Position 2

Figure 52 Final Positions for Two Cases

This shows that the guessing parameter can also be used to obtain a pseudo-optimal path for the arm to move. Additionally, if a solution was needed where the final link had to obtain a specific orientation, this approach can also be used to obtain a certain end-effector orientation. As an example, take the previous arbitrary point but add constraint that the final link must be facing upwards. The solution obtained is thus:

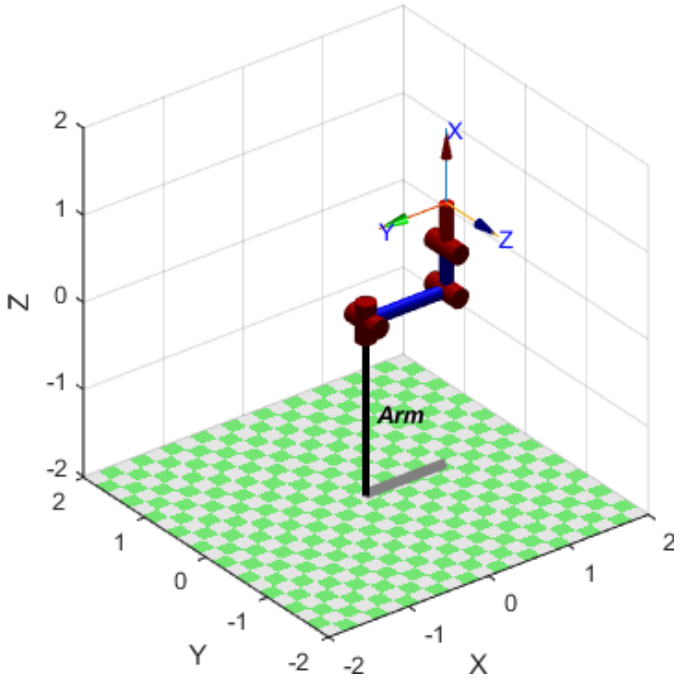


Figure 53 Orientation-Dependent Final Position

However, one major flaw with this entire approach lies in situations where the solution cannot be found due to the presence of local minimums. Usually, algorithms enforce brute force techniques such as temporarily increasing step size to avoid this; however, these measures might not be enough. In these cases, no solution is obtained unless a different starting point is passed that allows the algorithm to converge to the absolute minimum instead of a local minimum.

To account for this major drawback, we recommend using both a numerical approach, combined with the neural network approach. The neural network solution gives values that are mostly close to the true solution, but not exact, and can allow the algorithm to find the true value. Hypothetically, there may exist a local minimum close to the neural network's approximate joint values, however, there hasn't been an observed test case in our simulations yet. To be certain, it is better to test for all values within the arm's enclosed region.

To conclude, our final approach for solving the inverse kinematics of gripping an object includes finding an optimum solution using the numerical approach. If a solution is obtained, we move towards grasping the object. If a solution cannot be reached, obtain an approximate value from the neural network approach, and use it as the best guess parameter. If a solution is still not obtained, use the neural network value, and obtain analytical solutions by enforcing appropriate constraints or brute-force (Giresh K. Singh, 2010). This can be summed up using the flowchart below.

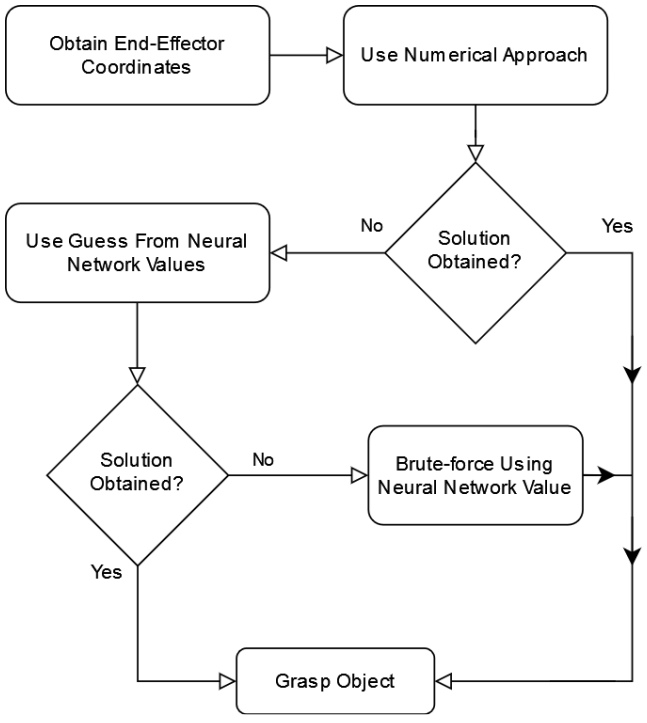


Figure 54 Flowchart for Inverse Kinematic Approach Hierarchy

6.6.3 Limitations and Assumptions

Solving for the joint parameters of the arm via this method leads to a few limitations. Firstly, the location of the grasping point should be within the arm's reach, which includes the length of the arm, as well as the constraints present on the setup, or the joint parameters.

Secondly, it is assumed that the length of the arm is large enough to allow the final link (gripper) to easily orient itself in the desired way for any grasping point. For instance, if the arm must fully extend to reach a point, it is limited to only one specific orientation, thus complicating the problem. This is also a design criterion which must be met by the arm.

Finally, this method of incorporating numerical approaches combined with neural networks to solve this problem does not consider collision detection. While some obstacles can be avoided by imposing additional constraints in the joint parameter values (e.g., first link cannot move more than 180 degrees to avoid hitting the table it is placed on), adding objects on top of the table or the sink would cause difficulties for the arm to function, or damage the object, as well as the arm.

6.7 Placement Algorithm

6.7.1 Tests and results

As mentioned prior, since a brute force approach was used for this task and not an AI model, not a lot of emphasis was put on simulations for this step. To decrease costs and limit computational and hardware complexity, this component was dealt with in a brute force manner. This brute force approach will require some trial and error, when hardware is available, to successfully store the coordinates for each placement destination. As well as calculate the area that can comfortably take an object of each class. This approach however will limit the hardware expenses and computational complexity of our solution causing for the process to be faster and more efficient.

Due to the unavailability of hardware and simulations that would capture the specifics of our setup, no tests or simulations were performed for this component.

6.7.2 Limitations and Assumptions

As this component is not dependent on vision, the limitations of the brute force algorithm lie in the fact that if unexpected errors were to occur in the environment, such as a plate slipping to occupy the slot for two plates instead of one, there will be no way to correct it. The algorithm will not halt as work arounds can be thought of ahead, but it will decrease the efficiency of the algorithm. Another limitation is that the dishwasher space will not be optimally occupied since the area is divided based on an average size for each object and not that actual objects at our disposal. Thus, the actual objects might be significantly smaller than the space designated for it resulting in a lot of wasted space.

Three assumptions were made in this section. First it was assumed that the objects are placed in a standard dishwasher where the space available for each class is predetermined and fixed. Furthermore, that this space is unchangeable in the future. Second assumption is, the objects belonging to each class do not vary drastically in dimensions as the slot for each of the classes will be predetermined and hard coded based on tests conducted in the future. It was also assumed that the object detected in the tray will always fit inside the dishwasher.

7 Conclusion

In conclusion, we proposed a method to have an autonomous robotic dishwasher. The proposed robot is a three-link 5DOF arm with two fingers. This arm would automatically grasp common kitchen dishes from a sink and place them in a dishwasher located beside the sink. The main advantage of our proposed method is that it only requires a monocular camera for the detection, localization, and pose estimation of objects. To our knowledge, available products often combine different hardware to perform those tasks such as lidar, stereo-camera, and RGB-D camera. Our approach aims at reducing the cost of the overall robot and reducing the dependencies among hardware components.

For our proposed solution, we used different resources that were available in different literature and combined them. The main steps used are like robots proven to work for grasping complex objects. Our main steps consist of object localization, object pose estimation, grasp estimation, and motion planning of the robotic arm. The proposed method uses YOLOv7 to detect and classify the objects in the sink. A different approach is used to pick up each object. Then, a deep learning model named MiDaS is used to detect the highest object. After locating the highest object in the pile of dishes, the camera needs to find a grasping spot on the desired object. Depending on the object, two different methods are used. The first one uses YOLOv7 to locate pre-determined spots on certain dishes. The second approach locates the edges of the objects. The depth estimation and the edge location are used together to determine a picking spot for the plate and bowl. After finding a grasping point, the robot hands orient itself in the same orientation as the object. To find the orientation of the object a pose estimator called Gen6D is used. Then, the common force-closure approach is used for the grasping method. The real-world length of the object is measured along the common force-closure technique to have a better idea of the required distance between the fingers of the robotic hand. Finally, to determine the motion planning of the robot arm, a numerical approach combined with a failsafe artificial neural network is used to determine its path.

In the report, multiple tests were showcased for different placement of dishes in different orientations. However, there are multiple possibilities. The next step is to try different case scenarios for our proposed approach. Also, combining all the steps in one simulation to create a more realistic simulation is necessary.

8 References

(n.d.). Retrieved from COLMAP Tutorial: https://colmap.github.io/tutorial.html

Akhilesh Kumar Mishra, O. M.-P. (2015). Robot arm manipulation using depth-sensing cameras and inverse kinematics.

Andras, I. M. (2020). Artificial intelligence and robotics: a combination that is changing the operating room.

Ashutosh Saxena, L. W. (2010). A Vision-based System for Grasping Novel.

BCN3D MOVEO: A fully Open Source 3D printed robot arm. (2016). Retrieved from BCN3D: https://www.bcn3d.com/bcn3d-moveo-the-future-of-learning-robotic-arm

C. -H. Chen, H. -P.-Y. (2011). Stereo-based 3D localization for grasping known objects with a robotic arm system,.

Christianversloot. (2022). *machine-learning-articles/relu-sigmoid-and-tanh-todays-most-used-activation-functions*. Retrieved from https://github.com/christianversloot/machine-learning-articles/blob/main/relu-sigmoid-and-tanh-todays-most-used-activation-functions.md

DishFish. (2019). Retrieved from Survey Finds Americans Would Rather Clean Their Toilet Than Wash Dishes By Hand: https://www.prnewswire.com/news-releases/survey-finds-americans-would-rather-clean-their-toilet-than-wash-dishes-by-hand-300912714.html

Duka, A.-V. (2014). *Neural network based inverse kinematics solution for trajectory*.

Francesco Aggogeri, N. P. (2022). *Inverse kinematic solver based on machine*.

Francisco Gomez-Donoso, S. O.-E. (2019). *Accurate and efficient 3D hand pose regression for robot hand teleoperation using a monocular RGB camera*.

Giresh K. Singh, J. C. (2010). *An Analytical Solution for the Inverse Kinematics of a Redundant*.

Guoguang Du, K. W. (2020). *Vision-based robotic grasping from object localization,*.

Hiroki Suzuki, J. T. (2021). *Underactuated robotic hand for a fully automatic dishwasher based on grasp stability analysis*.

Hui, J. (2018). *Object detection: speed and accuracy comparison (Faster R-CNN, R-FCN, SSD, FPN, RetinaNet and YOLOv3)*. Retrieved from Medium: https://jonathan-hui.medium.com/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359

Jiaming SUn, Z. W. (2022). *OnePose: One-Shot Object Pose Estimation without CAD Models*.

Jiaoyang Lu, T. Z. (2022). *A Neural Network Based Approach to Inverse Kinematics Problem for General Six-Axis Robots*.

Jun Kinugawa, H. S. (2022). *Underactuated robotic hand for a fully automatic dishwasher based on grasp stability analysis*.

Jun Kinugawa, Hiroki Suzuki,Junya Terayama,Kazuhiro Kosuge. (2021). Underactuated robotic hand for a fully automatic dishwasher based on grasp stability analysis.

Kaimeng Wang, T. T. (2022). *Robot programming by demonstration with a monocular RGB camera*.

Li Tian, N. M. (2019). *Object Grasping of Humanoid Robot Based*.

Martin Sundermeyer, M. D.-C. (2020). Multi-path Learning for Object Pose Estimation Across Domains.

MathWorks. (2022). Retrieved from Inverse Kinematics Algorithms: https://ww2.mathworks.cn/help/robotics/ug/inverse-kinematics-algorithms.html

Muhammad Zubair Irshad, S. Z. (2022). *ShAPO: Implicit Representations for*.

OpenCV. (n.d.). *Canny Edge Detector*. Retrieved from https://docs.opencv.org/3.4/da/d5c/tutorial_canny_detector.html#:~:text=Canny%20does%20use%20two%20thresholds,threshold%2C%20then%20it%20is%20rejected.

Poonam Dhankhar, N. S. (2013). *A Review and Research of Edge Detection*.

Rene Ranftl, K. L. (2020). Towards Robust Monocular Depth Estimation:.

Robotics, C. (2020). Retrieved from https://connected-robotics.com/en/

Section II. MODELING. (2020). Retrieved from Inverse Kinematics: https://motion.cs.illinois.edu/RoboticSystems/InverseKinematics.html

Shitian Zhang, J. H. (2020). Robotic Grasping Position of Irregular Object Based Yolo Algorithm.

Sugihara, T. (2009). *Solvability-unconcerned Inverse Kinematics*.

Van Nguyrn, Y. H. (2022). *Templates for 3D Object Pose Estimation Revisited: Generalization to New Objects and Robustness to Occlusions*.

Wang, C.-Y. a.-Y. (2022). *Yolov7*. Retrieved from https://github.com/WongKinYiu/yolov7

Xiaojun Wu, P. L. (2022). A cascaded CNN-based method for monocular vision robotic grasping.

Yann Labbé, J. C. (2020). CosyPose: Consistent Multi-view Multi-object 6D Pose Estimation.

Yisheng He, Y. W. (2022). *FS6D: Few-Shot 6D Pose Estimation of Novel Objects*.

Yuan Liu, Y. W. (2022). Gen6D: Generalizable Model-Free 6-DoF Object.

Yuan, L. (2022). Retrieved from Gen6D: https://connecthkuhk-my.sharepoint.com/personal/yuanly_connect_hku_hk/_layouts/15/onedrive.aspx?id=%2Fpersonal%2Fyuanly%5Fconnect%5Fhku%5Fhk%2FDocuments%2FGen6D&ga=1