Final Report

Grasping Regions through Learning and Computer Vision Techniques

CM3203 – ONE SEMESTER INDIVIDUAL PROJECT

CREDITS: 40

SCHOOL OF COMPUTER SCIENCE

CARDIFF UNIVERSITY, 2022

Author: Hebe Wrench

Supervisor: Juan Hernandez Vega

Moderator: Stuart M Allen

Abstract

Grasping objects is one of the basic and most common tasks that a robot (such as manipulator arms or mobile manipulators) has to do. While it is considered a simple task for humans, sometimes it could be a challenging robot operation. For a robot, grasping an object normally involves perception to detect the object, planning to approach and grasp the object, and control to drive the robot through the planned motion. With the increasing popularity of learning techniques, new frameworks/libraries have been proposed to automatically detect the best grasping poses of an object.

This project aims to implement a learning and computer vision-based framework to determine the grasp affordances of an object. More specifically, the framework uses computer vision to detect and learn the best grasping poses of an object that might be surrounded by obstacles in cluttered settings. *The expected output of the proposed framework is one or more grasping regions around the object or, alternatively, a set of grasping poses.*

This project is simulation based and uses the CoppeliaSim robotic simulation program with the Franka Emika Arm to demonstrate the applications of the proposed framework.

Acknowledgements

I would like to thank my supervisor, Juan Hernandez Vega for all of his support throughout the duration of this project. Our meetings greatly helped with keeping me motivated and remaining on track despite difficulties that were encountered. His feedback throughout has also been incredibly helpful and insightful in enabling me to enhance the project.

I would also like to thank Furkan Duman and José Johil Patino Minan for providing me with the initial code for the manipulation of the Franka Emika Robot Arm and for providing support when needed.

Additionally, I would like to thank my housemates and friends, Solvig, Cai, and Lou for also having large amounts of work themselves during this project, helping me to stay motivated and consistent with my work efforts as they were conveniently working alongside me. Also for generally keeping spirits up and helping to maintain decent university life outside of working on this project, thus greatly helping to prevent burnout.

Finally, I would like to thank my sister, Anis, for checking through and making sure that I had finished all of my sentences and had not made too many grammatical faux pas.

Table of Contents

Table of Figures

Introduction

Project Aims

This project is focussing on using machine learning techniques to identify specific objects and plot routes to pick up said objects, and intends to demonstrate this via simulation using a CoppeliaSim and a Franka Emika Robot Arm. The main aim of this project shall be to identify the specific objects in a variety of locations - i.e. on a cluttered table or cluttered shelf where there could be a potential obstructions above the objects - then to plot the best route to the object and propose a range of grasping techniques to pick it up. There are already existing algorithms for this problem but they tend to be fairly time consuming and thus I am aiming to utilise machine learning to reduce the time that these tasks generally take. I will be primarily focussing on the perception aspect of this task as my background is computer science - having completed several modules in artificial intelligence, image processing, and computer vision during my degree - rather than robotics.

Intended Audience and Beneficiaries

The intended audience for this project is the robotics community as a whole and industries that currently have human workers doing mundane repetitive tasks - such as selecting ordered items and putting them into the correct boxes - that could be automated instead.

One application for this project is the Amazon Picking Challenge (Amazon Robotics Challenge, 2017), which was a challenge to try and improve the uses of robotics and the existing algorithms within robotics. The goal of the challenge is to pick out 12 of 25 objects from a warehouse shelf and place them into a storage container within 20 minutes, and the applications of a successful algorithm can lead to more automation within this industry. This is still a challenge on a large scale, for instance, in Amazon warehouses.

Overall, the audience is any industry or individual who could require a robot to pick up items and move them to secondary locations - there are many examples of industrial and individual applications for this. An example of an individual use of this project would be as support robots for people with mobility issues, so that they could be marginally more independent. This kind of application is also demonstrated by the Toyota Human Support Robot (Upton, 2021) which can help individuals with tasks such as object retrieval.

Project Scope

This project covers object detection using machine learning as well as path and grasp planning of the Franka Emika robot arm. While the project is specifically focussing on the Franka Arm, it should be relatively generalisable to other robot arms with similar degrees of freedom and joints.

The project has several stages: first, there is the issue of object perception and deception, then object grasping planning, then robot arm planning. These are the key aspects that will be focussed on.

The first stage (perception and detection) will take the existing code and add in machine learning techniques to attempt to optimise the routine and reduce the time required for this process. The second stage (object grasping planning) will also attempt to optimise the existing code by using machine learning algorithms. The optimisation of this process should

be especially evident in the cluttered scenes where multiple cylinders are getting moved in turn. The final stage will also look to reduce operation time of the path planning again, ideally, by using machine learning techniques.

Approach Used

As there were many potential methods that could have been used for tackling the problems this project was addressing, it was decided that there would be an almost brute force approach to trying out machine learning solutions. This was decided with the understanding that if one method was either particularly inefficient or causing difficulties with connecting it to the existing code, another method could be attempted to see if that would solve the issue instead.

Assumptions the Work is Based On

This project is building off of the work of José Johil Patino Minan, a Masters' student, who created the initial code for the path planning, object detection, and grasp detection aspects of this project. His work forms the initial infrastructure that this project is based on and attempts to improve upon via the addition of machine learning techniques. As this project building on his work, José's code will be used as the baseline for benchmarking so that it will be clear if this project has been an improvement or not.

Beyond the initial code, a key assumption that this project has made is that by implementing machine learning algorithms, the completion time for the task at hand (detecting an object, then planning the path and grasping pose necessary to pick it up, then picking up the object and relocating it) shall be greatly reduced.

Summary of Outcomes

Overall, this project is aiming to optimise the object detection and object grasping pose detection algorithms used in the existing program for manipulating the Franka Robot arm. It aims to implement machine learning techniques to improve and accelerate the initial detection and planning aspects of the grasping regions task.

The anticipated outcomes also include proposing a range of grasping techniques that could be used to identify the grasping poses necessary to pick up the object, planning a path to the identified object, and simulating picking up the identified object. Additionally, once these have all been achieved, benchmarking the proposed solution with the alternative approach that is not based in machine learning techniques to identify the extent to which it has improved the execution time, if it has at all.

Background and Related Work

Overview

In this section, I will be giving a brief overview of deep learning and reinforcement learning techniques and their applicability to the problem at hand. Then, after a brief overview of the overall problem this project is addressing, I will be reviewing some relevant work on computer vision and deep learning based grasping detection algorithms, with the objective of building on this prior research and reducing execution time of the source code. Additionally, I will be looking at the gaps left by the existing research that this project will aim to fill.

The problem that this project is focussing on is the overall execution time of robotic arm manipulation algorithms in an attempt to make robotic arms more feasible as replacements for humans when it comes to repetitive pick-and-place style tasks.

Current algorithms for object and grasp detection are time consuming (Wei, 2022). This is because, although for humans, this is a simple and automatic action where we identify objects and then pick them up with relative ease, for robots this task must be broken up into the individual steps: object perception, distance detection, grasp planning, path planning, then execution. The program that is being modified currently does not utilise deep learning algorithms for many of these steps, so this will be a focus of the background and related works section.

Deep Learning vs Reinforcement Learning

Many different aspects of robotics and computer vision can utilise machine learning techniques (Niroui et al. 2019), for example, image classification and target object detection. Although deep learning and reinforcement learning are fairly similar, there are key differences that distinguish them from one another. A key difference between them is that deep learning algorithms are generally trained by existing big data to identify patterns and use these to inform predictions about novel data, whereas reinforcement learning essentially learns by trial and error (Marr 2021). As a result of this, in terms of robotics with particular interest in exploring unknown and new environments, reinforcement learning techniques are often used (Niroui et al. 2019). Although reinforcement learning algorithms tend to have low dimensionality (Arulkumaran et al. 2017) this can be combated through using convolutional neural networks (CNN's) which progress the reinforcement learning techniques to deep reinforcement learning techniques.

CNN's work by taking the initial input and convoluting it across several layers while extracting the features. Then pooling and flattening this data so you end up with a fully connected and classified set of data that is able to accurately identify certain objects etc. This is demonstrated below in the figure.

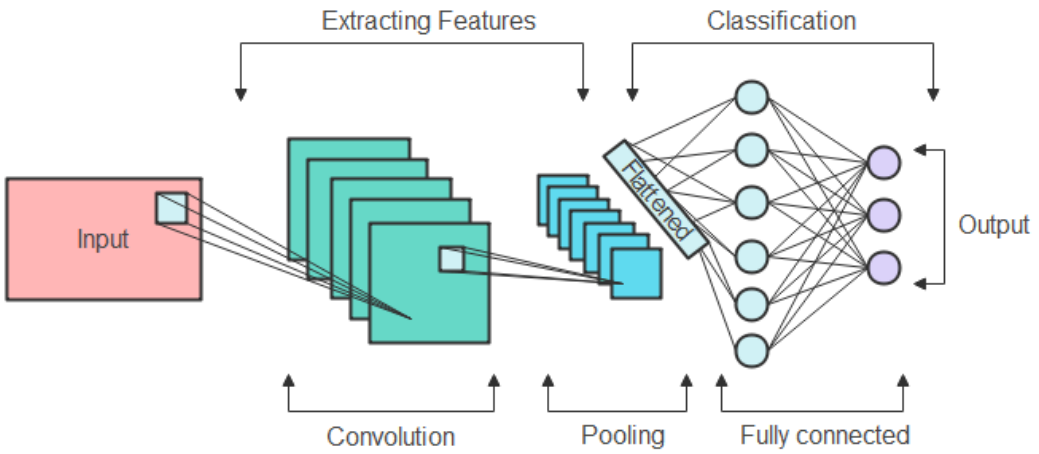


Figure 31: visualisation of a CNN

Computer Vision with Machine Learning

With robotic grasping algorithms there are two approaches that could be taken, as outlined in the DexNet paper (Mahler et al., 2017). These are: using analytical grasp planning methods which register images and involves segmentation, classification, and geometric pose estimation from 3D point clouds, and using deep learning from the information that the cameras on the robot have supplied. Using the camera and related sensors for the grasp and object detection can produce adequate execution times, however, for training the algorithms, simulated or real image data sets are necessary and can provide their own advantages and disadvantages as well. The DexNet paper also describes empirical training methods which use machine learning to directly map from robotic sensor readings; these can be expensive to acquire for large datasets but have been used for successful and fast CNN-based detection models (Mahler et al., 2017).

In terms of estimating grasping regions, algorithms typically require 3D models of the objects (Zeng et al. 2019) - this means that determining the grasp poses for novel objects is much more difficult if the system is only using a limited selection of views of the scene or if there are obstacles in front of the desired object. This issue can be mildly mitigated, as shown by Jiang et al. (2016) who described a robotic system that can handle novel objects by actively rearranging objects in the scene (by pushing) in order to improve recognition accuracy.

Determining Grasping Poses using Deep Learning

As described in the GraspNet paper (Fang et al. 2020), there are 3 main categories for deep learning based grasping detection algorithms. These are: using RGB-D image input to detect a graspable rectangle; using an existing grasping dataset that has been annotated by humans; and point-cloud based deep learning.

The GraspNet paper (Fang et al. 2020) introduces the graspnet architecture which uses depth sensors to generate a point cloud, then outputs a ranked list of 6D grasp candidates (Wei, 2022).

An example of a point cloud is shown below. Point clouds are used to detect the potential points on a particular object that could be a grasping point.

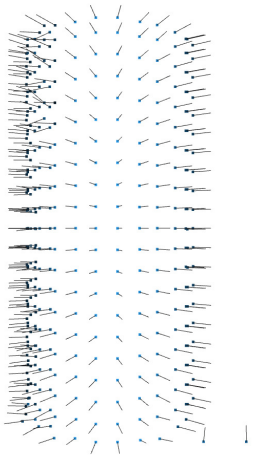


Figure 32: visualisation example of a point cloud for a cylinder

The dataset required for training was massively larger than previous grasping datasets but this did result in higher success rates than the benchmark comparisons. Meanwhile, Mahler et al. (2017) proposed DexNet which uses a GQ-CNN (grasp quality convolutional neural networks) algorithm specifically for an ambidextrous robot that has both a jaw gripper and suction cup. While this approach was 3x faster and just as reliable a method as point cloud registration (Mahler et al., 2017), it required a very large dataset and only returns 4D grasp poses, which constrains applications to a 2D plane (Wei, 2022).

Many approaches (Wei, 2022)(Fang et al. 2022) for determining grasp poses using machine learning use Pointnet++ (Charles Ruizhongtai Qi et al., 2017), which is an extension of PointNet that has an added hierarchical structure. Benefits of PointNet++ over standard PointNet include the fact that PointNet++ does not require uniformly sampled point sets to be able to learn features with a reasonable success rate. PointNet++ works through recursive functions on a nested partitioning of the input point set and due to this is effective in learning hierarchical features (Charles Ruizhongtai Qi et al., 2017).

Along with the 3 main categories for deep learning based grasping detection, there are additional methods within each of these for grasp pose detection - either top-down approaches or methods which can output 6DOF grasps (Breyer et al. 2021). The 6DOF approaches return 3D point-clouds and identify the object surfaces. There are multiple different methods for creating the point clouds (Kasaei and Kasaei, 2023), for example, moving the robot around the scene to obtain multiple views (Breyer et al., 2021). This kind of approach generates higher success rates in real-robot experiments than other approaches due to the greater variety in views (Breyer et al. 2021) of the target objects. The approach by Breyer et al., discovered that the top-down approaches were, overall, more successful in the real-robot experiments than in the simulated experiments. They also showed that their approach was generally more successful than the benchmark comparison approaches in both real and simulated experiments. Another method to obtain the additional views is by

manually placing the sensor in different views to generate the point clouds (Hu and Pagilla, 2022), which was found to be a more robust method than the baseline approaches.

Deep Learning on Motion Planning

Piecewise prediction is a common method used to help break the task of incorporating deep learning methodologies with motion planning (Wei, 2022). For example, in the Grasping In the Wild Paper (Song et al., 2020), they discuss their approach whereby they used a reinforcement learning algorithm and multiple different RGB views of the robot arm. They developed the dataset for training the algorithm by having 8 participants perform various pick-and-place tasks in a variety of different environments. This generated over 12 hours worth of gripper-centric RGB-D videos, which were labelled with the time of when the user pushed the button to close the gripper. This was able to generate accurate labels for when and where the robot should be closing the gripper in order to pick up an object. As a result of this large labelled dataset and the training algorithm that they used (deep neural networks), they were able to achieve a 92% grasp success rate. Additionally, they found that by using this demonstration data for training they were able to improve both grasping performance and learning efficiency (Song et al., 2020) as well as the ability to handle a variety of different pick-and-place type situations and environments.

Another approach, (Pfeiffer et al., 2017), uses the robot's vision sensor directly to plan the route, rather than predetermined RGB-D videos. They found that it was possible to learn a navigation policy using this type of approach and that it was possible to train a model in simulation, then deploy it into a real platform with satisfactory levels of success. Their method input raw sensor data, from the vision sensor attached to the robot arm, then utilised CNN architecture to map this data to steering commands. Their CNN model computed the steering commands frame-by-frame and thus did not need to account for any previous results.

The Grasp-Optimised Motion Planner (GOMP) paper (Ichnowski et al., 2020) incorporated the dynamics of a robotic arm and a set of candidate grasps to reduce the execution time of the algorithm. They did this by recasting the problem to a sequential quadratic program (SQP) which led to a definite improvement in the execution time for the entire pick-and-place operation. But they believe that incorporating the DexNet pipeline or accounting for avoiding dynamic objects could further improve this work.

Existing Gaps In Research

There are many examples of research in this subject area but there are still limitations to the proposed algorithms and, of course, still gaps to fill. None of the existing algorithms have managed to achieve a 100% success rate for picking-and-placing objects, particularly not for novel objects. Some of the approaches also focus on solely cylindrical objects (Wei, 2022). Additionally, algorithm training and execution times are generally rather long, potentially due to large datasets or novel environments. Although there were examples where obstacles were moved to get better views of desired objects (Jiang et al., 2016) these algorithms were not very stable and were also very time consuming.

Approach

Initial Design

After conducting the necessary background research into similar projects which are also trying to solve the problem of identifying and moving objects from cluttered surfaces, there were many potential machine learning algorithms that could be implemented at each stage of this project. As a result of this research, which showed many positives for all the different algorithmic routes, it was decided to try and attempt a range of different approaches to determine which would produce the most efficient and best improvement to the source code.

For the object perception and detection aspect of the problem, the initial plan was to implement a convolutional neural network in an approach similar to that demonstrated in the aforementioned GOMP paper (Ichnowski et al., 2020). For this CNN, an initial cylinder training dataset was acquired from Kaggle (de, 2021) that was used in a paper on probabilistic pose estimation (Del et al., 2021). This dataset was already split into training, validating, and testing directories, making it a strong basis for a CNN.

For the grasp planning aspect, it was decided to initially attempt to implement machine learning with the assistance of the PointNet++ architecture. This approach was decided on, along with using the RGB-D information, because the initial source code also made use of this information and so implementing PointNet++ should have been a relatively easy method to try first.

Then, for the motion planning stage, the initial design decision was to use a CNN and the RGB-D information, taken from the robotic arm, and utilise this in the machine learning algorithm. This decision was taken as the two preceding stages were already using this data, so it would be more efficient than having to use new data. It was also decided that if all stages of development were completed with ample time to spare, this stage would attempt to implement the DexNet pipeline to optimise it even further, as was suggested could be possible by previous research.

In terms of task prioritisation, the path planning is the aspect which takes the longest to complete in the benchmark solution so that is the main priority focus as it will have the largest effect on execution times.

Specification Design

User Interface

In terms of user interface, PyRep enables the whole project to be launched with ease through the terminal. This then launches CoppeliaSim which demonstrates the simulation. In CoppeliaSim, the scene can be traversed and viewpoints rotated while the simulation is running (depending on the scale of the program being simulated as if they are labour intensive, this functionality is much less consistently successful). The simulation demonstrates the robotic arm perceiving the objects on the table and then deciding on

appropriate grasping poses and the route to take to move the cylinders from one location to the next. This is demonstrated below.

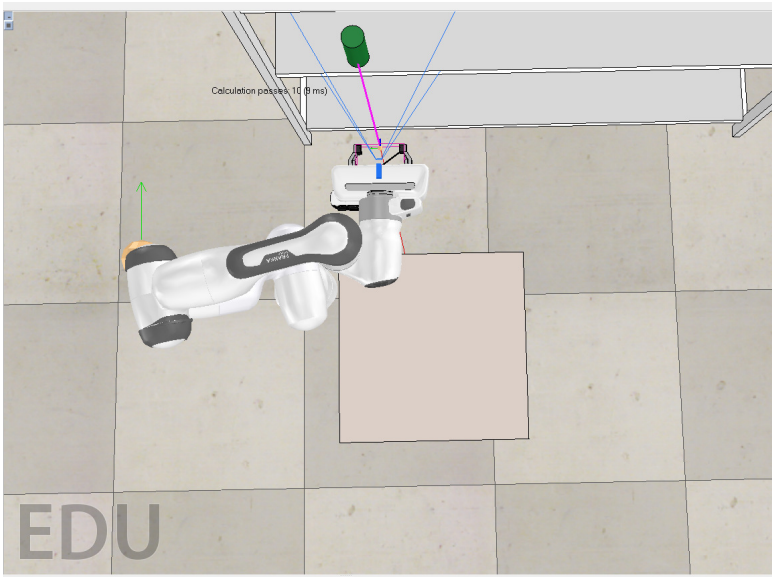


Figure 1: Franka Emika Arm in simulation planning the route to pick up a single cylinder

The environment within CoppeliaSim displays a small table placed next to a tall shelf, where a variety of different coloured cylinders are placed, and the Franka Emilia Robot Arm placed between the two. Depending on which simulation was chosen, there are between 1 and 5 cylinders on the second shelf, directly in front of the robotic arm, in its field of vision - these are the desired target objects.

Once the simulation has begun, there is no necessary interaction from the user with the system. The simulation will run until all desired objects have been moved and the calculations have all finalised. Upon completion, CoppeliaSim closes automatically.

Project Background

The project is primarily focusing on simulating using the Franka Emika Robot Arm but should be applicable to other robotic devices and, post simulation, to physical robotic systems.

The project aims to implement machine learning algorithms to optimise the run time of the software already developed by José Johil Patino Minan as part of his masters' project. This existing software is programmed in python and has two distinct parts, which can then be broken down into 3 very similar pipelines each. The two parts refer to whether the program is picking-and-placing a singular object or multiple objects. When identifying and moving a singular cylinder, the environment is not cluttered and there is just one cylinder on a shelf in line with the robots vision sensors. Whereas, when performing the task in the multiple cylinders variation, there are a minimum of three cylinders on the shelf, for the robot to identify and move in turn. The 3 steps in the pipeline, which both aforementioned parts follow, are: 1. object perception and detection, 2. grasping regions detection and decision, 3. path planning to the identified object.

The initial runtime of identifying a singular object and moving it from the shelves to the table takes a minimum of approximately 5 minutes to complete the simulation. Performing the tasks on multiple objects from a cluttered surface and moving each object in turn is an even longer process. Using 5 objects total to create a cluttered surface, the runtime is a minimum of 1 hour 44 minutes, while if the number of cylinders is reduced to 3 the runtime is approximately 1 hour.

This project has a distinct aim to reduce both of these runtimes as much as possible via the addition of machine learning.

Dynamic Behaviour of The System

The functionality of the project is very dynamic. The focus is on a simulation to demonstrate the implementation of machine learning into the computer vision and grasping pose detection aspects of the problem. In addition to this, there is also the physical movement of the robotic arm which happens both at the beginning of the simulation, so that the robot can use its vision sensors to determine the location of the desired objects, and after the motion planning phase, when a route has been chosen.

In the benchmark system, at each stage of the object detection and grasp pose detection process, the system displays a visual indication of what stage it is at. This means that users can understand where in the process the system is, and can also establish whether the arm has correctly identified all of the objects in the scene.

Code Partitioning

The code that formed the basis of the project has 11 functions. These cover the robotic arm orientation and configurations, the cylinder detection, the gripper movement and rotations, and vision. All of these are contained within a singular python file along with the motion planning code, which is not contained in a function. The environments that are called during the simulation are hard-coded into the single cylinder and multiple cylinder files but are separate .ttt files.

For the machine learning additions, these are placed into a separate file where the functions can be called from. In addition to this deliberate segmentation, the data preprocessing is also done within a separate file as it does not need to be run multiple times if the training dataset is remaining unchanged. Instead it can be run once, separate from the simulation, and the reshaped data then stored in .npy files, which can then be called in the machine learning functions. By splitting these functions across several files, the likelihood of accidentally breaking the base code is greatly reduced as they can be made and tested so that they are known to be working as desired before adding them into the main program.

The new machine learning functions will be called within the existing functions in the base code and should be replacing the non-machine learning aspects in particular.

Implementation

Initialisation

The project is limited to simulation based results and these are demonstrated through the use of the robotics simulation software CoppeliaSim. This is an open-source software that was required to be downloaded in order to work on and run this project. Additionally, a toolkit for CoppeliaSim called PyRep was required. This toolkit enables CoppeliaSim to be launched, and largely coded, through python. Pyrep requires Ubuntu 16.04 - 20.04 and CoppeliaSim 4.1 (and will not work with other versions of either). In addition, both require the robotic operating system (ROS) to be installed on the device to provide them with full functionality. Furthermore, there are several specific python libraries that the system also requires; these are listed in the requirements.txt file. The simulation demonstrates the code functionality via a Franka Emika Robot Arm.

The simulation is launched via a terminal operation of "python3 ml_tesispy_one.py" or "python3 ml_tesispy_multiple.py" which then will launch the respective simulations. The scenes are set up with the Franka arm placed between a shelf and a small table, on the second shelf there are either 1 or 5 cylinders, depending on the chosen simulation. An example of this, demonstrated for detecting and moving a singular cylinder is shown below in figure 1.

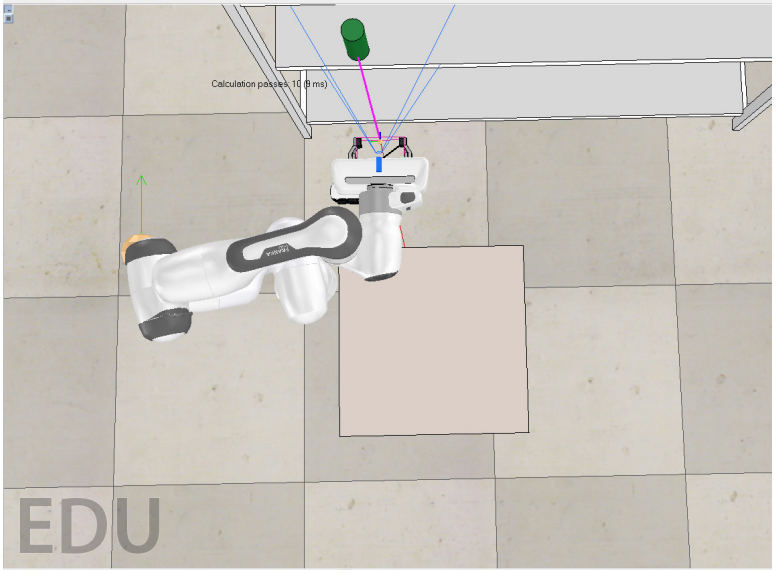


Figure 1: Franka Emika Arm in simulation planning the route to pick up a single cylinder

The angle at which you may view the simulation can be changed before the calculations begin, or more reliably, separately to the simulation and then saved to be run later. In the above figure, the viewing angle is the current default for the single cylinder environment.

Data Preprocessing

For the creation of a CNN for the computer vision aspects of this task the dataset first had to be preprocessed and reshaped. The dataset used for training the computer vision CNN was already split into three directories for training, testing, and validating. The training directory contains over 250,000 images and their respective labels and the testing and validating data sets contain over 84,000 images and labels each as well. The images show cylinders in

various orientations on checkerboard backgrounds. Where cylinder faces were shown in the training set, they showed either a yellow square or a purple triangle, while in the testing and validating sets they showed either a green square or a red triangle. These datasets were further split into hundreds of sub-directories, each containing approximately 13 png and txt files (a pair of txt and png making up one example image), and a rotation and translation txt file (which could be applied to the whole subdirectory). The png processing function is shown below.

```
# Image preprocessing using ImageDataGenerator
train_image_generator = ImageDataGenerator(rescale=1./255)
val_image_generator = ImageDataGenerator(rescale=1./255)
test_image_generator = ImageDataGenerator(rescale=1./255)

# Helper function to load and preprocess an image
def preprocess_image(image_path, image_generator):
    image = Image.open(image_path)
    image = image.resize((256, 256))  # desired width and height
    image = np.array(image)
    image = image_generator.random_transform(image)
    return image
```

Figure 2: snippet of image reshaping and data preprocessing code

The images are rescaled through the ImageDataProcedure and resized in the preprocess function ready to be called for each image at a later point. Within the data preprocessing code there is another very similar function for preprocessing the label text files.

Trial and Error

As there were many potential different methods for adding in machine learning throughout the pipeline, it was decided to adopt a bit of a trial-and-error approach to see what could actually work with the existing system.

For the computer vision aspect of the task, a CNN was initially decided on for the first method to attempt to implement. For this, the data had to be preprocessed so that it was ready for training and had all the necessary relevant data. The preprocessing of the data caused several issues as the dataset was so vast that attempting to use the laptop for tasks other than preprocessing, in parallel to the preprocessing caused crashes. However, eventually it seemed that the data had successfully been preprocessed and was ready for training a CNN. Upon running of this network, an error was thrown as the preprocessing had not accounted for the additional translation and rotation files that were within most of the subdirectories within the dataset. This meant that the x and y arrays were different sizes and the CNN wouldn't work. It was then attempted to correct this array size disparity but due to the labour-intensive nature of preprocessing the dataset, this was without success so other methods of implementation for machine learning were considered.

A deep q-learning method was considered due to its known uses within robotics in unknown environments, again this method did require the dataset to be preprocessed, however. Another method was thus tried for preprocessing and preparing the data, but again after many long and unsuccessful attempts, other routes were again considered.

These included attempting to use the PointNet++ architecture or a Mask R-CNN approach. Unfortunately, these also proved to be unsuccessful due to the dataset being too large for the laptop to reasonably process.

As a result of the slow going on the computer vision aspect, a change of tact was necessary so the focal point was adjusted for a while. This shifted the focal point away from creating a new computer vision function and onto creating the benchmark data from the source code so there would be a definite baseline to acknowledge and work towards improving upon. To do this, a series of timers were added throughout the code to output the time after key events, such as object or grasping pose detection, had occurred. Upon doing this, it was found that the path planning was by far the most time-consuming aspect of the code, as opposed to the general object or grasp region detection functions. As a result of doing this, it was found that the route planning took up the vast majority of the execution time. Thus, another change of tactic was taken and to try and reduce the execution time of the path planning.

One of the initial aims had been to improve the execution time of the path planning through machine learning and so A-star algorithms were initially considered as a potential route. But these generally don't operate on a 3 dimensional plane and if they do, they do not consider movement with as many DOF as the Franka Arm uses.

Difficulties

This project was not without difficulty, particularly in terms of unforeseen problems that arose. The first unforeseen problem that was encountered was that the laptop being used for the project did not agree with the Ubuntu installation process. This caused a great delay in the initial few days of the project as due to no longer having any operating systems installed, little to no progress could be made. Eventually, this problem was resolved and working on the project was actually able to begin.

The next major unforeseen problem that I had was that, to speed up the initial testing of the machine learning algorithms for object perception, I had created a new file that I was working in, rather than running through the original code file. I did this because I decided it was less likely to cause any irreparable damage to the source code and would mean that I could run through the testing process much faster as I wasn't having to launch CoppeliaSim each time. Theoretically, this was a good idea, however, in practice it meant that I didn't try testing the code in the simulation software until very late on, when I discovered that some of the file dependencies and files had been corrupted and no longer worked. This meant I couldn't run any of the simulations, so had to reinstall and try to fix many of these files. Unfortunately, while attempting to fix this issue, another issue arose: Ubuntu crashed and stopped booting. This meant that, again, I had to spend several days reinstalling and mending the system to try and get the software to work once more. Yet again, Ubuntu was reinstalled, along with ROS, CoppeliaSim, and Pyrep, and another attempt to test the code to check that CoppeliaSim would work was made, but unfortunately that error persisted. Due to the scale of this issue, progress with the project itself was massively stunted. Eventually, the issue was discovered - the wrong version of CoppeliaSim had been installed and was not compatible with PyRep so could not be launched through that.

Non-Machine Learning Improvements to the Source Code

During the benchmarking determination stage, it was discovered that the slowest aspect of the source code, particularly for the singular object code, was the motion planning aspect. Overall, the computer vision aspects of the code were very efficient and after a closer inspection and deeper understanding of the code, the reason for this became clear: these two aspects already utilised machine learning. As a result of not wishing to break the source code, it had not fully been investigated or understood until the timers had been added and benchmark data collection had begun, which was relatively late in the project time-scale. This meant that the fact that machine learning was already in use was overlooked until there was not a lot of time left to complete the project anyway. Due to the limiting time factor, the approach was once again updated to attempt to improve the execution time without using machine learning - especially as I do not have a background in robotics so attempting to implement machine learning into a relatively new area in a very short time was not overly feasible.

One way in which the execution time was reduced without machine learning was through commenting out the aspects of the code which returned the visualisation of the computer vision and grasp design processes. This massively reduced operation time as it could not proceed to the next step until the most recent image had been closed and this removed the necessity for the user to be engaging with the system during operation. Further improvements made include reducing the number of trials required in both the individual cylinder and the multiple cylinder code. These were reduced from 600, for the singular cylinder, to 50, despite this massive reduction the success rate remained at near 100%, while the execution time became approximately 1 minute - down from approximately 6 minutes. With the multiple cylinders variation this time reduction was even more drastic and obvious.

A further improvement to the code was made to the orientation function as it would occasionally throw the error displayed in figure 3 below.

```
total object detection time: 1.3961713314056396
Grasp Pose detection time: 1.3593652248382568
tesispy_one.py:60: RuntimeWarning: invalid value encountered in double_scalars
  R= np.identity(3)+ v_skew + np.dot(v_skew,v_skew)*( (1-c)/(s*s))
```

Figure 3: snippet of error message displayed on some runs of the code

This fortunately was not a difficult error to solve and just involved updating the orientation function to account for the possibility of ‘s’ being 0 and causing a divide by zero error. This update is shown below in figure 4.

```
s=np.linalg.norm(v)
v_skew = np.array( [ [0,-v[2][0],v[1][0]], [v[2][0],0,-v[0][0]], [-v[1][0],v[0][0],0] ] )
if s == 0:
    R= np.identity(3)+ v_skew + np.dot(v_skew,v_skew)*( (1-c))
else:
    R= np.identity(3)+ v_skew + np.dot(v_skew,v_skew)*( (1-c)/(s*s))
return R
```

Figure 4: snippet of code to demonstrate the error catching update.

Results and Evaluation

Aims of the Project

The goals in this project were:

1. To implement machine learning algorithms to the object detection aspects of the source code
2. To benchmark these algorithms against the source code and see a reduction in execution time
3. To implement machine learning algorithms to the grasp pose detection aspects of the source code
4. To benchmark these algorithms against the source code and see a reduction in execution time
5. To implement machine learning algorithms to the route planning aspects of the source code
6. To benchmark these algorithms against the source code and see a reduction in execution time
7. To benchmark the whole system against the source code and see an overall reduction in execution time.

Demonstration of System Outputs

With zero modifications, besides the addition of timers, to the benchmark code, it takes under 10 seconds to produce all of the images demonstrating the object detection and the grasping pose for the singular cylinder code. The majority of the time taken to detect the objects and the potential grasping poses is simply just due to the speed at which the user can close the previous image. Without modification, the benchmark code produces 10 images demonstrating what the system has perceived and detected to that point. These are shown, for the single cylinder detection, in the figures below, and presented in the same order as they are when presented by the simulation, the scale of the images has been maintained to give a cohesive demonstration of the output.

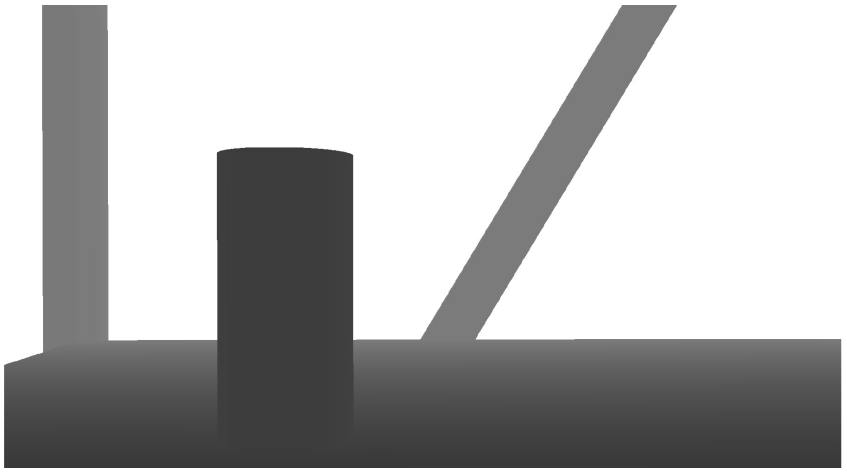


Figure 5: 1st image generated during the single cylinder object detection phase, displayed in black and white

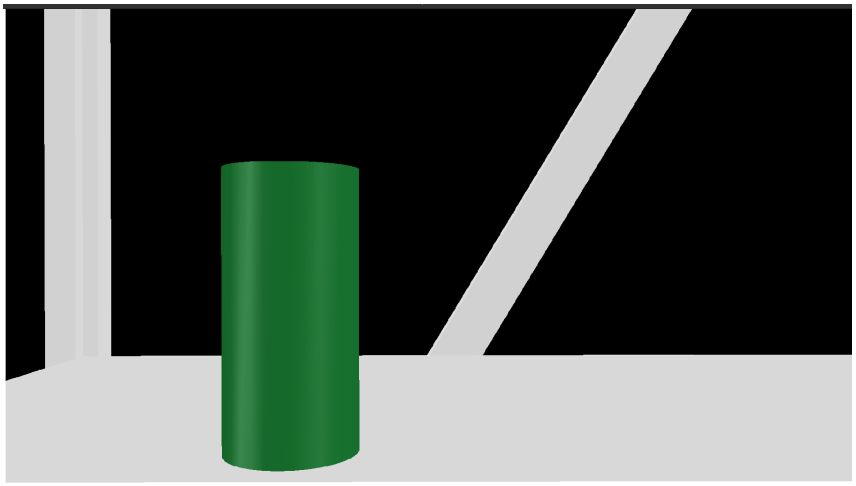


Figure 6: 2nd image generated during the single cylinder object detection phase, with the addition of colour to demonstrate where the object has been perceived to be.

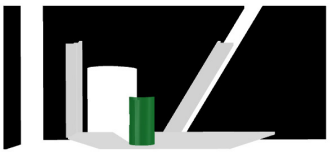


Figure 7: 3rd image generated during the single cylinder object detection phase, with colour and shadows being identified

Figure 8: 4th image generated during the single cylinder object detection phase, with the addition of colour to demonstrate where the object has been perceived to be.

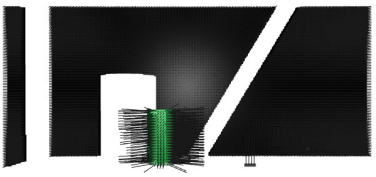


Figure 9: 5th image generated during the single cylinder computer vision method. The beginnings of the point cloud generation to give suggestions for the grasping regions.

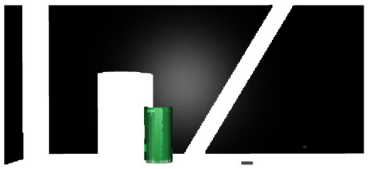


Figure 10: 6th image generated during the single cylinder computer vision method.

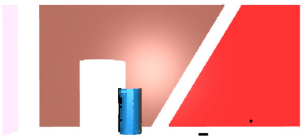


Figure 11: 7th image generated during the single cylinder computer vision aspect. A recolouring of the perceived environment making use of the depth information.

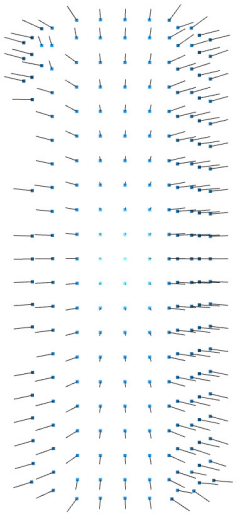


Figure 12: 8th image generated during the single cylinder computer vision aspect. The point cloud of the cylinder.

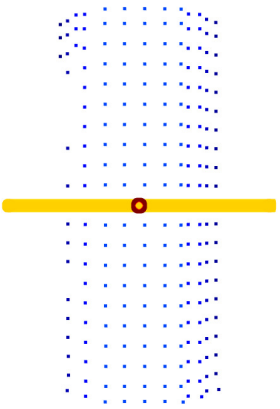


Figure 13: 9th image generated during the single cylinder computer vision aspect. The point cloud for the cylinder with the proposed grasping region.

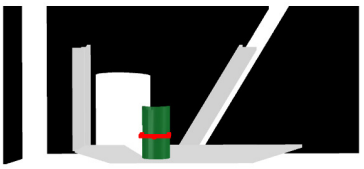


Figure 14: Final image generated during the single cylinder computer vision stage. The grasping region displayed on the cylinder in the scene.

Each image is displayed in turn but only after the previous image has been closed can the next image be generated. Once the object detection images and point cloud have been generated, the grasping pose image is generated, then the route planning begins. In general, the route planning aspect takes approximately 4 minutes 56 seconds - this is observed as being the point from when the final image is closed to the point when the arm begins moving. Completing the pick-and-place motion aspect of the simulation takes approximately 15 additional seconds for the single cylinder code.

For the multiple cylinders, the (unmodified except for the addition of timers) benchmark code is far slower in every aspect. For the object detection image generation, at the beginning of the simulation all 5 cylinders are detected and these images are displayed in turn. This takes up to approximately 39 seconds for all object detection images, point clouds, and the grasping pose of the cylinders to be generated. The multiple cylinders code not only detects the first cylinder, but all of the cylinders on the shelf that it can see and generates the point clouds for each of them that it can see on each run. The images generated for the multiple cylinders, using an adapted environment with just 3 cylinders are shown below.

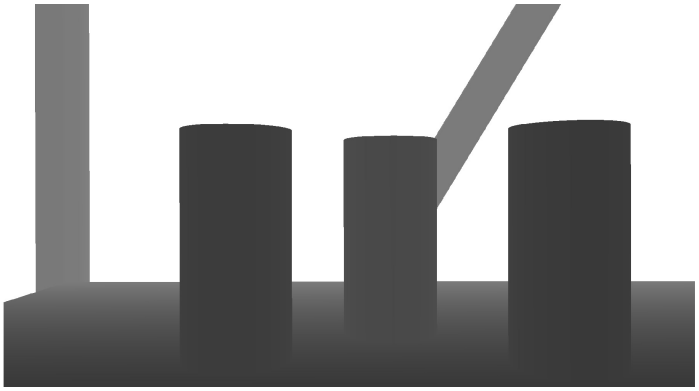


Figure 15: the 1st image generated during the multiple cylinder computer vision stage, using 3 cylinders within the scene, in black and white as it is the first stage in the vision method.

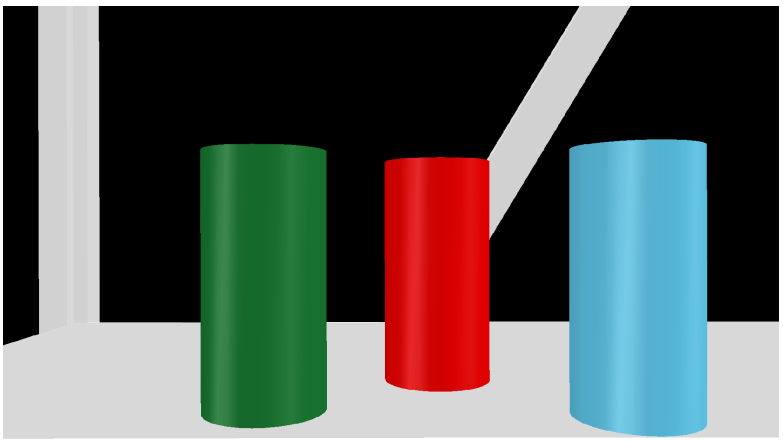


Figure 16: the 2nd image generated during the multiple cylinder computer vision stage, using 3 cylinders within the scene, in colour now as the cylinders have been identified as the target objects.

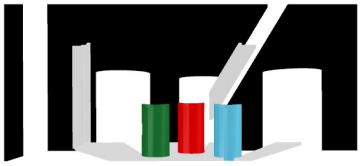


Figure 17: the 3rd image generated during the multiple cylinder computer vision stage, using 3 cylinders within the scene, in colour and with shadows being recognised.

Figure 18: the 4th image generated during the multiple cylinder computer vision stage, using 3 cylinders within the scene, in colour and with the identified shadows from the previous image having been recognised and removed as a result.

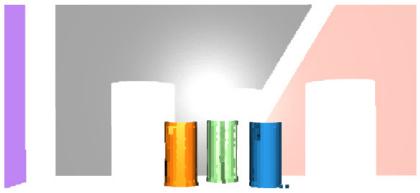


Figure 19: the 5th image generated during the multiple cylinder computer vision stage, with colour changed as a result of the depth perception process.

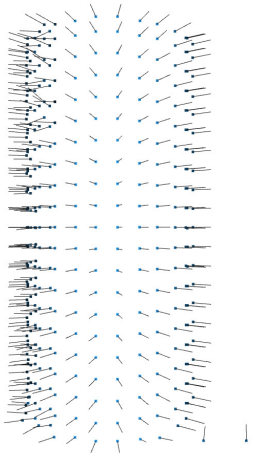


Figure 20: the 6th image generated during the multiple cylinder computer vision stage. The point cloud that is generated for the first cylinder.

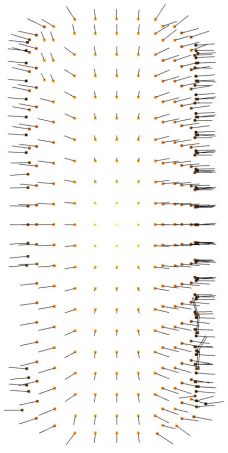


Figure 21: the 7th image generated during the multiple cylinder computer vision stage. The point cloud that is generated for the second cylinder.

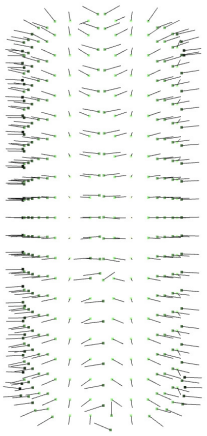


Figure 22: the 8th image generated during the multiple cylinder computer vision stage. The point cloud that is generated for the third and final detected cylinder.

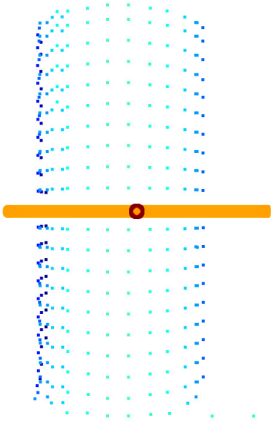


Figure 23: the 9th image generated during the multiple cylinder computer vision stage. The point cloud with the chosen grasping region for the first cylinder.

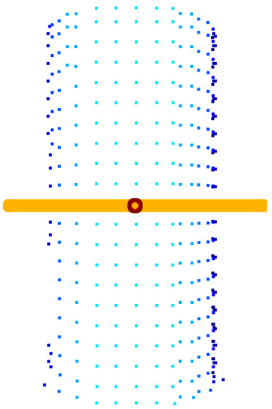


Figure 24: the 10th image generated during the multiple cylinder computer vision stage. The point cloud with the chosen grasping region for the second cylinder

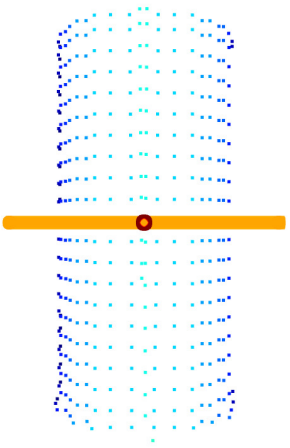


Figure 25: the 11th image generated during the multiple cylinder computer vision stage. The point cloud with the chosen grasping region for the third and final cylinder.

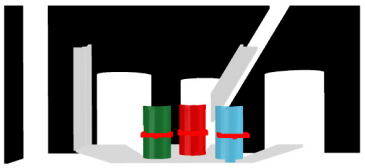


Figure 26: Final image generated during the multiple cylinder computer vision stage. The grasping region displayed on each cylinder in the scene.

The above figures are displayed on the first run through of the multiple cylinder simulation. After the first cylinder has been moved, it generates all of the above images again, but now for the 2 cylinder scene. Then again after the second cylinder has been moved, it does the same again for the now single cylinder scene.

The timers show that, as with the single cylinder code, the time consuming aspect of the program is as a result of the user having to manually close each image for the next one to be displayed, so after collecting the images, these lines of code were commented out. The route planning aspect also takes considerably longer when multiple cylinders are involved; the source code originally had 600 trials for both a single cylinder and the multiple cylinders, but this was adjusted to 100 trials to reduce the overall execution time.

The initial scene that had come with the source code for multiple cylinders displayed 5 cylinders in the workspace, however, this resulted in the benchmark solution taking

approximately 1 hour 40 minutes on average to complete the simulation. As this was a very time consuming process and used a huge chunk of CPU, meaning the simulation device could not be used for other purposes during the simulation, the scene was adjusted to contain only 3 cylinders. This meant that, although the benchmarking would still take approximately an hour for each simulation, it was less arduous to collect the data than it had been. The multiple cylinder benchmark code for generating the paths took a minimum of 15 minutes 49 seconds for each cylinder (this was the time from the generation of the grasping pose of the second cylinder to the initiation of movement of the arm), and a maximum of approximately 23 minutes 41 seconds (as was the case for the route planning for the penultimate cylinder).

Numerical Analysis of Results

After making the improvements to the code, average run times for each process within all 4 variations (the singular and multiple cylinder improvements, and the singular and multiple cylinder benchmarks) were created.

For both single cylinder variations, the code was run 20 times and the object detection, grasp pose detection, route planning, and motion execution times were all recorded. From these, average times were taken and the variance and standard deviations were additionally calculated. These are shown in the figure below.

Time (s)	**ml_tesispy_one**	**tesispy_one**
Object Detection	1.398828375	1.286957264
Grasp Pose Detection	1.565301383	1.521572721
Route Planning	74.39954064	313.90678
Motion Execution	8.936155033	15.73770052
Total	86.29982543	332.4530105

Figure 27: Average execution times for the single cylinder codes, ml_tesispy_one referring to the improved version and tesispy_one referring to the benchmark.

	ml_tesispy_one		**tesispy_one**	
Time (s)	variance	standard deviation	variance	standard deviation
Object Detection	0.006035665902	0.07768954822	0.007319682494	0.085555143
Grasp Pose Detection	0.02086672571	0.1444531956	0.03538062889	0.188097392
Route Planning	24.08021662	4.90715973	21.94155309	4.684181155
Motion Execution	0.2073230273	0.4553273847	0.04233217127	0.2057478342
Total	26.45719692	5.143655987	18365.48949	135.5193325

Figure 28: standard deviation and variances of the single cylinder codes.

The above tables show that the object detection times for both codes are very similar, with only marginal differences in the timings, variances, and standard deviations. The benchmark is marginally faster but as the code for this aspect was not adjusted, this is likely due to external factors, for instance, additional processes being run on the machine during simulation, rather than anything else. Grasp pose detection takes marginally longer for both codes, again with minimal differences in standard deviation and variance as well. The benchmark again is marginally faster, but this is likely due to the adjustment of the orientation algorithm in the improved version to account for 's' being 0. In terms of the route planning aspect, again the standard deviations and variances are very similar for both codes, but the average run time is massively different. The improved code, on average, is almost 4 times faster than the benchmark solution for this. By reducing the number of trials, the processing time was massively reduced but the number of trials was still high enough that the likelihood of finding a correct and feasible path was still very high, and this code had the same high success rate as the benchmark. This success rate for the 20 tests for both codes was 100%. The velocity of the Franka arm was also adjusted which meant that the motion execution time for the new code was also almost half that of the benchmark solution.

For the multiple cylinder variations, the simulations were run 9 times each. Despite this being far fewer simulations than for the singular cylinder versions, each run produced at least 3 lots of timings for object detection, grasping pose detection, route planning, and motion execution as these processes were replicated for each individual cylinder.

The average timings for these are demonstrated in the figure below.

Time (s)	**ml_tesispy_multiple**	**tesispy_multiple**
Object Detection	1.443466487	1.363337114
Grasp Pose Detection	2.61302374	1.642617877
Route Planning	298.0859274	1019.711962
Motion Execution	14.90736802	16.49558534
Total	318.2266817	1013.389616

Figure 29: Average execution time for multiple cylinders, with ml_tesispy_multiple referring to the new implementation, and tesispy_multiple referring to the benchmark version.

Again, the standard deviations and variances for this data were generated and are shown in the figure below.

	ml_tesispy_multiple		tesispy_multiple	
Time (s)	variance	standard deviation	variance	standard deviation
Object Detection	0.06839781211	0.2615297538	0.03258939278	0.1805253245
Grasp Pose Detection	2.889400946	1.699823799	0.03820036409	0.1954491343
Route Planning	4142.449999	64.36186758	70084.58185	264.7349275
Motion Execution	213.3308637	14.60585033	4.002184597	2.000546075
Total	4651.222466	68.19987145	109183.5415	330.429329

Figure 30: standard deviation and variances of the multiple cylinder codes.

The above figures, similarly to the single cylinder versions, show that the object detection and grasping pose detection are very similar, with the unedited benchmark being marginally faster for both. Again, the variances and standard deviation for these are fairly small, although for the grasping pose detection for the modified code these are a bit higher - likely due to small outliers in the data. The route planning timings above also show a clear improvement in operation time between the two versions, with the modified version on average being approximately 3.42 times faster. The velocity of the Franka Arm was not modified even for the modified version and so the execution times are relatively similar again.

Strengths and Weaknesses of the Solution

There are clear improvements in the route planning stage of the simulations, this is a definite improvement - particularly for operations on a larger scale such as with more cylinders as the simulation overall takes approximately 15 minutes for the modified version to move all 3 cylinders, as opposed to about an hour. The solution also does some mild error handling which means that the simulation is less likely to crash and end before completion.

Despite the definite improvement, the goals of the project were to implement machine learning techniques to make these improvements and this was not done.

Future Work

In terms of future work as a result of this project, there are many additional routes that could be taken to further develop this. Many of my suggestions for future work stem from both where I have seen opportunities in my own project and in the background research portion of the project.

This project's focus was on simulation using a Franka Emika Robotic Arm but could be applied to other robotic arms with 6DOF. Further work could be done to adapt this for robotic devices with either far fewer or far more degrees of freedom. Additionally, this project does not use machine learning for the route planning aspect. Thus, another route for further work would be to incorporate deep learning algorithms into the route planning process which could massively reduce the planning time required. Yet another potential pathway for further work from this project would be to have a more robust system for dealing with multiple cylinders. The code used in this project refers to the objects it wishes to handle directly by name, meaning that in scenes with a greater or smaller number of objects to move, errors can be thrown if this is not accounted for. Therefore, a suggestion for an improvement would be to detect what the objects are and generate their references within the code rather than having them hard-coded in.

Related to the previous suggestions, further extensions in this area could be training the arms to be able to pick up non-cylindrical objects, both regular and irregular shapes. This would be a useful extension as it would increase the range of items that a robotic arm would be able to pick- and-place. In addition to this, the ability to identify grasping poses for any novel objects could be a beneficial improvement to the system. This would mean that an additional dataset would be required for training the system, but would also massively increase the potential functionality of this as a pick-and-place system. It could also increase the number of potential uses for the system, for instance, in warehouses packaging obscurely shaped objects, or for pill packaging.

Other suggestions for this work could depend on the purpose of the project, for example, if the priority of the project is simply to move an object from one location to another, without much care for it being the shortest path, this could easily be adapted to simply use the first feasible path. Alternatively, learning the number of trials generally required for the best paths and accounting for this so that it is not simulating an excessive number of unnecessary paths.

Conclusions

The aims of the project were to improve upon the original source material to reduce the execution time of a simulation program for detecting objects, their grasping regions, and the shortest path to take to pick-and-place them elsewhere, ideally through the use of machine learning algorithms.

At the beginning of this project, it was believed that the computer vision and grasping poses aspects of the code were time consuming and inefficient, so machine learning algorithms should be implemented to improve the timings of these elements. However, after thoroughly examining the code it was discovered that these processes already utilised fairly efficient machine learning algorithms - particularly through the open source software, open3d. By adding in timers and commenting out the visualisations, it was found that these processes averaged at about 1.3-1.6 seconds for each individual run of the object detection process. Unfortunately, this discovery was only made very late into the project as it had been understood that the main goals of the project were to optimise these processes through the addition of machine learning. Because of this, I had not properly investigated the source code until I thought I was nearly ready to put in some machine learning that I had been working on in an external file.

The implementation of new machine learning algorithms caused some issues within the code for several reasons. For example, datasets for the sorts of objects that the project is focussing on are not very easy to find and they often are not labelled. Additionally, the main area of the code that new machine learning algorithms would have had the most impact would have been in the motion planning stage, however this was a slight problem as my experience in robotics was non-existent prior to this project. If there had been fewer issues with the hardware I was operating with, I could have made a better attempt to deal with this, however, given the timescale and how close to the end it began to fail it was not feasible for me.

Overall, although the project aimed to improve the runtime of the source code through machine learning, it was found that, as it already utilised machine learning, that other elements could be adjusted instead and it would remain as accurate but would just be faster. The project did succeed in speeding up the execution of the simulations, however further work would be required to add machine learning in to the motion planning phase.

Reflection on Learning

Through completing this project I have developed a much greater understanding of how machine learning works and of the vast range of different algorithms there are for these. Additionally, of how these different algorithmic designs are more suited to different purposes.

The project also really helped with my development of personal time management skills and reinforced the importance to me of being consistent with work effort for a long term project such as this. I found that if I had long rest periods between working on the actual project, rather than just working on the report, that when I returned to the code aspect, it took much longer and I was repeating the same sorts of mistakes that I'd previously made. This really demonstrated the importance of consistency with working on all aspects of the project at a time.

I have also developed my resilience through this project as there were many points where nothing seemed to be working or Ubuntu was being difficult, these could have been massively more disheartening than I found them as I was able to problem solve and figure out the best way forward so that I lost as little progress as possible. Furthermore, I am now very adept at installing dual boot software and working with Ubuntu, skills that will absolutely be useful in a career in computer science.

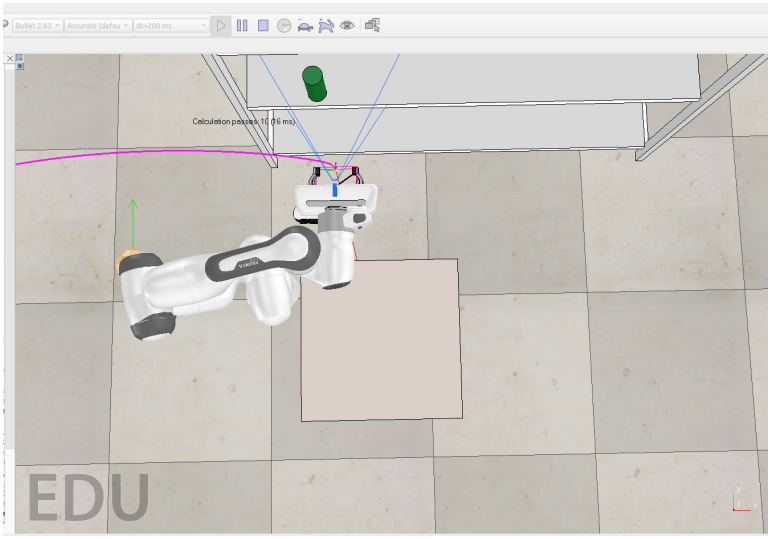
This project also gave me many transferable skills, and a greater appreciation of commenting code to a high standard to ensure that others can easily figure out what is going on so they could also work on it without too much difficulty.

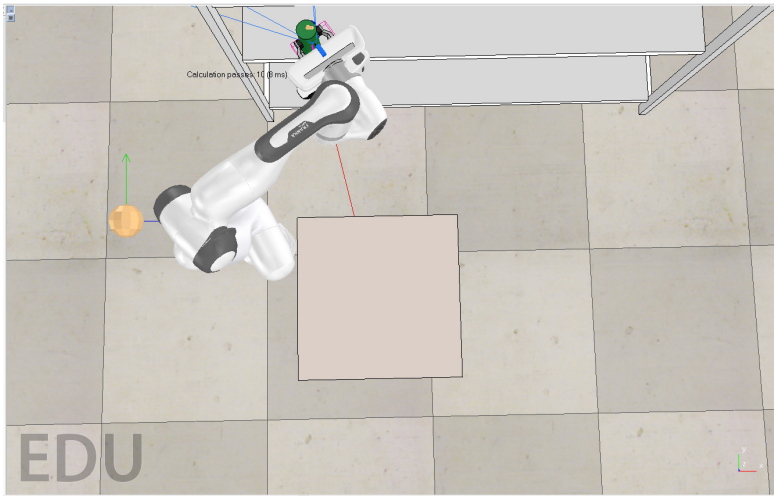
Table of Abbreviations

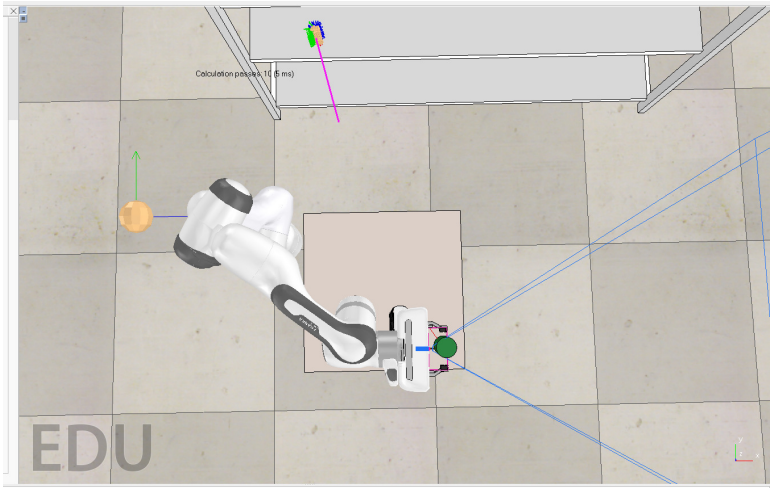
CNN - Convolutional Neural Network
ROS - Robot Operating System
GOMP - Grasp-Optimised Motion Planner
GQ-CNN grasp quality convolutional neural networks
DOF - Degrees of Freedom (references number of points of movement particularly in robotic arms)

Appendices

Below are some examples of the simulation for moving a single cylinder. Additionally, uploaded onto pats is a recording of one simulation of the improved single cylinder simulation.







References

Amazon Robotics Challenge (2017). Amazon Picking Challenge - Robohub, Connecting the robotics community to the world. [online] Available at: https://robohub.org/tag/amazon-picking-challenge/ [Accessed 5 Mar. 2023].

Arulkumaran, K., Deisenroth, M.P., Brundage, M. and Bharath, A.A. 2017. Deep Reinforcement Learning: A Brief Survey. *IEEE Signal Processing Magazine* 34(6), pp. 26–38. Available at: https://ieeexplore.ieee.org/abstract/document/8103164 [Accessed: 11 May 2023].

Breyer, M., Jen Jen Chung, Ott, L., Siegwart, R. and Nieto, J. (2021). Volumetric Grasping Network: Real-time 6 DOF Grasp Detection in Clutter. PMLR, [online] pp.1602–1611. Available at: https://proceedings.mlr.press/v155/breyer21a.html [Accessed 28 Feb. 2023].Charles Ruizhongtai Qi, Yi, L., Su, H. and Guibas, L.J. (2017).

PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. [online] ResearchGate. Available at: https://www.researchgate.net/publication/317426798_PointNet_Deep_Hierarchical_Feature_Learning_on_Point_Sets_in_a_Metric_Space [Accessed 20 Mar. 2023].de, T. (2021).

Pose Estimation. [online] Kaggle.com. Available at: https://www.kaggle.com/datasets/itiresearch/pose-estimation?select=cylinder [Accessed 11 Apr. 2023].

Del, O., Guardiola, J.-L., Javier Pérez Soler and Juan-Carlos Perez-Cortes (2021). Probabilistic Pose Estimation from Multiple Hypotheses. [online] ResearchGate. Available at: https://www.researchgate.net/publication/354543405_Probabilistic_Pose_Estimation_from_Multiple_Hypotheses [Accessed 11 Apr. 2023].

Fang, H.-S., Wang, C., Gou, M. and Lu, C. (2020). GraspNet-1Billion: A Large-Scale Benchmark for General Object Grasping. Thecvf.com, [online] pp.11444–11453. Available at: https://openaccess.thecvf.com/content_CVPR_2020/html/Fang_GraspNet-1Billion_A_Large-

Scale_Benchmark_for_General_Object_Grasping_CVPR_2020_paper.html [Accessed 28 Feb. 2023].

Ekin Keserer 2021. Deep Learning vs Reinforcement Learning: Key Differences and Use Cases. Available at: https://www.akkio.com/post/deep-learning-vs-reinforcement-learning-key-differences-and-use-cases#:~:text=Deep%20learning%20is%20a%20method%20of%20machine%20learning%20that%20enables,as%20to%20maximize%20a%20reward. [Accessed: 11 May 2023].

Hu, J. and Pagilla, P.R. (2022). View Planning for Object Pose Estimation Using Point Clouds: An Active Robot Perception Approach. IEEE Robotics and Automation Letters, [online] 7(4), pp.9248–9255. doi:https://doi.org/10.1109/lra.2022.3189821 [Accessed 14 Mar. 2023]

Ichnowski, J., Danielczuk, M., Xu, J., Satish, V. and Goldberg, K. (2020). GOMP: Grasp-Optimized Motion Planning for Bin Picking. [online] 2020 IEEE International Conference on Robotics and Automation (ICRA). Available at: https://www.semanticscholar.org/paper/GOMP%3A-Grasp-Optimized-Motion-Planning-for-Bin-Ichnowski-Danielczuk/619bfa271d90b7a3a216bebc74e8a78c76fcacf4 [Accessed 14 Mar. 2023].

Jiang, D., Wang, H., Chen, W. and Wu, R. (2016). A novel occlusion-free active recognition algorithm for objects in clutter. 2016 IEEE International Conference on Robotics and Biomimetics (ROBIO). [online] doi:https://doi.org/10.1109/robio.2016.7866521 [Accessed 28 Feb. 2023].

Kasaei, H. and Kasaei, M. (2023). MVGrasp: Real-time multi-view 3D object grasping in highly cluttered environments. Robotics and Autonomous Systems, [online] 160, p.104313. doi:https://doi.org/10.1016/j.robot.2022.104313 [Accessed 14 Mar. 2023].

Mahler, J., Liang, J., Niyaz, S., Laskey, M., Doan, R., Liu, X., Ojea, J.A. and Goldberg, K. (2017). Dex-Net 2.0: Deep Learning to Plan Robust Grasps with Synthetic Point Clouds and Analytic Grasp Metrics. arXiv.org. [online] doi:https://doi.org/10.48550/arXiv.1703.09312 [Accessed 14 Mar. 2023].

Marr, B. 2021. Artificial Intelligence: What's The Difference Between Deep Learning And Reinforcement Learning? *Forbes* 10 December. Available at: https://www.forbes.com/sites/bernardmarr/2018/10/22/artificial-intelligence-whats-the-difference-between-deep-learning-and-reinforcement-learning/?sh=32230729271e [Accessed: 11 May 2023].

Niroui, F., Zhang, K., Kashino, Z. and Nejat, G. 2019. Deep Reinforcement Learning Robot for Search and Rescue Applications: Exploration in Unknown Cluttered Environments. *IEEE Robotics and Automation Letters* 4(2), pp. 610–617. Available at: https://ieeexplore.ieee.org/abstract/document/8606991 [Accessed: 11 May 2023].

Pfeiffer, M., Schaeuble, M., Nieto, J., Siegwart, R. and Cadena, C. (2017). From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots. 2017 IEEE International Conference on Robotics and Automation (ICRA). [online] doi:https://doi.org/10.1109/icra.2017.7989182 [Accessed 14 Mar. 2023].

Son, D. (2022). Grasping as Inference: Reactive Grasping in Heavily Cluttered Environment. IEEE Robotics and Automation Letters, [online] 7(3), pp.7193–7200. doi:https://doi.org/10.1109/lra.2022.3181735 [Accessed 14 Mar. 2023].

Song, S., Zeng, A., Lee, J. and Funkhouser, T. (2020). Grasping in the Wild: Learning 6DoF Closed-Loop Grasping From Low-Cost Demonstrations. IEEE Robotics and Automation Letters, [online] 5(3), pp.4978–4985. doi:https://doi.org/10.1109/lra.2020.3004787 [Accessed 14 Mar. 2023].

Upton, M. (2021). Toyota Human Support Robot: What is it and how can it be used? - Toyota UK Magazine. [online] Toyota UK Magazine. Available at: https://mag.toyota.co.uk/toyota-human-support-robot/ [Accessed 7 Mar. 2023].

Wei, W. (2022). Improving grasp execution times by incorporating motion planning information. MSc Thesis. [Accessed 27 Jan. 2023]

Zeng, A., Song, S., Yu, K.-T., Donlon, E., Hogan, F.R., Bauza, M., Ma, D., Taylor, O., Liu, M., Romo, E., Fazeli, N., Alet, F., Chavan Dafle, N., Holladay, R., Morona, I., Nair, P.Q., Green, D., Taylor, I., Liu, W. and Funkhouser, T. (2019). Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. The International Journal of Robotics Research, [online] 41(7), pp.690–705. doi:https://doi.org/10.1177/0278364919868017 . [Accessed 7 Mar. 2023]

Bibliography

James, S., Freese, M. and Davison, A.J. 2019. PyRep: Bringing V-REP to Deep Robot Learning. Available at: https://www.researchgate.net/publication/334049230_PyRep_Bringing_V-REP_to_Deep_Robot_Learning [Accessed: 11 May 2023].

Chen, Y.-L., Cai, Y.-R. and Cheng, M.-Y. 2023. Vision-Based Robotic Object Grasping—A Deep Reinforcement Learning Approach. *Machines* 11(2), p. 275. Available at: https://www.mdpi.com/2075-1702/11/2/275 [Accessed: 11 May 2023].