See-n-Pick: Object Detection & Segregation using Mirobot

Fengyi Jiang[1] Tarun Rajnish[1] Alejandro Romero[1]

fengyi_jiang@brown.edu tarun_rajnish@brown.edu alejandro_romero@brown.edu

[1]Brown University

https://github.com/FJiangArthur/2952O_Final_Project

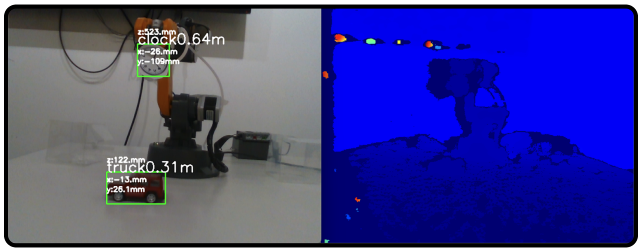

Figure 1: Graphical output of the Yolov5 object detection framework showing coordinates of a toy car relative to the Mirobot robot.

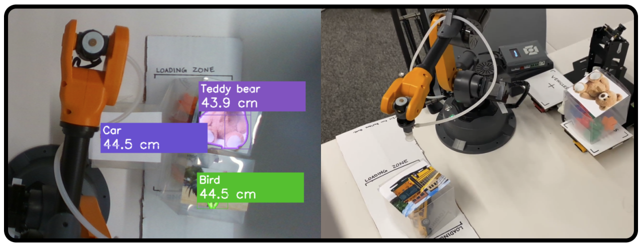

Figure 2: The MaskRCNN object detection framework displaying coordinates of objects relative to the Mirobot robot (left) and the robot sorting the objects (right).

Abstract

Object identification and segregation have been basic necessities of industry since the dawn of the industrial revolution. Different methods have been invented and refined over time in order to do such jobs efficiently and quickly. Many companies now utilize robots and cameras to correctly identify objects on conveyor belts and sort or segregate them based on certain destinations or attributes. However, such high-quality sensors and robots come at a high cost, and state-of-the-art robotic automation necessitates a significant investment. These robots and activities are rarely generalizable. We demonstrate an end-to-end industrial object identification and separation procedure in this study and developed a groundwork for future exploration. In order to do this, we use the Intel Realsense depth camera, the Mirobot robotic arm and two object detection neural network models. Our process is low-cost and precise. We augment it with a user-customizable web application that enhances generalizability and gives the user more control over how a task should be completed. We show how well-formulated pipelines and powerful object detection algorithms may replace expensive sensors, robots, and non-generalizable logic.

1. Introduction

Highly complicated robots and sensors are typically employed in industries such as automotive, packaging, and delivery to detect diverse parts, packages, and other objects and separate or sort them based on an end purpose. Typically, these robots, cameras, and sensors are costly and non-generalizable. For example, the Da Vinci Xi robot [10], which is designed to perform surgery with the help of computer vision algorithms, costs between \$1.855 million and \$2.3 million.

Humans, on the other hand, can easily change their actions based on eyesight and previous experience. To that end, we explore if by using visual feedback and computer vision approaches, we can achieve high precision and control while keeping costs low and generalizability high.

We can observe that today's robotic manipulation labor entails highly specialized activities. Each robot is designed to meet a specific requirement, and there is often little room for additional customization. We propose a web application for our workflow in this paper, which will give the end user additional control over the robotic arm and separation methods, satisfying the demand for generalizability.

We utilize the Mirobot [12], a low-cost desktop robotic arm, to imitate an end-to-end industrial operation at a smaller scale. From object identification to packaging,

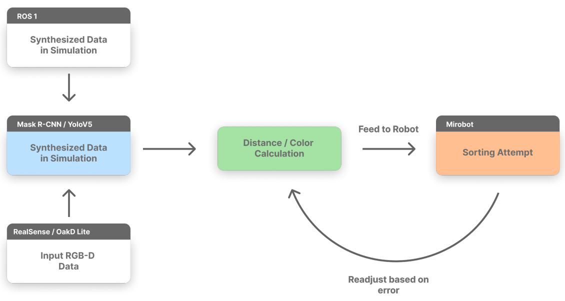

Figure 3: High level architecture of the project. Our approach takes in input depth data to perform transformation calculations to translate coordinates to the Mirobot's world space. These modified values are then fed to the Mirobot to perform the desired action sequence.

package identification, transportation, and delivery, we replicate the complete process. The Intel Realsense [7], a stereoscopic depth camera that costs around $320, is the camera we employ for our purposes. We evaluate two prominent computer vision object detection neural net models, Yolov5 [11] and MaskRCNN [4], in order to accurately identify object classes.

Object detection and picking are performed using Realsense, Yolov5, and Mirobot, followed by packaging depending on object class. Animals and automobiles are two of the classes we consider. After that, the package is transferred to a transporter bot, which carries it to the delivery station. Here, Realsense is utilized once more, but this time in conjunction with MaskRCNN, to identify the correct class package and divide it into clusters. Our online application, built with Flask and React, can control these actions. Finally, the transporter bot is employed to deliver the package and bring our workflow simulation to a close. Section 3 contains further implementation details.

2. Related Work

2.1. Manipulation with Visual Feedback Closed Loop

With recent advancements in deep learning, great effort has been made to directly map visual observations to robotic control tasks. In 2015, S. Levine [8] proposed the idea to train a network for learning perception and control methods jointly end-to-end for better performance on robotic manipulation tasks. Such methods do not require hand-crafting perception or low-level control components. J. Pachocki [5] proposed methods to learn in-hand manipulation through reinforcement learning (RL) through training in simulated environments and then transferring and testing these learned heuristics in the physical world. Another interesting method proposed by A. Morgan [2] trained a network to carry out object 6D pose tracking and achieved high prevision control on the robotic arm without force sensors or precision manipulators by relying on the vision feedback loop.

2.2. Reinforcement learning and self-supervised learning

Since data intervention can be expensive in robotic manipulation tasks, self-supervised learning and reinforcement learning can help obtain optimal models in such cases. Lucas Manuelli [9] demonstrated the possibility of self-supervised visual learning in robotic manipulation and showed such methods provide better generalization, especially in scenes involving occlusion. F. Ebert [3] proposed a vision based self-supervised method for robotic manipulation from raw sensor inputs that can generalize well on both rigid and deformable novel objects. However, such a method cannot handle occlusion or long-term execution and can only handle manipulating one to two objects in a single execution.

3. Method

3.1. Web Application

To increase generalizability, we support our work with a customizable and user-friendly web application. This web application allows the end user to observe object detection progress, change segregation criteria, and add new models and features (Figure 4).

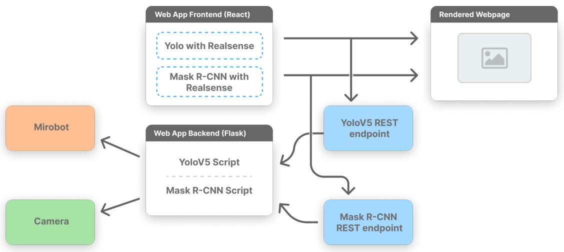

Figure 4: Diagram mapping the main functionality of the web application

Python and the Flask framework are used to build the backend. The backend's primary function is to interface with the multiple model scripts and feed data to the Mirobot. The Wlkata Studio Python SDK is also used to control the Mirobot.

Create-react-app is used to build the frontend in React. It comprises of a simple dashboard with a variety of options in the form of button controls for running specific scripts and viewing the realsense camera stream. React router is used to route REST calls that are mapped to different pathways and send them to the backend.

To serve frames from the backend to the frontend, we use the OpenCV Python library. One of the challenges we faced was overcoming the lag that occurred while serving frames.

Toy Car	Depth (m)	Toy Sheep	Depth (m)
Toy Car Ground Truth	0.47	Toy Sheep Ground Truth	0.30
Experiment 1	0.632	Experiment 1	0.471
Experiment 2	0.811	Experiment 2	0.000
Experiment 3	0.000	Experiment 3	0.512
Experiment 4	0.733	Experiment 4	0.423
Experiment 5	0.000	Experiment 5	0.545
Experiment 6	0.851	Experiment 6	0.000

Table 1. Running YOLOv5 on toy cars and toy sheep placed 0.47m and 0.3m away from center of the OAK-D-Lite camera.

3.2. OpenCV AI Kit Camera by DepthAI

At the onset of the project, we carried out instance segmentation through the Luxonis OAK-D-Lite camera, which has high resolution spatial vision and embedded machine learning capabilities. The goal of using OAK-D-Lite was to implement a simple pipeline for prototyping as the OAK-D-Lite was built with hardware, software and firmware integration in mind. It has an onboard Vision Processing Unit (VPU) that allows developers to run off-the-shelf computer vision models for pilot testing without extensive GPU resources.

We used the pre-processed YOLOv5 model that has been pre-compiled into MyriadX blob format, optimized for inference on the VPU processor on OAK-D-Lite. The YOLOv5 model was used for object detection on a single RGB-D camera input and the bounding box results are fused with the depth information calculated in real-time through semi-global matching. The central pixel of the bounding box is used to generate the depth results of segmented objects, and with the intrinsics of the camera known, object locations in 3D space can be calculated.

However, not until later in the project did we realize that the depth information calculated onboard has a large margin of error for our specific use case where the camera is mounted near the Mirobot within a 1m range. Our experiment in Table 1 shows that depth information on a single object varies largely and created issue for Mirobot manipulation.

3.3. ROS

Robotic Operating System (ROS) provides essential tools such as a hand-eye calibration toolkit, reverse/forward kinematics, communication channels between processes, error handling, and recovery. The team initially used RViZ and Gazebo for integration and simulation between RealSense and ROS. Due to the limited capabilities of the Parallel Desktop virtual machine running on OSX, RealSense encountered serial port driver buffer overflow issues on Ubuntu 16.04 with ROS Kinetic, and we resolved the issue by moving to Ubuntu 18.04 (with ROS Melodic) and 20.04 (with ROS 2 Galactic). With limited capabilities in Ubuntu 16.04 and ROS Kinetic, the team rewrote the Mirobot provided ROS package ***mirobot-urdf-2*** to run in the ROS 2 environment, which involves makefile changes, package upgrades, and switching from the *catkin-make* to the *ament* build tool. All changes can be found in the project Github repo under the *ROS* branch.

We decided not to continue development under this ROS method due mainly to the missing capabilities to control the Mirobot end-effector through ROS. Mirobot provides great python APIs for upper level extraction, but its ROS package is still under development and provides moving joint functionalities only. Extending such functionalities can be achieved by drilling deeper into the middle-ware C code, however it's beyond the scope of this project.

3.4. YOLO v5 [6]

We selected YOLOv5 for instance segmentation for its wide range of classes, speed, and accuracy, and used it to manipulate the Mirobot without the need of explicit rig-dependent calibration. We place an object image, recognizable by the YOLOv5 model, on the Mirobot's end-effector. We then minimize the distance between said object image and the target. We tested out various printed objects images including "bicycle", "truck", and "scooter" and found that the object class "clock" can be recognized best regardless of the orientation of the end-effector.

Following the official implementation of the YOLOv5 model [6] and getting the bounding box through RGB-D outputs of the RealSense, we randomly select several points within the bounding box, get the corresponding depth information from the RealSense point cloud, and average out these depths to output a single depth value of the object relative to the camera. Similar to the OpenCV AI Kit implementation, camera intrinsics were then used for 3D coordinate calculation. During testing of the above method, we found that the end-effector depth value was always faulty even though the depth of the target objects were accurate. The 3D coordinate calculated from depth values carries such error and results in failure of manipulation for most trials.

3.5. MaskRCNN [1]

We employed MaskRCNN [1] on the Realsense image segmentation task as another potential integration for Mirobot. We trained the model to identify table-top objects. Specifically, we wanted to sort between vehicle toys and animal toys. We feed the Realsense camera feed to the MaskRCNN framework to present a graphical interface displaying object masks and corresponding labels, which display the object name and distance from the camera (Figure 2).

We then perform a rudimentary coordinate-based scene mapping to create a set of instructions for the Mirobot robot to follow. It ignores all objects in the drop-off zone to prevent picking up objects that have already been sorted. The algorithm takes all objects it recognizes (in this case, we specified only animal and vehicle toys could be acted upon) and stores them in a queue. For each item in the queue, the algorithm uses the central x and y coordinates of its object mask, and its depth as the z coordinate. These coordinates are then translated from the Realsense camera's coordinate space to the Mirobot's world space.

The resulting coordinates are then passed into Mirobot command prompts, which have the robotic arm pick up the object. It then selects the correct drop-off location based on the object's class (animal or vehicle). These target locations were predetermined, but the vision model can be used to determine these locations in real-time by using markers such as ARTags.

4. Results

We met our main goal by successfully integrating MaskRCNN, RealSense, Mirobot, and a Flask web application for low-cost production line manipulation tasks, and have proven that it's possible to achieve high precision control without expensive equipment. Our approach resulted in successful object detection and segregation using the MaskRCNN-Mirobot integration, which provided reliable performance on the task.

Our approach failed on the manipulation task through YOLOv5. We implemented the general pipeline for a fine-control task using a closed feedback loop, but due to inaccuracies in depth data from the Realsense and coordinate data from Mirobot, we were unable to reliably succeed at the task via this approach. Future work may encompass finding the root cause for the mismatching depth information and implementing an algorithm such as a running average or a low-pass filter for noise control to produce more accurate behavior.

5. Conclusion

We formulated a low-cost and precise pipeline for leveraging state-of-the-art object detection techniques to inform the robotic manipulation task of object sorting. Though we encountered issues with noisy input data resulting in inaccuracies during one of our tasks, our work creates a generalizable framework that allows users to integrate various computer vision algorithms with the Mirobot robot, eliminating the need for explicit camera calibration and simulation via ROS. Additionally, we present a method for providing control of the pipeline to an end user via a novel web application.

6. Future Work

Though our work's main contribution is a comparison of different computer vision techniques for performing object sorting tasks inexpensively and in a generalizable format, it has many other potential avenues of expansion. For one, we explored a rudimentary closed feedback loop for real-time error correction as an alternative to explicit camera calibration. This loop could use further refinement such as by integrating an on-board camera mounted onto the robot's end-effector to better inform mid-flight trajectory adjustments. Further, we set the groundwork for an open source web application allowing users to remotely operate and supervise the robot arms. This web application can be further developed to allow for more fine-grained user control. We only use one depth camera and could benefit from the integration of various simultaneous camera feeds to increase accuracy in both object localization and robotic manipulation.

7. Division of Labour

7.1. Tarun Rajnish

Worked primarily on full-stack web application development in order to tie the MaskRCNN and YOLOv5 scripts between the front-end and the Mirobot. The back-end server was developed using the Flask framework and Python, while the front-end was in React. Assisted in the creation of the transporter bot. Helped with the presentation slides, the poster, and the final report.

7.2. Fengyi Jiang

Experimented on ROS Integration with Mirobot through ***MoveIt*** for Inverse Kinematics, forward Kinematics, collision prevention, cartesian path planning, ***ROS-bridge*** for communication between ROS and external python process, hand-eye calibration using ***easy-handeye***. Experimented on OpenCV AI Kit for instance segmentation and extracting 3D coordinates of the objects. Implemented YOLOv5 for instance segmentation and object 3D coordinate extraction. Experimented in controlling the Mirobot by minimizing the distance between target and Mirobot end-effector. Integrated the feedback model with Mirobot to perform the manipulation task.

7.3. Alejandro Romero

Developed the MaskRCNN solution, which included integrating the Realsense camera with the computer vision model, visualizing the model's output, and writing transformation calculations for converting from Realsense coordinates to 3D coordinates in the Mirobot world space. Handled the Mirobot command queue and motion planning. Assisted in creation of the transporter bot as well as testing the functionality of the web application. Helped with the presentation slides, the poster, and the final report.

References

[1] Waleed Abdulla. Mask r-cnn for object detection and instance segmentation on keras and tensorflow. https://github.com/matterport/Mask_RCNN, 2017. 3

[2] Junchi Liang Abdeslam Boularias Aaron M. Dollar1 Andrew S. Morgan, Bowen Wen and Kostas Bekris. “vision-driven compliant manipulation for reliable, high-precision assembly tasks”. *Robotics: Science and Systems*, 2021. 2

[3] Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control, 2018. 2

[4] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask r-cnn. *ICCV*, 2017. 2

[5] M. Plappert G. Powell A. Ray et al. J. Pachocki, A. Petron. “learning dexterous in-hand manipulation”. *The International Journal of Robotics Research, vol. 39, no. 1, pp. 3–20*, 2020. 2

[6] Glenn Jocher. ultralytics/yolov5: v3.1 - Bug Fixes and Performance Improvements. https://github.com/ultralytics/yolov5, Oct. 2020. 3

[7] Leonid Keselman, John Iselin Woodfill, Anders Grunnet-Jepsen, and Achintya Bhowmik. Intel realsense stereoscopic depth cameras. *CVPR*, 2017. 2

[8] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies, 2015. 2

[9] Lucas Manuelli, Yunzhu Li, Pete Florence, and Russ Tedrake. Keypoints into the future: Self-supervised correspondence in model-based reinforcement learning, 2020. 2

[10] James Chi-Yong Ngu, Charles Bih-Shiou Tsang, and Dean Chi-Siong Koh. The da vinci xi: a review of its capabilities, versatility, and potential role in robotic colorectal surgery. *NIH*, 2017. 1

[11] Ultralytics. Yolov5. *GitHub*, 2022. 2

[12] Dongxu Zhou, Ruiqing Jia, and Mingzuo Xie. Mirobot: A low-cost 6-dof educational desktop robot. *Springer Link*, 2021. 1