Object Oriented Programming Team 6 Project Report

Agrawal, Rehan; Ramos Baroni, Harris

December 2024

Introduction

This project focuses on the task of generating random grasps for a robotic gripper in a simulated environment, with the goal of training a classifier to distinguish between successful and failed grasps. The process begins by generating a set of random hand poses relative to the object, where the gripper's position and orientation are varied with added noise. These grasp poses are then labelled based on their success or failure and used to train a classifier that predicts the likelihood of a successful grasp. During testing, the system generates a grasp pose, predicts its success, and evaluates the prediction by attempting to lift the object. The performance of the classifier is analysed, particularly focusing on the impact of the amount of training data used on the success of the classifier. In this report, multiple classifiers have been developed and compared comprehensively in the analyisis section.

Methods

Overview of the program design

A diagram of the overall system is provided in Figure 1. The simulation project is designed to model a robotic grasping system that incorporates multiple objects and grippers in a generic, extensible framework. The `Simulator` class acts as the core component, orchestrating the interaction between grippers, objects, and labelling mechanisms. It supports modularity by integrating gripper-specific implementations, such as `ThreeFingers` and `ShadowHand`, through a shared `FreeGripper` base class. Gripper-specific details, including URDF paths and grip motions, are stored within these subclasses to encapsulate their unique characteristics while maintaining a consistent interface for the simulation. This design enables seamless switching between grippers without modifying the core logic.

To generalise the system for different objects and grippers, the `Simulator` takes configurable parameters, such as object paths and gripper models, as input. Object-specific attributes are loaded dynamically, and the system can refresh or restart the simulation for new configurations. Grasping data, including positional and contact-point features, is collected using the `Labeler` class, ensuring consistent output regardless of the input object or gripper.

The trained classifier could be extended into a basic grasp planner. This planner would generate grasp poses based on the object's position and orientation, using the classifier to evaluate each pose. The framework is designed to be extensible, which makes it easy to incorporate such a planner. New components or adjustments can be added to improve the grasp planning process, ensuring the system remains adaptable and scalable as future improvements are made.

Simulation details

The simulator is built to simulate and evaluate robotic grasping scenarios. It uses a configurable `Simulator` class that initialises components such as grippers, objects, and the environment. For grippers, the `FreeGripper` base class provides methods for initialisation, grasping actions, and generating random positions and orientations. The gripper subclasses, such as `ThreeFingers` and `ShadowHand`, define specific kinematic and motion profiles. Object data, including shapes and placement, is passed as arguments to the simulator. The simulation workflow includes invoking methods to refresh the scene, apply random orientations, and calculate key metrics such as grasp success. Additionally, the `Labeler` class automates data collection and feature extraction, making the system scalable for different tasks and datasets.

This design ensures that the project remains modular, reusable, and easy to adapt for different grippers and objects, making it suitable for experimentation and real-world robotic applications.

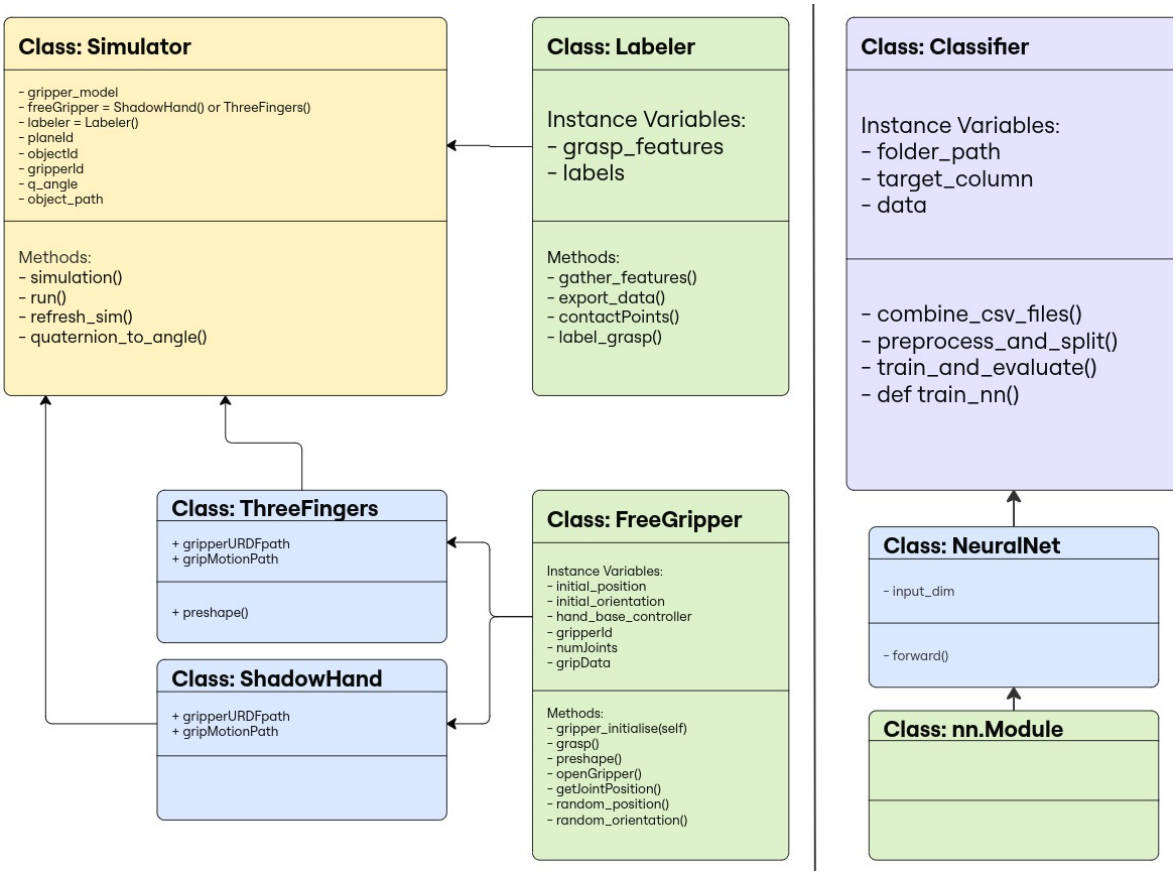


Figure 1: Gripper Simulation Flowchart

Data generation

Pose generation

Figure 2 shows arrows placed around objects which represent some of the grasps generated in the simulations. The robotic gripper poses were generated by sampling random positions on a sphere of radius `radius` centred at the object, with slight perturbations introduced by in the direction of each coordinate axis x, y and z in a specified range. Each position served as the tail of a vector pointing toward the object centre, computed and normalised as a base orientation. To simulate variability, a small random deflection within `max_deflection` was applied to the base vector using quaternion-based rotation. The result was a set of positions and deflected orientation vectors pointing approximately toward the object.

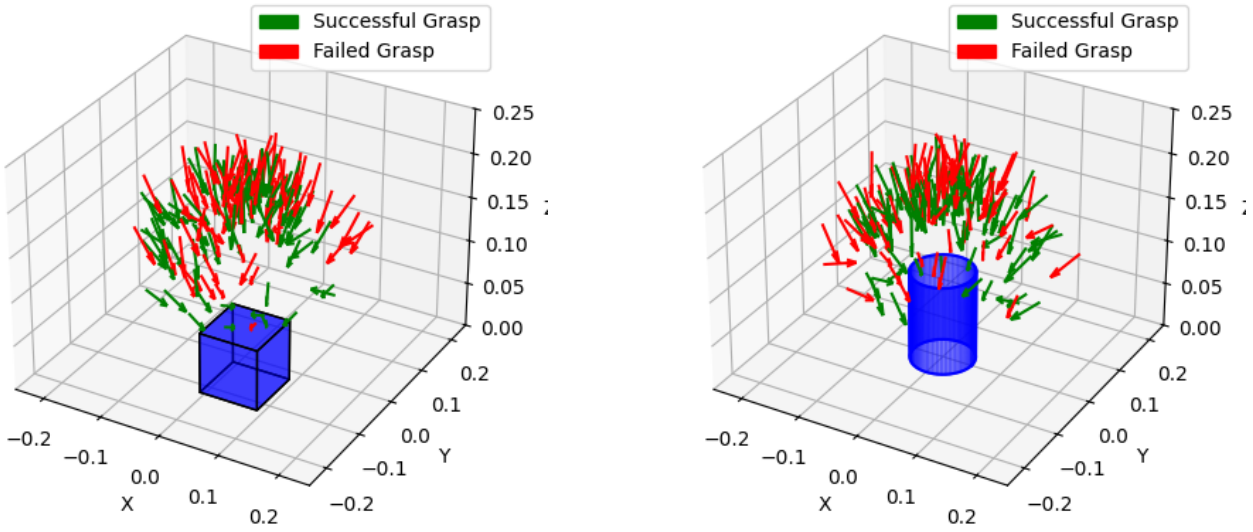
Feature Gathering

The `gather_features` method of the `Labeler` class collects features for each grasp. It computes the gripper's position and orientation relative to the object using PyBullet's `invertTransform` and `multiplyTransforms` functions. The relative distance (`gripperDist`) is included as a feature. Contact data, such as the number of contacts, positions, and forces, are extracted using `contactPoints`. If no contacts occur, penalising features like `centroid_dist` defaults to `gripperDist`. For valid contacts, the forces are summarised into mean, maximum, and total values, while the centroid of the contact points and its distance from the object centre (`centroid_dist`) provide additional grasp quality metrics.

All features were organised into a dictionary, augmented with optional keyword arguments passed via `kwargs`, and appended to the `grasp_features` attribute, a list for structured storage. In particular, this was useful to gather the `orientation deviation` feature since its computation required elements which were not necessary to pass to the `gather_features` method, so it was easiest to compute before its call and pass as an additional argument. This method showcases Python's OOP principles by combining encapsulation, modularity, and efficient data handling using libraries like NumPy and pandas, while PyBullet facilitates physics-based simulations for realistic feature extraction.

Labelling

The labelling of grasps was performed automatically by the program, rather than through manual intervention. The criterion for labelling a grasp as successful was based on whether the gripper maintained a sufficient distance from the object over a specified time threshold. If the distance between the object and a defined plane exceeded a threshold, the grasp was labelled as successful (1); otherwise, it was labelled as failed (0). This criterion is intuitively appropriate because it simulates the condition where the gripper successfully holds the object above a certain height. A potential downside is that this method may not capture all types of grasp failures, such as those due to instability or slippage, which might require additional criteria to improve accuracy. Figure 2 shows several grasps labelled in this way, the green representing the successful grasps, and the red representing the failed grasps.



(a) Grasp poses around cube object.

(b) Grasp poses around cylinder object.

Figure 2: Random grasps poses generated around objects.

Data preparation

The data preparation for this project focused on organising grasp data for training and evaluating a classifier to distinguish between successful and unsuccessful grasps. The `Labeler` class collected features from multiple simulation runs, including metrics like force applied, distance from the object, and number of contact points. Each simulation used random initial gripper states to ensure diverse data.

The data was exported as CSV files and organised into folders based on gripper pose randomness levels. This structure allowed the classifier to aggregate data by randomness level and analyse its impact on performance.

After aggregation, the data was preprocessed with a train-test split and feature standardisation to ensure equal contribution from all features. No dimensionality reduction was applied, as all features were deemed significant. This approach enabled effective classifier training and evaluation, while also analysing the effect of pose randomness on grasp prediction.

Classifier Model Description

Overview of the Classifier

The classifier presented in this work is a modular machine learning pipeline designed to train and evaluate different models incrementally using CSV datasets. The primary objective of the classifier is to predict the target variable, `success_rate`, through models such as Logistic Regression, Random Forest, and Neural Networks. The pipeline offers flexibility in preprocessing, training, and evaluation while supporting scalability to handle multiple datasets. It integrates data preprocessing, model training, and performance evaluation in a systematic manner.

Structure of the Classifier

The pipeline consists of the following core components:

- **CSV Combination Module:** Randomly selects and combines CSV files incrementally, ensuring no duplication, to create datasets of varying sizes for experimentation.
- **Preprocessing Module:** Handles splitting the data into training and testing sets while applying optional standardization to ensure consistent feature scaling.
- **Neural Network Architecture:** The classifier uses a custom-built neural network implemented in PyTorch. The architecture comprises:
  - `Input Layer`: Accepts the input features.
  - `Fully Connected Layers`: Four dense layers with ReLU activation functions for non-linearity.
  - `Dropout Layers`: Dropout regularization with a rate of 0.3 to prevent overfitting.
  - `Output Layer`: A single neuron with a Sigmoid activation function to predict probabilities for binary classification.
- **Training Module:** Supports Logistic Regression, Random Forest, and Neural Network training. For neural networks, training is performed using the Adam optimizer with a Binary Cross-Entropy loss function.
- **Evaluation Module:** Provides metrics such as accuracy, classification report, and confusion matrix for performance assessment. Iterative training results are aggregated for larger datasets.

Epoch Selection

The neural network was trained using a batch size of 8, with training conducted for up to 100 epochs. The training and validation loss trends were monitored to determine the optimal number of epochs. Figure 3 shows the training and validation loss curves, which played a crucial role in selecting the optimal epoch value.

The graph illustrates a consistent decrease in training loss, while the validation loss decreases initially but begins to increase after epoch 18. This divergence between training and validation loss is indicative of overfitting beyond this point. The range of epochs 18 to 25 was identified as a candidate for optimal training, with epoch 18 providing the best balance between training performance and generalization, as reflected by the validation loss curve. This selection was supported by observing that the model performed best on the validation set at epoch 18.

Batch Size and Learning Rate

The batch size was set to 8 to maintain a balance between training speed and model generalization. A smaller batch size allows the model to learn from more granular updates, although it can increase training time. The learning rate was set to 0.001 to ensure stable convergence while avoiding large oscillations in the loss function.

Training and Evaluation Results

The training results demonstrate the effectiveness of the classifier pipeline. By incrementally adding to the size of the dataset and analyzing model performance, the classifier achieves reliable results across various models. The aggregated classification reports and confusion matrices further validate its robustness and scalability.

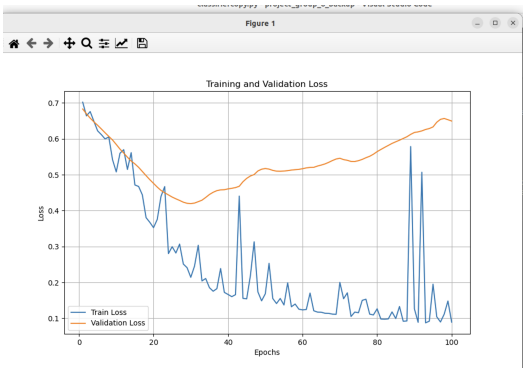


Figure 3: Training and Validation Loss Curves. Optimal epoch range is highlighted between epochs 18 and 25, with epoch 18 selected for best generalization.

Classification Model Performance

Model Performance Details

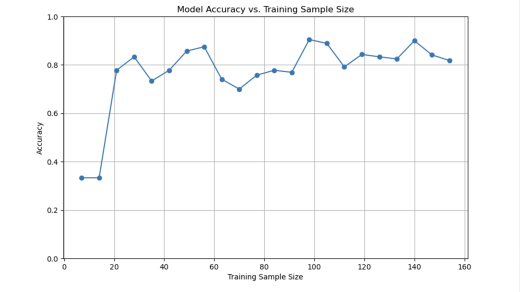
Note: Noise Level 1 is characterized by a translation range randomness in the range of 0.05 units along the x-axis, 0.05 units along the y-axis, and 0.04 units along the z-axis, combined with an orientation noise of up to 15 degrees (`max_deflection` = 15).

Noise Level 2 is defined by a translation range randomness in the range of 0.04 units along the x-axis, 0.04 units along the y-axis, and 0.04 units along the z-axis, along with an orientation noise of up to 10 degrees (`max_deflection` = 10).

The confusion Matrixes and Performance metrics are averaged from the last 1/3 sample size data. This is because our sample size becomes big enough to give consistent enough results.

Logistic Regression with Noise Level 1

Performance Metrics:



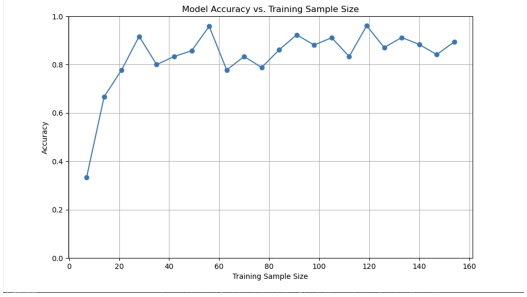
- Precision (class `0`): 0.903, Recall: 0.886, F1-score: 0.893
- Precision (class `1`): 0.728, Recall: 0.763, F1-score: 0.741
- Overall Accuracy: 84.95%

Confusion Matrix:

```
[[308  40]
 [ 34 104]]
```

Random Forest with Noise Level 1

Performance Metrics:



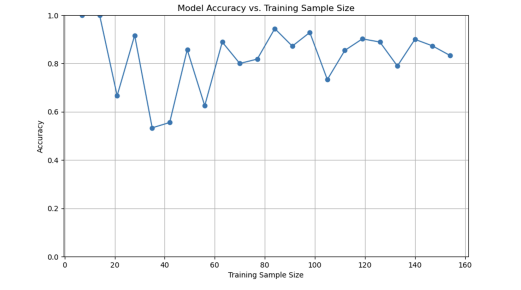
- Precision (class `0`): 0.896, Recall: 0.950, F1-score: 0.920
- Precision (class `1`): 0.884, Recall: 0.758, F1-score: 0.805
- Overall Accuracy: 88.75%

Confusion Matrix:

```
[[314  17]
 [ 38 117]]
```

Neural Network with Noise Level 1

Performance Metrics:



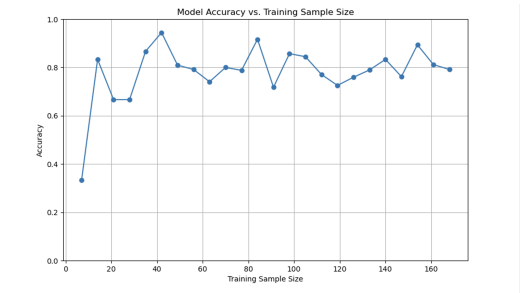
- Precision (class `0`): 0.883, Recall: 0.911, F1-score: 0.896
- Precision (class `1`): 0.795, Recall: 0.732, F1-score: 0.759
- Overall Accuracy: 85.58%

Confusion Matrix:

```
[[305  29]
 [ 41 111]]
```

Logistic Regression with Noise Level 2

Performance Metrics:

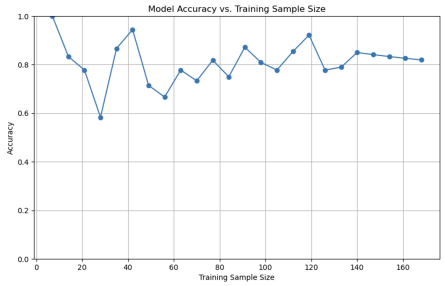


- Precision (class `0`): 0.808, Recall: 0.730, F1-score: 0.762
- Precision (class `1`): 0.787, Recall: 0.851, F1-score: 0.815
- Overall Accuracy: 79.31%

Confusion Matrix:

```
[[181  68]
 [ 42 249]]
```

Random Forest with Noise Level 2
Performance Metrics:

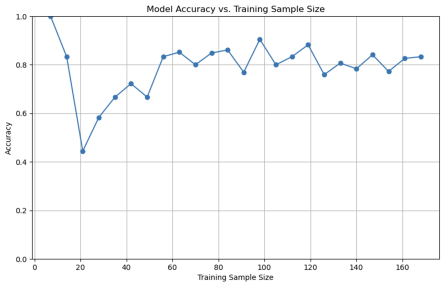


- Precision (class `0`): 0.878, Recall: 0.785, F1-score: 0.824
- Precision (class `1`): 0.808, Recall: 0.892, F1-score: 0.844
- Overall Accuracy: 83.48%

Confusion Matrix:

```
[[206  59]
 [ 31 244]]
```

Neural Network with Noise Level 2
Performance Metrics:



- Precision (class `0`): 0.785, Recall: 0.816, F1-score: 0.794
- Precision (class `1`): 0.849, Recall: 0.820, F1-score: 0.830
- Overall Accuracy: 81.54%

Confusion Matrix:

```
[[193  44]
 [ 56 247]]
```

Detailed Analysis

Random Forest Superiority:

- Random Forest consistently handles noisy data effectively, maintaining higher precision and recall across both classes compared to Logistic Regression and Neural Networks.
- Its ability to balance true positive and true negative rates highlights its robustness under varying conditions.

Logistic Regression Weaknesses:

- Logistic Regression is particularly sensitive to noise changes, resulting in lower accuracy and higher false negatives for class `1`.
- Under lower noise conditions, Logistic Regression's performance declines further, showing a lack of adaptability.

Neural Network Trade-offs:

- While Neural Networks show moderate stability, they exhibit fluctuations in accuracy, particularly under higher orientation noise.
- Their precision for class `1` remains lower than Random Forest, indicating a need for further tuning or feature engineering.

Noise Impact Analysis:

- Higher positional noise with higher orientation noise (`ori_noise_15`) results in more stable performance for all models compared to lower noise configurations.
- Models are more challenged by the second noise configuration (`pos_noise_[0.04, 0.04, 0.04]`, `ori_noise_10`), with Logistic Regression suffering the most.

Further Results

Varying Training Set Size

The graphs corresponding to this analysis are presented below under the section titled *Classification Model Performance*. For each noise level and classifier model, we observe the following trends:

Observing the figures relating to the model accuracies against the training set size, we see first of all that the general trend is for the accuracy to increase with the training set size, although this is not the case always.

We notice that when the sample size ranges between 0 and 30/40, the accuracy is either extremely low—potentially due to poor-quality data—or exceptionally high, nearing 1, which may suggest data bias. This scenario might arise from data exhibiting high variance, where instances are dispersed widely. In such cases, logistic regression and random forests tend to perform relatively well.

When the sample size increases to the range of 30 to 100, the accuracy becomes notably erratic or unstable. At this stage, accuracy fluctuates significantly, oscillating between extreme lows and highs. This instability indicates that the dataset is insufficient for achieving reliable accuracy, necessitating a larger sample size for improved precision.

Finally, when the sample size reaches the last one-third of the dataset, the accuracy stabilizes, and the fluctuations diminish. This plateau in accuracy suggests that the dataset has reached a minimum threshold, providing consistently reliable results. This stabilization serves as a positive indicator of adequate sample size for the models under consideration.

Feature importance

Figure 4 depicts how correlated some of the features were. In addition, the success was included in the plot which indicates to some extent how important that features were in the classification process. Observing the right-most column (or equivalently, the bottom-most row), the correlations with greatest magnitudes (irrespective of sign) of features with the grasp outcome indicate greatest importance.

In logistic regression, the decision variables (the coefficients) represent the change in the log-odds of the outcome for a one-unit change in the corresponding predictor, with the odds ratio e^{β_i} representing the multiplicative change in the odds of the outcome. For logistical regression classification, we can obtain these coefficents which correspond to the features and indicate the feature importance. Using a logistical regression classifier trained on 42 points and tested on 18 points, we indeed find that these coefficients are [-1.44, -0.44, 0.79, 0.40, -0.83, 0.99, -0.048], which, for the most part, clearly correspond both in sign and magnitude to the correlations in the bottom row of the figure [-0.65, -0.29, 0.56, -0.03, 0.27, 0.53, -0.07]. (These values may be obtained from the "classifier_notebook.ipynb" in the Github project.) Indeed, we find that greater relative distance of the gripper from the object corresponds the most negatively to the success rate, whereas total contacts and total force had the greatest relevance. This is evident in both the logistic regression coefficients and the correlations. Note that a limitation of this analysis is that it does not consider the greatest (nor least) relevance in any arbitrary direction in the feature space.

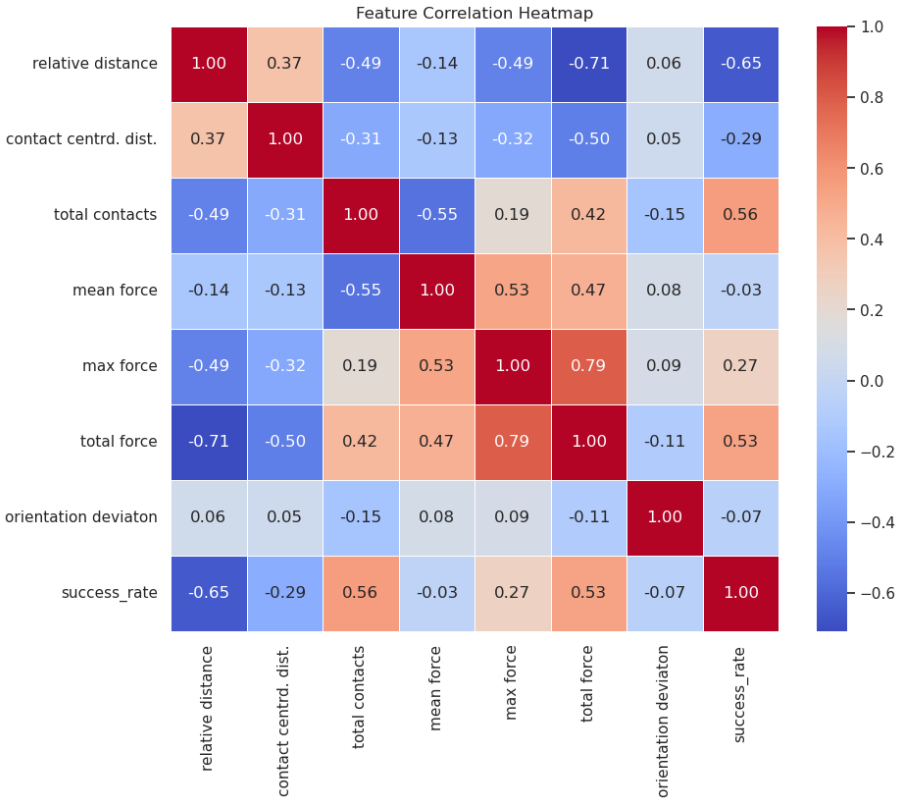


Figure 4: Feature correlation heatmap

Discussion

As discussed, a potential downside to our particular labelling criterion (checking for any object drops within a time limit) is that this method may not capture the effect of all types of grasp failures, such as those due to instability or slippage, which might require additional criteria to improve accuracy.

Another aspect which potentially have been improved is the feature selection. For example, the actual displacement of the gripper in any particular x, y or z axis was not considered in our testing and analysis (although it was perfectly possible to include it through our optional additional arguments `kwargs` to the `gather_features` method of the `Labeler`). This was under the assumption that the `gripperDist` would capture these more effectively and concisely. `gripperDist`, however, may be more limited in relevance for objects which do not exhibit rotational symmetry about a vertical axis, and it also may suffer if the angle of approach differs.

Regarding some general aspects of the scope and goals we focused on, while our program could theoretically train the classifiers based on grasp data obtained from using different grippers or objects, we mostly focused on the single ThreeFingers gripper on a cube object. Grasp performance on the cylinder was similar to the cube so its investigation was of little interest to us. We also tried loading URDFs for objects available from "kwonathan/ycb_urdfs" on Github (Kwonathan 2024) but the objects were rather difficult for the grippers to grab which meant that training data was not very clean. A similar issue occurred with the ShadowGripper where the grasping itself was difficult to achieve. Although very possible to achieve, it became less interesting for us to focus on this as it seemed like the success of the grasp itself in simulation was not exactly in the scope of the original project.

References

Kwonathan (2024). *YCB URDFs*. Accessed: December 11, 2024.