Vision-based Manipulation for Grocery Store Applications

GitHub repo: github.com/azzamshaikh/RBE_595_Vision-based_Manipulation_Project

Azzam Shaikh, Chaitanya Gokule, Swati Shirke, Niranjan Kumar Ilampooranan, Alex Brattstrom

***Abstract*—Major corporations like Amazon, Walmart, and Alibaba are investing millions of dollars in integrating robotics into their operations in the e-commerce industry. There is a large need to develop effective solutions to streamline tasks in the environments they operate in, such as grocery stores. Companies like Uber Eats and Instacart are also enabling customers to order groceries remotely, which require grocery stores to react to these requests by picking objects off their shelves and package them for customers. In this project, we develop a vision-based manipulation pipeline for picking objects from shelves for grocery store applications. Our pipeline includes the integration of an object perception stack, two grasping approaches, namely CAP-Grasp and GGCNN, closed-loop controls, and an overarching state machine to control the overall system. We compare our two grasping approaches by conducting a series of picking trials with varying number of objects. We evaluate the success of these approaches to determine the better model for the objective of shelf picking. The results of our experiments show that CAPGrasp generates grasps for objects on a shelf with 80% success rates, indicating that is the better approach for our application.**

I. Introduction

Robotics is transforming industries with advancements aimed at enhancing efficiency, reducing labour costs, and delivering better customer experiences. In the grocery sector, where automation is becoming essential, major players like Amazon, Walmart, and Alibaba have pioneered efforts to integrate robotics [1]. Amazon's "Amazon Go" exemplifies cashierless stores powered by artificial intelligence [1], enabling seamless shopping experiences. Walmart's use of shelf-scanning robots streamlines inventory management [1], while Alibaba's investments in warehouse automation and delivery robots are redefining logistics and fulfilment processes [1].

These advancements are not just technological leaps but are driven by necessity. The explosive growth of e-commerce and evolving consumer expectations have created a demand for systems capable of meeting these challenges efficiently. Robotic solutions are pivotal in bridging the growing supply-demand gap while fostering better customer engagement through timely and accurate fulfilment [1].

The focus of our project is the development of a vision-based pipeline to pick objects from a shelf and drop them in a bin. In specific, two generative grasp sampling algorithms, CAPGrasp and GGCNN, will be implemented and evaluated to determine the better approach for picking objects from a shelf. All aspects from object perception, grasp sampling, controls, and task planning are considered.

The rest of this paper is structured as follows: Sec. II: provides a literature review. Sec. III: describes our implementation. Sec. IV: presents our experiments and evaluation results, and Sec. V: discusses our work of limitations, challenges, and future work.

II. Related Work

This section will review relevant literature published in the domain of grocery store, shelf picking applications.

Robotic manipulation in cluttered or confined environments has been extensively studied, with various approaches addressing challenges in grasp synthesis, object identification, and manipulation planning. Grotz et al. proposed a system for picking unseen objects from densely packed shelves using synthetic data-driven segmentation and heuristic grasp planning [2]. While effective in structured settings, the reliance on synthetic data limits its generalizability to real-world environments. Complementing this, Murali et al. introduced a 6-DoF grasping framework leveraging a Variational Autoencoder and collision-checking modules to improve grasp success rates in occluded environments. However, their method does not account for full arm trajectory collision checking, which can lead to motion planning failures in cluttered spaces [3].

Costanzo et al. addressed manipulation planning for shelf replenishment by integrating tactile feedback and reactive control, achieving success in confined spaces [4]. Meanwhile, Kang et al. proposed Monte Carlo Tree Search (MCTS) with decision networks for planning grasps in occluded and constrained spaces. Their method introduced flexibility in rearranging objects to reach the target but struggled with handling object placement strategies in confined environments [5]. Garcia Ricardez et al. developed an autonomous service robot for retail, incorporating compliance mechanisms and custom shelf modifications, though scalability to unstructured retail environments remains an open challenge [6].

Cavallo et al. demonstrated a high success rate in robotic shelf refilling using a combination of visual and tactile sensing but faced limitations due to reliance on specific learned models that reduce adaptability [7]. Bajracharya et al. focused on mobile manipulation in semi-structured environments, using field testing to validate reliability in grocery shopping tasks. Their work highlighted the importance of hardware-software co-design but acknowledged relatively low success rates compared to human performance [8].

Morrison et al. introduced a real-time grasp synthesis method, GGCNN [9], which is object-independent and supports closed-loop control at a high frequency [10]. Unlike methods that rely on computationally expensive architectures, such as Fang et al.'s GraspNet-1Billion, which predicts 6-DoF grasp poses for large-scale cluttered scenes but requires significant resources [11], or Kang et al.'s MCTS-based grasp planning, which focuses on rearrangement in occluded environments [5], Morrison's GG-CNN emphasizes speed and simplicity. However, it predicts grasp rectangles from depth images rather than directly computing full 6-DoF poses, which

can be a limitation when compared to techniques like Fang et al.'s end-to-end decoupled grasp prediction [11]. More recent work from Weng et al. introduces a 6-DoF, continuous approach-constrained generative grasp sampler, referred to as CAPGrasp [12]. This grasp sampler introduces a novel learning strategy which eliminates the need to develop large datasets and improves grasp poses while respecting grasp constraints. For applications where approach constraints are required, such as shelves, this solution provides a more robust grasp sampling approach when compared to GGCNN.

Efforts to improve efficiency in dense environments include Wang et al.'s monotone and non-monotone rearrangement planning approach, which uses precomputed constraints for pruning invalid plans. While efficient, their exclusion of non-prehensile actions limits adaptability for a broader range of scenarios [13]. Lastly, Klingbeil et al. proposed a grasp selection algorithm for autonomous checkout robots, enabling versatile handling of unknown objects. However, the single-view depth data used for grasp planning introduces uncertainty in occluded regions, reducing reliability in cluttered environments [14].

III. Implementation

The implementation of our pipeline is described below based on the four following subsystems:

1) Object Perception
2) Grasp Sampling
3) Controls
4) State Machine

Each of these four subsystems will be discussed in detail. To begin, our simulation environment will be presented.

A. Simulation Environment

For our simulation environment, Gazebo Classic 11 was utilized with ROS2 Humble, on an Ubuntu 22.04 OS. Two simulations were developed, one with a table and one with a shelf, as shown in Figure 1 and 2, respectively.

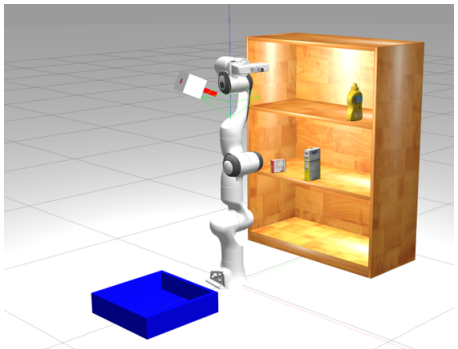

Fig. 1: Shelf environment.

In these simulations, objects from the YCB dataset are utilized for grasping [15]. The models themselves were utilized from a GitHub repo provided by Central Lab Facilities [16].

In the simulations, there is a depth camera that obtains image feeds, depth maps, and point cloud data, allowing us to visualize our scene.

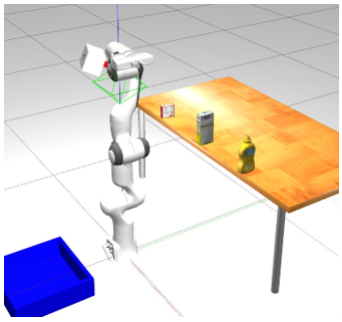

Fig. 2: Table environment.

B. Object Perception

With our simulation environment defined and our camera integrated, objects needed to be perceived within the environment. To achieve this, state-of-the-art computer vision techniques were developed and deployed.

In order to detect objects in the scene, an object detection algorithm was developed. The algorithm was based on a You Only Look Once (YOLO) 11 Model [17]. YOLO11 is a state-of-the-art object detection algorithm that is capable of classifying objects within any scene and providing bounding boxes around their location. In order to achieve an accurate prediction, the model had to be fine-tuned on the YCB objects. A dataset developed by ycbfoods on Roboflow was utilized to fine-tune the model [18]. The dataset consists of 1000 images of rendered, synthetic visuals of YCB object with random occulsions, rotations, lighting conditions, etc. After training the model for 60 epochs, the results of the training can be seen in Figure 3.

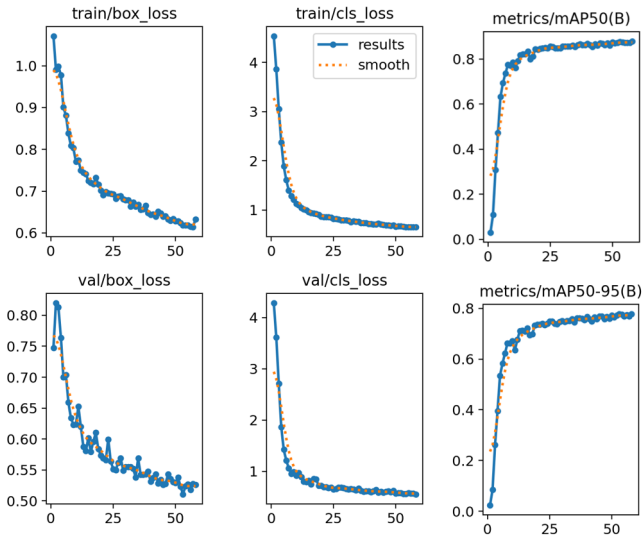

Fig. 3: Training and validation results of the YOLO11.

The model had a validation mean average precision of 0.88 for intersection over unions over 50%, and a mean average precision of 0.78 for intersection over unions between 50 to 95%. This model was deployed into the simulation and overall, the models is successful in classifying object within the Gazebo simulation. Figure 4 shows a sample visual

of the real-time object detection and classification. Further implementation can be seen in the yolo and yolo_finetune packages in the linked GitHub repository.

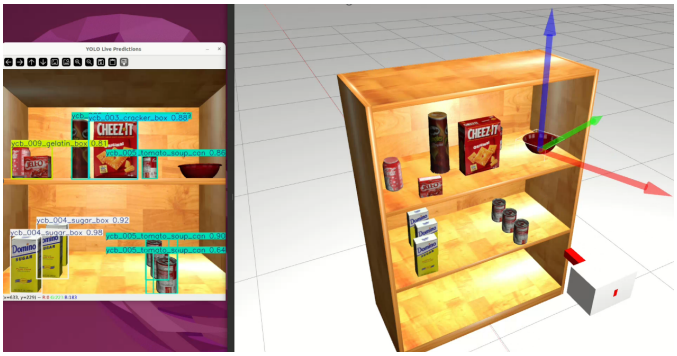

Fig. 4: Sample image of the trained YOLO11 model making predictions in real-time.

In addition to object detection, object segmentation is also required. The segmentation required in the point-cloud space is needed to obtain segmented point-clouds of a desired object. To achieve this, various point-cloud processing techniques are employed from the Point Cloud Library to downsample our point-cloud scene, extract point-cloud data based on bounding box regions, and remove outliers to achieve a segmented model. Further implementation can be seen in the pcl_transform package in the linked GitHub repository.

C. Grasp Sampling

Once objects within our scene can be detected, algorithms can be implemented to synthesize a grasp. To achieve this, two state-of-the-art approaches are considered and implemented for this project: GGCNN and CAPGrasp.

1) GGCNN: Generative Grasping Convolutional Neural Network, or GGCNN, predicts grasp quality for every pixel of an input depth image [9]. It was trained on Cornell Grasping dataset containing RGB-D images on positive and negative labeled grasps. The network weights along with its implementation is made open source via Github [19]. Given an input depth image of size 300x300, the model outputs four different heatmaps - Q_{img}, W_{img}, $\sin\phi$, $\cos\phi$. Subsequently, the Q_{img} heatmap can be queried for the highest value to obtain the best grasp, upon which the width and angle of the grasp can be found. Using this information along with the camera extrinsics and gripper's geometry, 6D pose in the world frame can be obtained accordingly.

A sample grasp rectangle output for a given input image is shown in Fig. 5. With the heatmaps obtained from the network for the given scene, a grasp rectangle is constructed and plotted. Although GGCNN is meant for top-down grasps, for which the end-effector orientation would be strictly vertical, surface normals can be estimated on the grasp point of pointcloud, with which the grasp orientation can be obtained. For this work, the GGCNN ROS2 wrapper as a service-client node was implemented for which it subscribes to the bounding boxes and depth image topics. Once the client calls for the service, the highest grasp quality in the region of the bounding box is provided as the output, for which the 3D grasp pose in base-frame for gripper is further calculated.

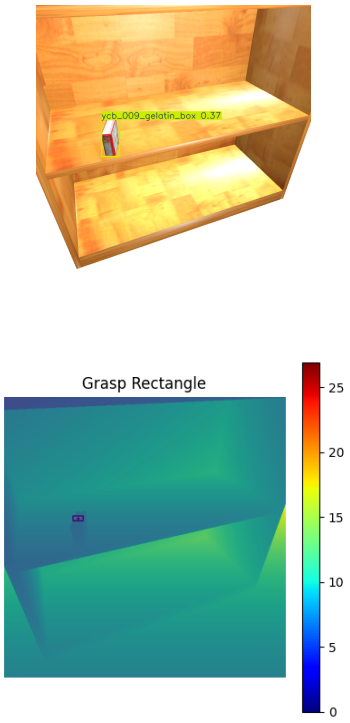

Fig. 5: Camera feed and GGCNN predicted grasp rectangle

2) CAPGrasp: As mentioned in the related work, CAPGrasp is a generative grasp sampler that takes into account an approach constraint. For a shelf picking objective where our approach angle is constrained by shelves and surrounding objects, CAPGrasp provides a unique approach to the grasping problem as we can now define our approach angle when attempting to grasp an object. The CAPGrasp implementation is open source via GitHub [20]. The model takes an object point-cloud as an input to the model. The point-cloud is then passed to the Validator network to compute the 6-DoF grasp point for the given input model. The Validator network generates 200 grasps which are then passed to an Evaluator network to refine the grasp found by the Validator based on internal scoring metrics. The final output is a list of 6-DoF grasp poses for the given object, ordered in terms of accuracy, and the best grasp pose is then selected. Figure 6 shows the prediction pipeline and sample grasp for a mustard bottle.

A pretrained model was utilized for our application and the code base was wrapped into a ROS2 service client node. The service server subscribed to live point-cloud data in a thread. When the service server receives a start command from the client, the model will get the point-cloud data at that point in time, pass it to the networks to generate grasp samples, and return a grasp pose.

D. Controls

For the motion planning pipeline, we utilized the Cranfield repository as the foundational framework. This repository

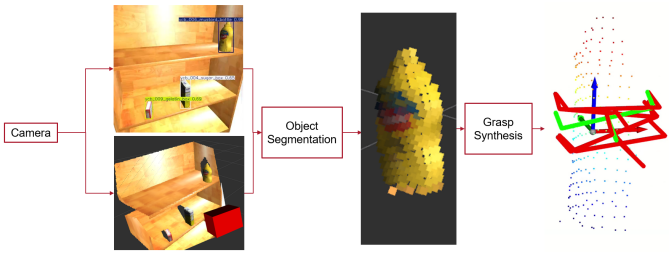

Fig. 6: A visualization of the grasping pipeline required for CAPGrasp.

offers a structured approach for executing basic actions, such as moveXYZW, which moves the robot to a specified XYZ position and orientation. Building upon this, we developed a sequencer logic to orchestrate a series of actions, enabling a closed-loop pick-and-place operation. The sequencing of the pick and place tasks was implemented using a finite state machine which ensured smooth transition between tasks. The controller node was implemented as an action server which executed the commands sent by the sequencer. The controller also simulated object handling with gripper open and close actions, calling the attacher function which enabled the robot to grasp and release objects. A central trigger function was used to parse incoming commands and generate appropriate action calls, which were then executed sequentially.

1) Collision with Table or Shelf: The Panda robot frequently collided with the table or shelf while picking objects. To address this issue, we leveraged the collision avoidance feature of MoveIt. However, for this feature to work, the motion planner must have knowledge of the object's presence to design a collision-free trajectory. To achieve this, we developed a ROS publisher node that continuously publishes the collision object's data to the planner, ensuring safe and efficient operation.

2) Collision with objects on the shelf/table: To address collisions with objects on the shelf or table, we introduced a pre-grasp pose. Instead of directly moving to pick the object, the robot pauses at a designated distance from the object before approaching it. This intermediate step ensures stable grasping without toppling the object.

Additionally, the robot occasionally collided with objects located near the target object. A more robust solution to prevent such collisions would involve generating a voxel representation of the entire scene, derived from 3D point cloud data. This would include not only the target object but also all other nearby objects, providing a complete spatial awareness of the environment. By passing this voxel map to the MoveIt planner, the robot would be able to plan its trajectory with a more comprehensive understanding of potential obstacles, thereby avoiding collisions with surrounding objects. However, this approach was not implemented in the current project.

E. State Machine

Once all the subsystems for object detection, grasp sampling, and controls were implemented, the entire solution could be wrapped into a state machine. A state machine was developed using the python-statemachine library [21].

The following five states are defined for the machine:

1) *home*: The state machine begins in the 'home' state and transitions to the 'yolo' state upon launch.
2) *yolo*: This state decides the main flow of the state machine. As the name implies, this state looks at the information being published from the yolo package. If objects are available in the scene, the state machine will transition to the next state, 'obtain_grasp_pose'. If there are no objects available and the sequence is completed, the transition will go to the final state, 'done'.
3) *obtain_grasp_pose*: Once the state machine enters this state, depending on the grasp configuration, either the GGCNN service or the CAPGrasp service will be called. The services will return a grasp pose for the object classified with the highest accuracy in the scene. One returned, the machine will transition to the 'move' state.
4) *move*: This state manages the execution of the robot controller. The robot will execute a sequence of motions required to complete the shelf pick-and-place task. Once all motions are executed, the machine will transition back to the 'yolo' state.
5) *done*: Once all objects in the scene are picked, the machine transitions to the 'done' state and the system shuts down.

A diagram of the state machine can be seen in Figure 7.

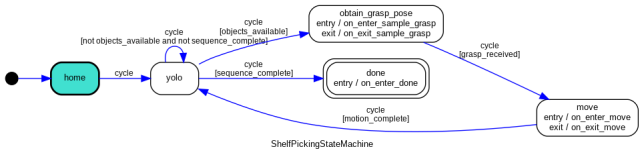

Fig. 7: A diagram of the implemented state machine.

With the state machine developed, the entire pipeline could be deployed for shelf pick and place. A sample visual of a the pipeline in action can be seen in Figure 8.

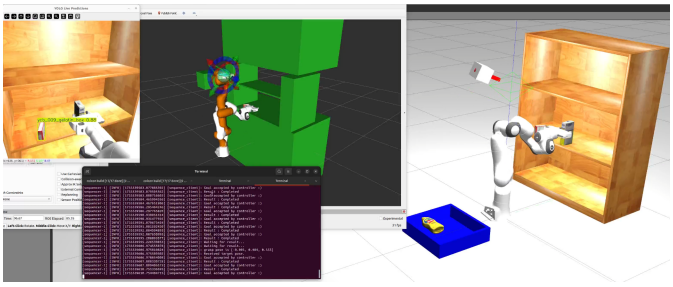

Fig. 8: A sample visual of the vision-based manipulation pipeline in action.

IV. Experiments and Results

The implementation of our pipeline includes two main objectives, the ability to use vision to obtain a grasp, and the ability for the pipeline to execute the motions required to move objects to and from the shelf to the basket. For our implementation, two approaches for grasping were implemented and each of them will be evaluated.

A. Experimental Setup

In order to compare the effectiveness of the models, the following experimental setup was designed. A varying number of objects - in this case, 1 object, 2 objects, and 3 objects, were placed onto a table and shelf environments and the system was tasked to retrieve the object and drop it to the basket. For each case of varying objects, the simulation was ran 5 times. This results in a total of 60 grasp attempts. All objects were placed in the same position for each test. In each case, the success of each individual grasp was determined. Effectively, this looks at the ability of the robot to grasp a single object at any time, not considering the entire sequence. Additionally, the success of the entire sequence, i.e. picking all objects in the scene in succession, was determined. From these results, a success rate can be determined for each approach and the better algorithm for shelf picking can be determined.

B. Results

The following sections include the results for the GGCNN testing and CAPGrasp testing.

1) CAPGrasp Results: The results for the CAPGrasp experiements are included in Table I and Table II for the shelf and table simulation, respectively.

2) GGCNN Results: The results for GGCNN experiments are included in Table III and Table III for the shelf and table simulation, respectively.

3) Combined Experimental Results: Once both CAPGrasp and GGCNN were evaluated, the results from the trials can be compared.

Figure 9 shows the outcome of the grasps for individual objects. This metric looks purely at the ability for pipeline to pick up a single object, at any point time. Based on the results, it can be seen that the CAPGrasp has a success rate of 80% for any object. For GGCNN, the model has a much higher failure rate - the causes for this failures will be presented later on.

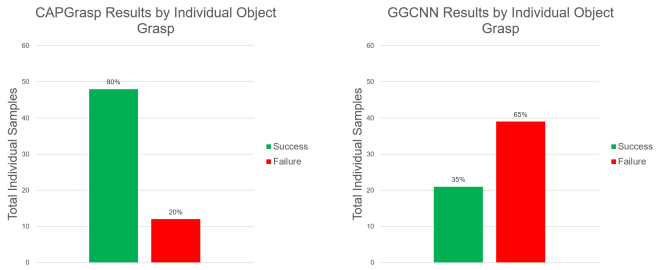

Fig. 9: Results of the individual grasp metric.

Figure 10 shows the outcome of the grasps for an entire sequence. This metric looks at the entire pipelines ability to grasp objects in succession. For example, if there are two objects and the pipeline picks and drops off both items successfully, this is considered a success. However, if there are two objects, and one object gets picked up successfully while the other fails, this would be considered a failure. Based on the results, it can be seen that CAPGrasp has no general trend available in its results. In single object cases, the success is high. However, for the 2 object case, the failure is high. For GGCNN however, there is a clear trend, which is that the model fails consistently.

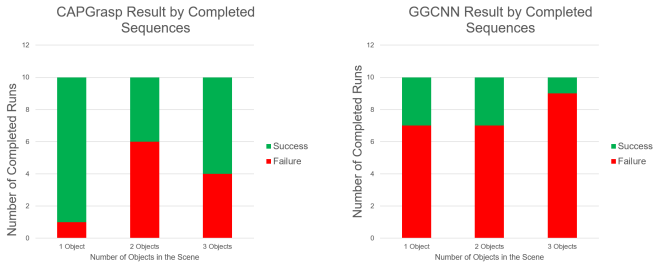

Fig. 10: Result of the completed sequence metric.

These failures were investigated and categorized into a few issues:

1) Collision: This is defined as any instance where the robot arm collided with the environment or other objects.
2) Controls: This is defined as any instance where the robot reached the grasp position but failed to move afterwards. It was unclear what was causing this issue.
3) Bad Grasp: This is defined as any grasp position that is 'unreachable' by the controller. 'Unreachable' is further defined as a position that is within the robot workspace, but is at a position that will create a collision, according to the planner. Thus, resulting in a failure to find a path to that point in space.
4) No Object Detection: This is defined as any instance where YOLO could not detect an object, even though it is present in the scene.

Based on these failure modes, Figure 11 shows the distribution of failure modes for each of the algorithms. Since GGCNN had a higher failure rate as compared to CAPGrasp, there is a larger distribution of failure across the modes. For GGCNN, the main contributing factor toward the failed runs is controls. For CAPGrasp, the main contributing factor towards the failed runs is collisions.

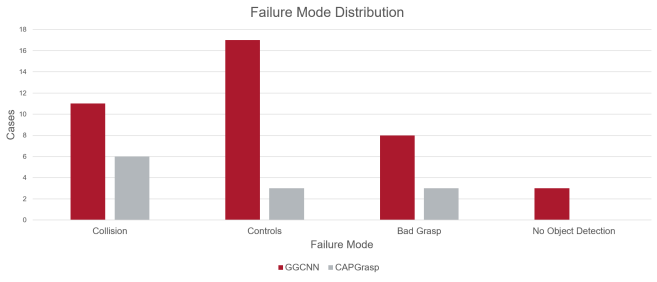

Fig. 11: Distribution of the identified failure modes.

C. Conclusion

Based on the results of these experiments, when comparing the success between individual grasps and success for an entire picking sequence, CAPGrasp, across the board, generates better grasps for the objective of shelf picking.

TABLE I: CAPGrasp experiment results for a shelf configuration.

Shelf Configuration	Number of Objects	Run 1		Run 2		Run 3		Run 4		Run 5	
		Result	Failure Mode	Result	Failure Mode	Result	Failure Mode	Result	Failure Mode	Result	Failure Mode
	1	0%	Collision	100%		100%		100%		100%	
	2	100%		100%		100%		100%		50%	Collision
	3	66%	Collision	66%	Collision	66%	Bad Grasp	100%		100%	

TABLE II: CAPGrasp experiment results for a table configuration.

Table Configuration	Number of Objects	Run 1		Run 2		Run 3		Run 4		Run 5	
		Result	Failure Mode	Result	Failure Mode	Result	Failure Mode	Result	Failure Mode	Result	Failure Mode
	1	100%		100%		100%		100%		100%	
	2	50%	Bad Grasp	0%	Controls	50%	Collision	50%	Bad Grasp	50%	Collision
	3	66%	Collision	100%		100%		100%		100%	

TABLE III: GGCNN experiment results for a shelf configuration.

Shelf Configuration	Number of Objects	Run 1		Run 2		Run 3		Run 4		Run 5	
		Result	Failure Mode	Result	Failure Mode	Result	Failure Mode	Result	Failure Mode	Result	Failure Mode
	1	0%	Bad Grasp	0%	No Object	0%	Controls	0%	No Object	0%	Collision
	2	0%	B.G. / Ctrl.	50%	Controls	0%	B.G. / Ctrl.	100%		100%	
	3	0%	Controls	33%	Collision	0%	Collision	0%	Collision	0%	Controls

TABLE IV: GGCNN experiment results for a table configuration.

Table Configuration	Number of Objects	Run 1		Run 2		Run 3		Run 4		Run 5	
		Result	Failure Mode	Result	Failure Mode	Result	Failure Mode	Result	Failure Mode	Result	Failure Mode
	1	100%		100%		0%	Collision	0%	Bad Grasp	100%	
	2	50%	Bad Grasp	100%		50%	No Object	0%	Ctrl. / B.G.	50%	Controls
	3	66%	Controls	33%	Coll. / B.G.	33%	Ctrl. / B.G.	33%	Controls	100%	

V. Discussion

This section will discuss some of the limitations of our work, the challenges that were encountered, and future work.

A. CAPGrasp Limitations

One of the limitations of CAPGrasp is that the model is extremely dependent on high quality point clouds. In cases where poor point clouds are generated, the model will generate a poor grasp that will lead to failure. For example, in Figure 12, for a point cloud such as this, the model will generate an extremely poor grasp. Thus, for any implementation of CAPGrasp, obtaining a clean, segmented model is imperative for success.

Additionally, for smaller objects that fit within the grippers width, regardless of orientation, the model has a higher chance of generating a bad grasp. For example, in Figure 13, the generated grasp is in an orientation that will collide with the table top surface, which will result in a collision. Thus, utilizing objects that have geometry constraints that remove doubts regarding orientation of the grasp will improve the model results.

B. GGCNN Limitations

Since GGCNN relies on the depth-image input, it is heavily influenced by the camera viewpoint. Example input and grasp rectangles for gelatin box on a shelf is shown in Fig. 14.

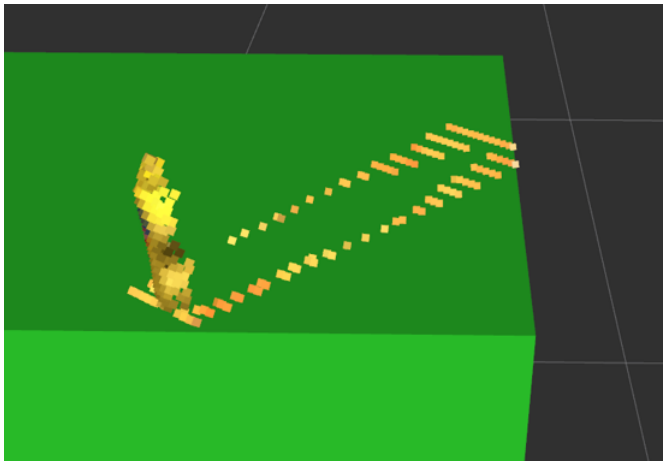

Fig. 12: An example of poor pointcloud segmentation for an object.

For the scenario where the camera is positioned towards the top of the gelatin box, the generated grasps are on the top surface of the box. If this pose is provided to the robot, the motion planner would fail unless there is sufficient space inside the shelf rack to grasp the object from the top. In the second scenario where the camera is positioned in the same level as that of the object, the grasp with the predicted highest quality is on the front surface of the object. These factors determine the overall quality and feasibility of grasps and hence, the

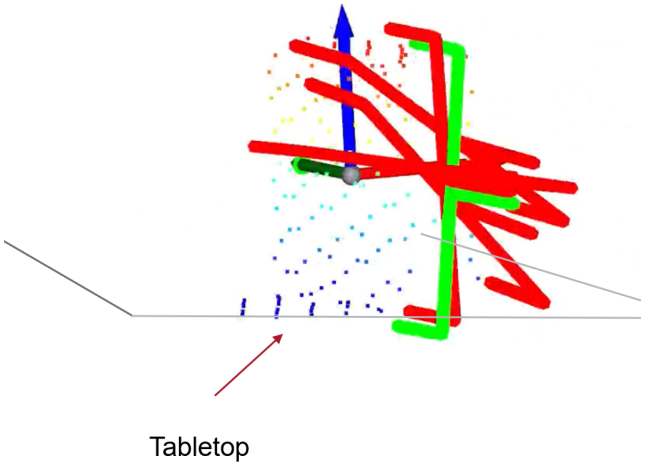

Fig. 13: An example of a bad grasp generated by CAPGrasp for the smaller Gelatin box.

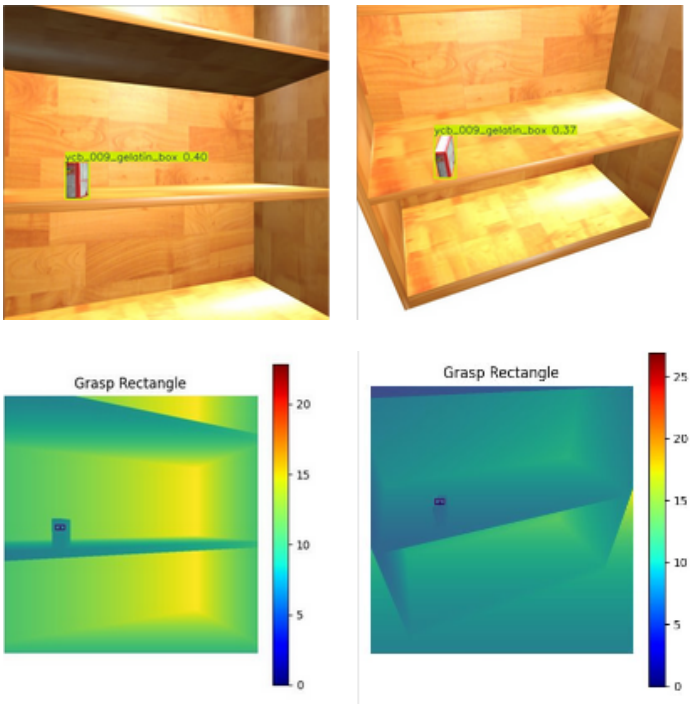

Fig. 14: Scene with different camera poses and GGCNN output rectangles

viewpoints need to be considered. Possible solution include usage of in-hand camera for generating grasp poses on the required object, which can be done so by moving the robot arm in level to the rack where the object is present. Since the shelf rack positions would be known beforehand, this solution is quite feasible in real-world implementations.

C. Challenges and Future Work

During our project, we have encountered several challenges that provide opportunities for improvement in future work. Object detection showed slight inconsistencies, likely from due to limitations in the dataset. To address this, we propose expanding the dataset with additional images of objects from the YCB dataset captured from the camera's point-of-view in Gazebo, ensuring better alignment with the simulation environment. Simulation stability was another issue, as we experienced varying difficulties opening the simulation consistently. This could be mitigated by exploring alternative workspaces or simulations outside of the Cranfield repository for greater reliability.

We also faced challenges with occasional robot collisions and poor grasp decision and execution. A potential improvement involves transitioning the camera setup from eye-to-hand to eye-in-hand or eye-on-hand. This adjustment would allow tools like GGCNN to evaluate grasps more effectively from a head-on perspective relative to the camera's view. Additionally, the dataset's relatively small size limited the statical power of our results. Increasing the number of runs and trials would address this issue, and automating the result collection and verification process would streamline data analysis as the dataset grows.

Lastly, we encountered inconsistencies with the Cranfield's link attachment plugin–IFRA_LinkAttacher, to allow objects linked to URDF to improve pick-and-place–would sometimes link objects to the end effector when the object was not in proximity or within proper orientation limits. Refining the plugin by reducing its activation range and adding alignment checks could improve its reliability. Alternatively, exploring solutions that bypass the plugin, such as enhancing the simulation's physics engine or adopting a different simulation platform, may provide more robust results.

These improvements may offer a path to addressing the identified challenges and enhancing the overall effectiveness of the system.

REFERENCES

[1] Foodverge, "Inside the robot-powered grocery store of the future," https://www.youtube.com/watch?v=I_uiPqFG5J4&t=1s, 2024, accessed: 2024-12-13.

[2] M. G. et al., "Towards robustly picking unseen objects from densely packed shelves," *RSS Workshop on Autonomous Robots for Logistics*, 2023.

[3] A. M. et al., "6-dof grasping for target-driven object manipulation in clutter," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 6232–6238.

[4] M. C. et al., "Manipulation planning and control for shelf replenishment," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1595–1601, 2020.

[5] M. K. et al., "Grasp planning for occluded objects in a confined space with lateral view using monte carlo tree search," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022.

[6] G. A. G. R. et al., "Autonomous service robot for human-aware restock, straightening and disposal tasks in retail automation," *Advanced Robotics*, vol. 36, no. 17–18, pp. 936–950, 2022.

[7] A. C. et al., "Robotic clerks: Autonomous shelf refilling," in *Robotics for Intralogistics in Supermarkets and Retail Stores*, L. V. et al., Ed. Springer, 2022.

[8] M. B. et al., "Demonstrating mobile manipulation in the wild: A metrics-driven approach," *arXiv*, 2023.

[9] D. Morrison, P. Corke, and J. Leitner, "Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach," *arXiv preprint arXiv:1804.05172*, 2018.

[10] D. M. et al., "Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach," *Australian Centre for Robotic Vision*, 2018.

[11] H.-S. F. et al., "Graspnet-1billion: A large-scale benchmark for general object grasping," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.
[12] Z. Weng, H. Lu, J. Lundell, and D. Kragic, "CAPGrasp: An r3xso(2)-equivariant continuous approach-constrained generative grasp sampler," *arXiv preprint arXiv:2310.12113*, 2023.
[13] R. W. et al., "Efficient and high-quality prehensile rearrangement in cluttered and confined spaces," in *2022 International Conference on Robotics and Automation (ICRA)*, 2022.
[14] E. K. et al., "Grasping with application to an autonomous checkout robot," in *2011 IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 2837–2844.
[15] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar, "Benchmarking in manipulation research: Using the yale-cmu-berkeley object and model set," *IEEE Robotics Automation Magazine*, vol. 22, no. 3, pp. 36–52, 2015.
[16] L. G. Leroy Rügemer, Markus Vieth, "GitHub - CentralLabFacilities/gazebo_ycb: SDF for YCB models for usage in gazebo — github.com," https://github.com/CentralLabFacilities/gazebo_ycb, [Accessed 10-12-2024].
[17] G. Jocher and J. Qiu, "Ultralytics yolo11," 2024. [Online]. Available: https://github.com/ultralytics/ultralytics
[18] ycbfoods, "ycb-foods-v2 dataset," https://universe.roboflow.com/ycbfoods/ycb-foods-v2, aug 2023, visited on 2024-12-10. [Online]. Available: https://universe.roboflow.com/ycbfoods/ycb-foods-v2
[19] D. Morrison, P. Corke, and J. Leitner, "Generative grasping cnn (gg-cnn)," 2018. [Online]. Available: https://github.com/dougsm/ggcnn
[20] Z. Weng, H. Lu, J. Lundell, and D. Kragic, "Wengzehang/capgrasp: Capgrasp: An so(2) equivariant continuous approach-constrained generative grasp sampler." [Online]. Available: https://github.com/wengzehang/CAPGrasp?tab=readme-ov-file
[21] F. Macedo, "GitHub - fgmacedo/python-statemachine: Python Finite State Machines made easy. — github.com," https://github.com/fgmacedo/python-statemachine, [Accessed 10-12-2024].