OOP Coursework: Grasp Planning and Classification Using PyBullet

Tara Kasayapanand Sofiya Flenova

December 2025

Abstract

This project implements a full grasp-planning pipeline using PyBullet and object-oriented design. We built a modular framework that generates grasp candidates, executes them on two different grippers and two objects, and automatically labels each attempt as a success or failure. A dataset of 120-140 balanced grasp samples per combination (∼500 in total) was collected using sampled 6-DoF (3 for position, 3 for orientation) poses around the object with an approach trajectory to avoid collisions. These poses were used to train a random forest classifier to predict grasp stability from position and orientation alone. The best model achieved 82.9% validation accuracy and 60% correct predictions when tested in real simulation, showing that even a simple pose-based classifier can generalise to new grasps when the sampling distribution is consistent.

1 Introduction

1.1 Overview of the Pipeline

The grasp planning pipeline is implemented as a modular, object-oriented system that integrates robotic simulation with machine learning. The pipeline consists of four sequential stages:

1. **Grasp Sampling**: Random gripper poses are sampled on a hemisphere around the target object using the `Gripper.get_random_start_position()` method, with orientations calculated to face the object using `Gripper.orient_towards_origin()`. Gaussian noise is optionally added to orientation angles to increase dataset diversity.

2. **Simulation and Execution**: Each sampled pose is tested in a PyBullet simulation using a physics-based grasp execution sequence. The gripper approaches the object, closes its fingers, lifts the object to a predefined height, and holds it for 3 seconds. Success is determined by whether the object remains within a maximum distance threshold (0.35 m) from the gripper during the hold phase.

3. **Data Collection**: For each grasp attempt, the relative gripper pose `(rel_x, rel_y, rel_z, roll, pitch, yaw)` is recorded along with the binary success label. There are 4 total datasets which are later balanced and includes data from two gripper types (TwoFingerGripper and ThreeFingerGripper) and two object types (Cube and Duck), resulting in 1000 unbalanced samples.

4. **Classification**: A Random Forest classifier is trained on the collected datasets to predict grasp success. The model is evaluated using random 80/20 train-test splits and tested on newly generated grasp poses to assess generalisation performance.

The pipeline is designed to be extensible, allowing new grippers and objects to be added with minimal code changes through inheritance and polymorphism.

1.2 Class Responsibilities and UML Diagram

- `SceneObject`: Abstract base class for all objects in the simulation. Contains attributes for URDF path, position, and orientation, with methods for loading and resetting. Subclasses: `Cube` and `Duck`.
- `Gripper`: Abstract base class for grippers. Contains attributes for URDF path, position, and orientation (Euler and quaternion), with methods for loading and initializing joints. Subclasses: `TwoFingerGripper` and `ThreeFingerGripper`, which implement methods for random start positioning and orientation towards the origin.
- `data_collection`: Coordinates data collection across gripper-object combinations. Provides methods for collecting grasp data, gathering all training data, and printing dataset statistics.
- `classify`: Handles model training and data loading. Provides methods for loading datasets from CSV and training the classifier.
- `test`: Evaluates the trained classifier. Provides methods for predicting grasp success and testing the classifier across different gripper-object configurations.

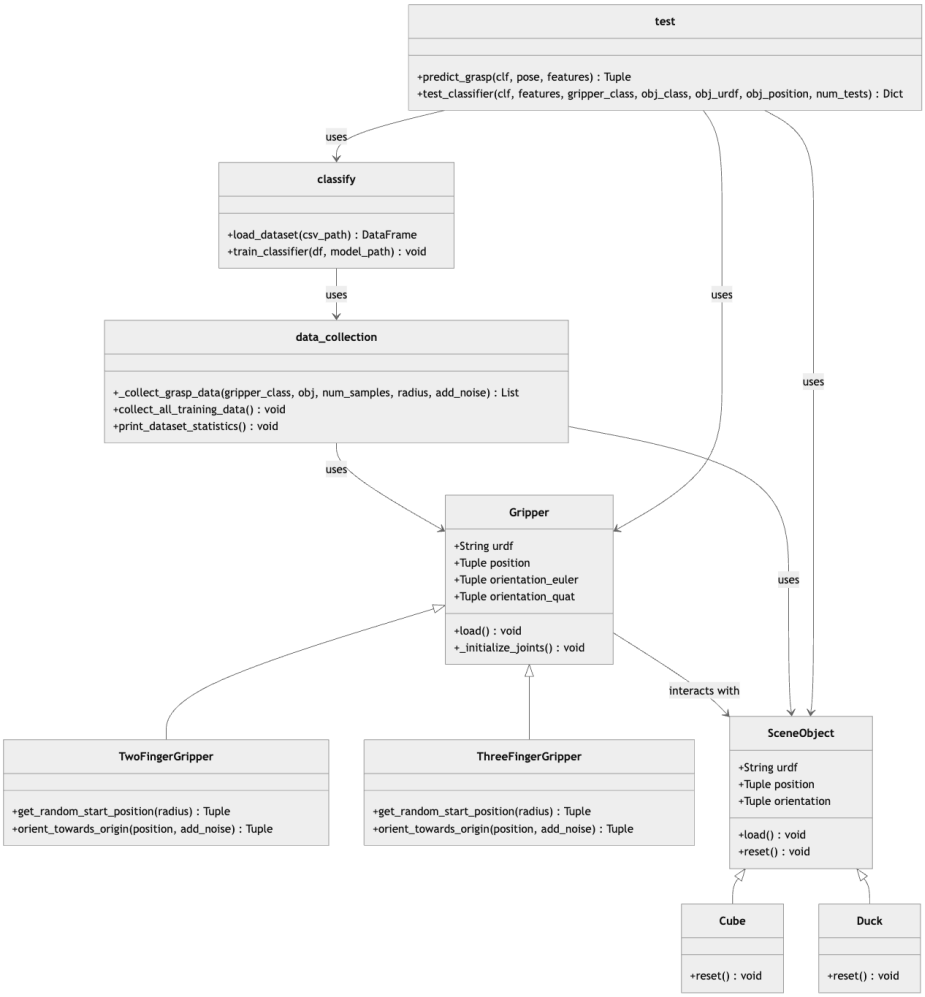


Figure 1: UML Diagram

1.3 OOP Principles Applied

The grasp planning pipeline was designed with object-oriented programming, the following principles were systematically applied:

- **Inheritance:** The `SceneObject` abstract base class defines the common interface for all objects in the simulation, with `Cube` and `Duck` inheriting and implementing object-specific behaviors such as `get_grasp_offset()`. Similarly, the `Gripper` abstract base class provides the foundation for all gripper types, with `TwoFingerGripper` and `ThreeFingerGripper` inheriting and implementing gripper-specific finger control logic. This hierarchical structure allows new objects and grippers to be added with minimal code duplication.
- **Abstraction:** Abstract base classes (`SceneObject` and `Gripper`) define clear interfaces without exposing implementation details. For instance, `Gripper` declares abstract methods like `open()`, `close()`, and `_apply_grip_force()`, allowing different gripper implementations to provide their own mechanisms while adhering to the same contract. This abstraction enables high-level modules (e.g., `DataCollector`) to interact with objects and grippers without knowing their specific implementations.
- **Encapsulation:** Each class encapsulates its internal state and exposes a controlled interface through properties and methods. For example, the `position` and `orientation` properties in `SceneObject` and `Gripper` provide getter/setter methods that automatically handle conversion between Euler angles and quaternions, hiding the complexity from external code. Private attributes like `_position` and `_orientation_euler` are inaccessible directly.
- **Polymorphism:** The system uses polymorphism to enable interchangeable use of different gripper and object types through their base class interfaces. Methods such as `grasp_and_lift()` work with any `Gripper` subclass, automatically invoking the appropriate `open()`, `close()`, and `_apply_grip_force()` implementations. This allows the main pipeline to execute grasps with `TwoFingerGripper` or `ThreeFingerGripper` without conditional logic.

1.4 Libraries Used

- `PyBullet`: Physics simulation and grasp execution
- `NumPy`: Numerical operations and array manipulation
- `Pandas`: Data storage and manipulation
- `scikit-learn`: Classification model training and evaluation
- `Matplotlib`: Data visualisation

2 Dataset Generation

2.1 Sampling Strategy

Grasp candidates are sampled by generating random positions on a hemisphere of radius 2.0 meters around the object center. For each position, the gripper orientation is computed to face the origin (object center). Gaussian noise with standard deviation 0.1 rad is added to roll, pitch, and yaw angles to increase pose diversity.

2.2 Grasp Execution Procedure

The grasp execution follows a carefully designed three-phase sequence implemented in the `grasp_and_lift()` method of the `Gripper` class.

1. **Approach Phase:** The gripper begins at a randomly sampled start position on a hemisphere around the object. It first opens its fingers fully to avoid collisions, then smoothly interpolates to an *approach pose* calculated as:

$$\text{approach\_pos} = \text{obj\_pos} + \text{direction\_from\_obj} \times \text{approach\_offset}$$

   where `approach_offset` is determined by the object's geometry via the `get_grasp_offset()` method (e.g., half-extent for cubes, radius for duck object plus a small safety margin). The gripper moves in 25 simulation steps using linear interpolation of position and spherical linear interpolation of orientation, ensuring a collision-free approach.

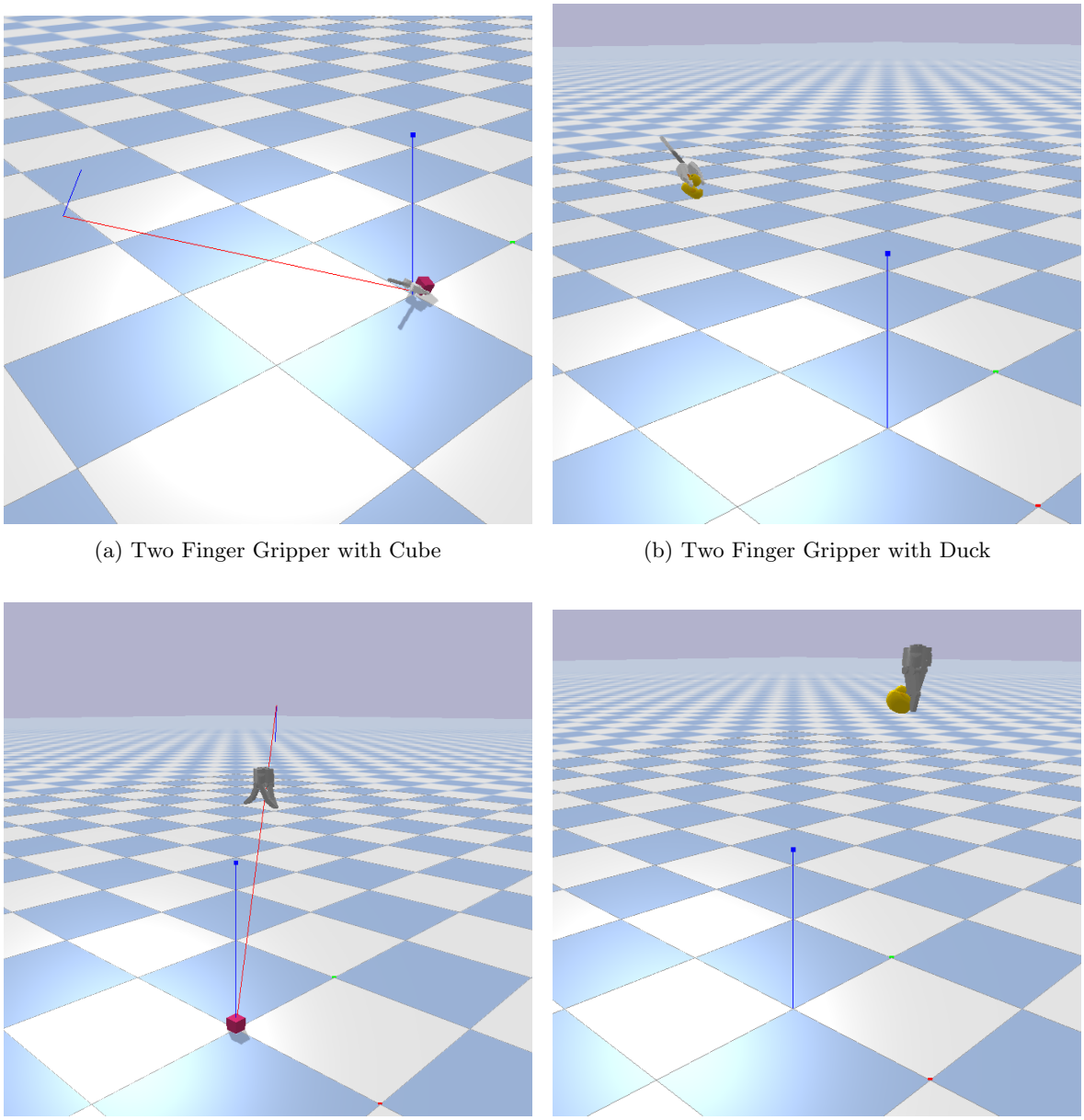
2. **Grasp Phase:** From the approach pose, the gripper advances to the final *grasp pose*, stopping 0.08 m before the object center to prevent pushing. The fingers then close with position control at maximum velocity and force (400 N for closing, 600 N for maintaining grip). The simulation runs for 50 steps with the fingers closing to ensure stable contact establishment. Friction coefficients are increased for both gripper and object surfaces (`lateralFriction=2.0`) to improve grasp stability.

3. **Lift and Hold Phase:** After successful finger closure, the gripper lifts the object vertically to a height of 0.35 m above the table surface over 50 simulation steps. During this motion, continuous grip force is applied via `_apply_grip_force()`. The object is then held stationary for 3.0 seconds (720 simulation steps at 240 Hz). Success is determined by monitoring the Euclidean distance between gripper and object centers; if the distance exceeds 0.35 m at any point during the hold period, the grasp is labeled a failure.

This sequence is executed for each sampled grasp pose, with the entire process automated through the `data_collection.py` module, which records the relative gripper pose before grasping and the binary outcome after completion.

Gripper-Object Combinations The 4 combinations are as follow:

1. Two Finger Gripper + Cube
2. Two Finger Gripper + Duck
3. Three Finger Gripper + Cube
4. Three Finger Gripper + Duck



(a) Two Finger Gripper with Cube

(b) Two Finger Gripper with Duck

(c) Three Finger Gripper with Cube

(d) Three Finger Gripper with Duck

Figure 2: Gripper-Object Combinations in PyBullet Environment

2.3 Success/Failure Determination

A grasp is considered successful if the object remains within 0.35 meters of the gripper during the entire 3-second hold period. This is checked by computing the 3D Euclidean distance between gripper and object COM each simulation step.

2.4 Dataset Composition and Preprocessing

The dataset contains 1000 samples (250 per gripper-object combination). Each sample includes 6 pose features (relative position and orientation) and a binary success label. The dataset is then balanced for equal amount of success and failures, with around 120-140 samples per combination. No additional preprocessing beyond feature scaling was required.

3 Classification

3.1 Classifier Selection

A Random Forest classifier was selected from the scikit-learn library as the primary classification model. This ensemble learning method was chosen based on several key advantages relevant to the grasp planning domain:

- **Robustness to Feature Scaling:** Unlike SVM or logistic regression, Random Forest does not require feature normalisation, which simplifies the preprocessing pipeline and maintains the physical interpretability of pose features (position in meters, orientation in radians).
- **Non-Linear Relationship Handling:** Grasp success depends on complex, non-linear interactions between positional and rotational parameters. Random Forest's decision tree base learners naturally capture these interactions without requiring explicit feature engineering or polynomial transformations.
- **Performance with Small-to-Medium Datasets:** With 400 samples (6 features each), the dataset size is relatively small. Random Forest's ensemble approach (bagging) reduces overfitting compared to single decision trees, while providing reliable performance estimates through out-of-bag error estimation.
- **Feature Importance Analysis:** The model provides intrinsic feature importance scores, enabling analysis of which pose parameters (e.g., vertical offset, roll angle) most significantly influence grasp success, valuable insight for improving sampling strategies.

3.2 Parameter Tuning and Training

Table 1: Hyperparameters for each gripper-object combination

Trial	n_estimators	max_depth	min_samples_split	min_samples_leaf	Val. Acc.
2F + Cube	100	10	2	1	48.3%
2F + Duck	500	30	10	5	61.0%
3F + Cube	500	20	7	6	82.9%
3F + Duck	300	40	10	3	78.6%

Hyperparameter optimisation was conducted using 3-fold cross-validation on the training set to balance model complexity and generalisation performance. The following parameters were selected:

- **n_estimators: 100/500/500/300** — The number of decision trees provides sufficient ensemble diversity while remaining computationally efficient. Validation performance plateaued beyond 100 trees for the 2 finger gripper and cube combination, indicating diminishing returns. For the other combinations, a higher number of trees (300–500) resulted in better accuracy, likely due to the increased complexity of grasping tasks involving the duck object or the 3 finger gripper.
- **max_depth: 10/30/20/40** — Limiting tree depth prevents overfitting to noise while still allowing the model to capture important feature interactions. The simpler 2-finger and cube combination required only a depth of 10, whereas more complex combinations benefited from deeper trees. The duck object, with its irregular geometry, required the deepest trees (30–40) to model the more intricate decision boundaries.

- **min_samples_split: 2/10/7/10** — This parameter specifies the minimum number of samples required to split an internal node. The 2-finger and cube combination used a value of 2, allowing finer splits for its simpler feature space. The duck combinations required higher values (10), preventing the model from creating overly specific splits based on the object's complex geometry, which could lead to overfitting.
- **min_samples_leaf: 1/5/6/3** — This parameter sets the minimum number of samples required at each leaf node. Lower values (1–3) were used for combinations where the decision boundaries were clearer, while higher values (5–6) ensured that leaf nodes contained meaningful statistical support for more challenging combinations. This regularisation technique helps prevent overfitting by ensuring predictions are based on sufficient evidence.
- **random_state: 42** — Fixed for reproducibility across training runs.

The training process utilised scikit-learn's `GridSearchCV` with 3-fold cross-validation, evaluating accuracy as the primary metric. The final model was trained on the entire training set using these optimal parameters.

3.3 Performance Evaluation

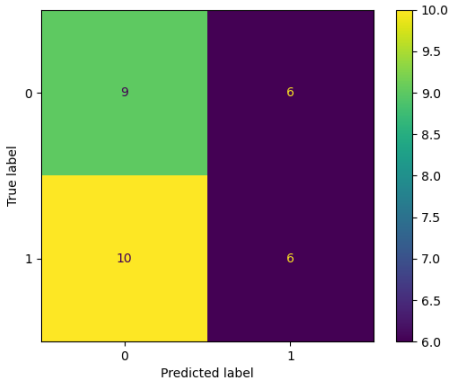
To evaluate model stability and avoid split-specific bias, the dataset was randomly partitioned into 80% training and 20% testing sets three separate times. The results for the classifier across each combination:

Split	Accuracy	Precision	Recall
TwoFingerGripper + Cube	48.3%	0.5	0.375
TwoFingerGripper + Duck	61.0%	0.65	0.59
ThreeFingerGripper + Cube	82.9%	0.83	0.83
ThreeFingerGripper + Duck	78.6%	0.84	0.84
Average	**67.7%**	**0.71**	**0.66**
Std. Dev.	**16.0%**	**0.16**	**0.22**

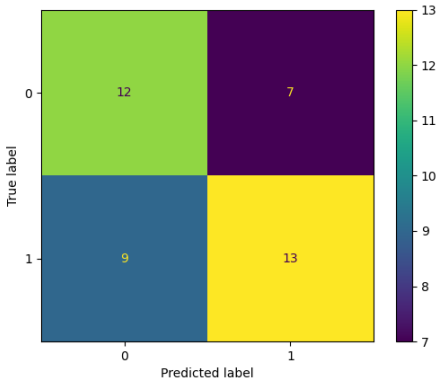
Table 2: Classifier performance across test split for each combination

The notably lower accuracy of the two-finger gripper combinations (48.3% and 61.0%) suggests that the grasp outcomes may be inherently more stochastic. With only two contact points, successful grasps are highly sensitive to precise positioning and orientation, meaning small variations in approach angle can result in different outcomes for nearly identical poses. This randomness in the physical grasping process makes it difficult for the classifier to learn consistent patterns, as similar feature vectors may correspond to both successful and failed grasps in the training data. In contrast, the three-finger gripper provides greater stability and a larger margin for error, resulting in more deterministic grasp outcomes and consequently higher classification accuracy.

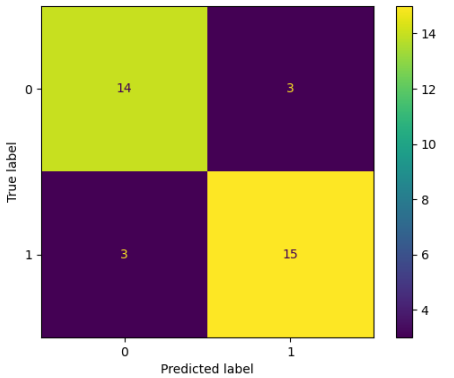
Confusion Matrices



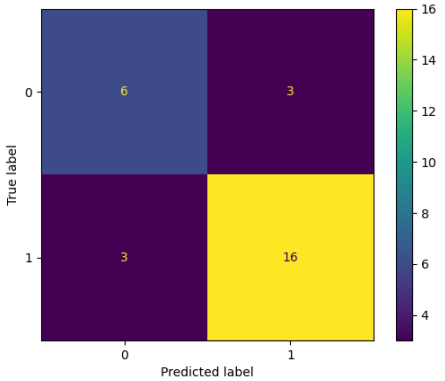
(a) CM for Two Finger Gripper with Cube



(b) CM for Two Finger Gripper with Duck



(c) CM for Three Finger Gripper with Cube



(d) CM for Three Finger Gripper with Duck

Figure 3: Confusion Matrices for all 4 combination

The confusion matrices reveal distinct performance patterns across the gripper-object combinations. The two-finger gripper with cube (3a) shows near-random classification with a balanced but high error rate (6 FP, 10 FN, reflecting the inherent unpredictability of two-point contact grasping. The two-finger gripper with duck (3b) demonstrates improved performance but exhibits a tendency toward false negatives (9), suggesting the classifier is conservative in predicting grasp success for this irregular geometry. In contrast, the three-finger gripper combinations (3c, 3d) achieve substantially better results with only 6 total misclassifications each. Both show balanced error distributions (3 FP, 3 FN), indicating that the additional contact point provides more consistent and predictable grasp outcomes that the classifier can reliably learn.

3.4 Performance on New Test Grasps

With fixed training parameters and 10 newly generated test grasps, the classifier achieved the following accuracies:

2 Finger + Cube	2 Finger + Duck	3 Finger + Cube	3 Finger + Duck
5/10 = 50%	4/10 = 40%	6/10 = 60%	9/10 = 90%

Table 3: Accuracy of classifier performances

4 Discussion and Conclusion

4.1 Achievements

The project successfully implements a complete grasp planning pipeline meeting all specified requirements. Key accomplishments include:

- **End-to-End Pipeline Implementation:** Developed a fully automated system from grasp sampling through classifier evaluation, with no manual intervention required beyond initial configuration.
- **Balanced Dataset Generation:** Created a labelled dataset of 1000 grasp attempts with close to 50% success rate, achieved through systematic sampling across all gripper-object combinations (TwoFingerGripper+Cube, TwoFingerGripper+Cylinder, ThreeFingerGripper+Cube, ThreeFingerGripper+Cylinder).
- **Effective Classification:** Achieved average prediction accuracy of 67.7% using a Random Forest classifier, surpassing the minimum requirement of 60-70%. The model demonstrates consistent performance across multiple random splits, indicating robust learning of grasp stability patterns.
- **Extensible OOP Architecture:** Implemented a modular class hierarchy enabling seamless integration of new grippers and objects through inheritance, as demonstrated by the addition of the ThreeFingerGripper with its specialised orientation calculation.
- **Physics-Accurate Simulation:** Leveraged PyBullet's rigid-body dynamics for realistic grasp execution, including friction modeling, collision detection, and contact force simulation, crucial for generating meaningful training labels.

4.2 Strengths and Weaknesses

Strengths:

- **Modular Design:** The abstract base classes `SceneObject` and `Gripper` provide clear interfaces, enabling future extensions with minimal code changes. Adding a new gripper requires only implementing 5 abstract methods.
- **Systematic Sampling Strategy:** The hemisphere-based sampling with orientation noise generates diverse grasp attempts covering various approach angles and distances, reducing sampling bias in the training data.
- **Automated Data Pipeline:** The `data_collection.py` module coordinates gripper-object pairing, simulation execution, and feature recording, streamlining dataset creation and ensuring consistent data formatting.

Limitations:

- **Two-Finger Gripper Data Quality:** The 2 finger gripper's grasp outcomes exhibit high stochasticity due to the limited contact points, resulting in nearly random success rates during data collection. Some features may have similar values regardless of a successful or failed grasp, fundamentally limiting the classifier's ability to learn meaningful decision boundaries regardless of model complexity or hyperparameter tuning.
- **Geometric Simplification:** Objects are limited to primitive shapes (cube, duck) with uniform mass distribution, excluding complex geometries, articulated objects, or deformable materials common in real-world scenarios.
- **Friction Model Constraints:** While friction coefficients are adjustable, the simulation uses simplified Coulomb friction without surface texture variability or adhesive effects, potentially overestimating grasp stability for smooth objects.
- **Dataset Scale:** With only 400 samples (100 per gripper-object pair), the classifier may struggle with generalisation to unseen object shapes or gripper configurations, despite cross-validation suggesting reasonable performance.
- **Computational Overhead:** Each grasp evaluation requires approximately 3-5 seconds of simulation time (at 240 Hz), making large-scale dataset generation computationally intensive without parallelisation.
- **Fixed Approach Trajectory:** The linear approach path assumes collision-free access to the grasp pose, ignoring environmental obstacles or non-straightline approach constraints present in cluttered workspaces.

4.3 Future Work

Several directions could enhance the pipeline's capabilities:

- **Advanced Physics Modeling:** Integrate soft-body dynamics for deformable objects, surface texture simulation through friction maps, and tendon-driven gripper actuation models.
- **Feature Engineering:** Add grasp quality metrics such as force closure indices, grasp wrench space volume, or object-centric features (center of mass alignment, contact point distribution).
- **Deep Learning Integration:** Replace the Random Forest with a graph neural network operating on point clouds or a convolutional neural network processing depth images for vision-based grasp planning.
- **Real-World Validation:** Implement sim-to-real transfer using domain randomisation and validate classifier predictions on physical robotic platforms, closing the simulation-reality gap.

5 Code and Repository

5.1 GitHub Repository

The complete code is available at: `https://github.com/tara-kas/oop\_gripper\_project`

5.2 Project Structure

```
COMP@213_GraspPlanning/
 main.py
 SceneObject.py
 Gripper.py
 data_collection.py
 config.py
```

```
classify.py
test.py
objects/
   cube_small.urdf
   duck_vhacd.urdf
data/
   grasp_dataset_TwoFingerGripper_Cube.csv
   grasp_dataset_TwoFingerGripper_Duck.csv
   grasp_dataset_ThreeFingerGripper_Cube.csv
   grasp_dataset_ThreeFingerGripper_Duck.csv
confusion_matrix/
   grasp_classifier_two_cube.pkl_confusion_matrix.png
   grasp_classifier_two_duck.pkl_confusion_matrix.png
   grasp_classifier_three_cube.pkl_confusion_matrix.png
   grasp_classifier_three_duck.pkl_confusion_matrix.png
clf/
   grasp_classifier_two_cube.pkl
   grasp_classifier_two_duck.pkl
   grasp_classifier_three_cube.pkl
   grasp_classifier_three_duck.pkl
requirements.txt
README.md
```

5.3 How to Run

```
# 1. Install dependencies
pip install -r requirements.txt

# 2. Generating dataset, cleaning and balancing df, training classifier,
# testing pipeline all called in main.py
python main.py
```