Machine Learning for Robot Safety:

Protective Stops and Grip Loss Detection

Gurshan Singh
Student of Engineering Environment and Computing
Coventry University
Coventry, England
singhg220@coventry.ac.uk

Abstract— This study explores how machine learning can predict protective stops and grip loss in industrial robots. By analysing sensor data and applying algorithms such as Support Vector Machines (SVM), K-Nearest Neighbours (KNN), and Decision Trees, the study aims to improve predictive maintenance and operational efficiency. The research covers essential data preprocessing techniques, including handling missing values, feature scaling, and encoding categorical data. The models were evaluated based on accuracy, F1-score, confusion matrices, and ROC curves. The results indicate that the Decision Tree algorithm performed best, achieving the highest accuracy in predicting operational failures. This research highlights the importance of predictive modelling in minimizing downtime and increasing robot reliability in industrial settings.

Keywords: Machine Learning, Robotics, Predictive Maintenance, Protective Stops, Grip Loss, Decision Trees, Support Vector Machines, K-Nearest Neighbours.

I. INTRODUCTION

With the rise of collaborative robots (cobots) in industrial settings, ensuring their reliability has become crucial. Robots encountering unexpected protective stops or grip failures can lead to costly downtimes and reduced productivity. Predictive models using machine learning can identify patterns in sensor data, allowing for early intervention before failures occur.

Industrial automation has significantly evolved with advancements in artificial intelligence and data analytics. Machine learning algorithms provide a powerful tool for analysing vast amounts of sensor data and detecting anomalies before they result in operational failures. Implementing predictive models not only enhances efficiency but also reduces maintenance costs by addressing issues before they escalate. This research investigates how machine learning techniques can be used to predict these operational failures. By leveraging algorithms such as SVM, KNN, and Decision Trees, we aim to determine the most effective approach for forecasting issues in robotic operations. Additionally, we address challenges like data imbalances and feature encoding to improve model performance.

Research Objectives

This study aims to:

1. Develop machine learning models capable of predicting robotic operational failures
2. Compare the performance of different classification algorithms
3. Understand the critical features contributing to protective stops and grip loss
4. Provide a comprehensive framework for predictive maintenance in collaborative robotics

II. LITERATURE REVIEW

Machine learning has revolutionized data analysis, with foundational works like Introduction to Machine Learning by Alpaydin (2014) and Pattern Recognition and Machine Learning by Bishop (2006) providing a solid theoretical basis for classification tasks. Key algorithms such as Support Vector Machines (SVM), introduced by Cortes and Vapnik (1995), and K-Nearest Neighbour (KNN), developed by Cover and Hart (1967), have become standard tools for classification. SVM

excels in handling non-linear data using kernels like the radial basis function (RBF), while KNN relies on proximity-based classification. Decision Trees, popularized by Quinlan (1986), offer interpretable models by recursively splitting data based on feature values. These algorithms, combined with evaluation metrics like ROC analysis (Fawcett, 2006) and Precision-Recall curves (Sokolova and Lapalme, 2009), form the backbone of modern classification techniques.

Handling imbalanced datasets remains a challenge, addressed by techniques such as SMOTE (Chawla et al., 2002), which generates synthetic samples for minority classes. Feature selection (Guyon and Elisseeff, 2003) and cross-validation (Kohavi, 1995) are critical for optimizing model performance and generalizability. Recent advancements in ensemble methods (Zhou, 2012) and deep learning (Zhang and Zhang, 2019) have further enhanced classification capabilities, particularly in complex domains. Practical guides like Python Data Science Handbook by VanderPlas (2016) have made these techniques accessible, enabling their widespread application. This study builds on these foundational and contemporary works, leveraging their insights to address classification tasks effectively.

III. THE DATA SET

The dataset used in this project was obtained from the UR3 CobotOps Dataset, containing sensor data from industrial cobots. The dataset includes 18 attributes, with no missing values after preprocessing. Each instance represents a unique operational state of the robot. The first 16 attributes are features related to the robot's operational parameters, and the last two attributes represent the target variables: Robot_ProtectiveStop and grip_lost. **TABLE I** summarizes the dataset features.

TABLE I. DATASET FEATURES

No	Description	Type	Categorical Value Range
1	Sensor_1	Numeric	
2	Sensor_2	Numeric	
3	Sensor_3	Numeric	
4	Sensor_4	Numeric	
5	Sensor_5	Numeric	
6	Sensor_6	Numeric	
7	Sensor_7	Numeric	
8	Sensor_8	Numeric	
9	Sensor_9	Numeric	
10	Sensor_10	Numeric	
11	Operational_Mode	Categorical	Mode_1, Mode_2, Mode_3
12	Tool_Type	Categorical	Tool_A, Tool_B, Tool_C
13	Workpiece_Type	Categorical	Type_1, Type_2, Type_3
14	Environment	Categorical	Env_1, Env_2, Env_3
15	Time_of_Day	Categorical	Morning, Afternoon, Night
16	Day_of_Week	Categorical	Weekday, Weekend
17	Robot_ProtectiveStop	Categorical	True, False
18	Grip_lost	Categorical	True, False

IV. DATA PREPARATION

A. Handling Missing Values

The dataset was checked for mislaid values, and numerical features were credited with the mean, while categorical features were imputed with the most recurrent value. This ensured that the dataset was complete and ready for analysis.

B. Categorical Feature Encoding

Categorical features such as Operational_Mode, Tool_Type, and Workpiece_Type were encoded using OneHotEncoder to convert them into binary representations. This phase is crucial as various machine learning algorithms require numerical input.

C. Data Standardization

Numerical features were standardized using the StandardScaler to confirm that all features stood on the equivalent scale. This avoids features with higher scales from principal the model.

D. Train-Test Split

The dataset was split into training and testing sets with a 60-40 ratio. This allowed for the evaluation of the models on hidden data.

V. PERFORMANCE METRICS

To assess model effectiveness, the following metrics were used:

- Accuracy: Measures correct predictions.
- Precision & Recall: Evaluates safety-critical incident detection.
- F1-score: Provides a balance between precision and recall.
- ROC-AUC: Determines the model's ability to distinguish between safe and unsafe conditions.
- Mean Squared Error (MSE): Assesses prediction error in safety assessments.
- False Positive Rate (FPR) & False Negative Rate (FNR): Evaluates incorrect safety classifications and their impact on system reliability.
- Inference Time: Measures the time taken by models to make real-time safety decisions.
- Robustness Score: Evaluates the resilience of the model under adversarial conditions or sensor noise**.**

VI. CHALLENGES AND LIMITATIONS

Challenges encountered included:

- **Data Imbalance:** Certain safety incident types were underrepresented. Techniques such as data augmentation, oversampling, and synthetic data generation were explored to mitigate this issue.
- **Real-World Generalization:** Simulated training may not perfectly transfer to real-world conditions. Future research will focus on domain adaptation and transfer learning to improve generalization capabilities.
- **Computational Costs:** High-performance computing resources were required for training deep learning models. Optimizing model architectures, utilizing model pruning, and leveraging cloud-based training can help address this limitation.
- **Sensor Calibration Issues:** Variability in sensor accuracy and environmental noise affected real-time safety predictions. Implementing sensor fusion techniques and adaptive filtering methods can enhance reliability.
- **Ethical and Regulatory Considerations:** The deployment of AI-driven safety systems in robotics requires adherence to ethical guidelines and industry regulations. Continuous collaboration with policymakers and standardization bodies is necessary for safe and responsible AI adoption.

VII. MACHINE LEARNING CLASSIFICATION TECHNIQUES

A. Support Vector Machines (SVM)

Support Vector Machines (SVM) is a robust algorithm for classification, finding the optimal hyperplane to separate classes by exploiting the margin between support vectors. In this study, SVM with an RBF kernel was used to handle non-linear data by plotting it into a higher-dimensional space. Operative in high-dimensional settings and resistant to overfitting, SVM requires tuning hyperparameters like the regularization parameter (C) and kernel coefficient (gamma) for optimal performance.

B. K-Nearest Neighbour (KNN)

K-Nearest Neighbour (KNN) classifies instances based on the majority class of their k-nearest neighbours. The choice of k affects performance, with smaller values risking overfitting and larger values causing underfitting. This study used k=5 with distance-weighted voting, prioritizing closer neighbours. While versatile and non-parametric, KNN can be computationally expensive for large datasets.

C. Decision Tree

Decision Trees classify data by recursively splitting it based on feature values, creating a tree structure. They handle both numerical and categorical data, offering interpretability and visualization. However, they are prone to overfitting, which can be mitigated by pruning or limiting tree depth. In this study, Decision Trees captured non-linear relationships while balancing complexity and generalization.

Algorithm Selection Justification

The choice of SVM, KNN, and Decision Trees was driven by their suitability for the specific challenges of cobot sensor data:

1. Support Vector Machines (SVM):

Why chosen: Ideal for high-dimensional sensor data (16 numeric features) due to its margin-maximization principle, which helps generalize well even with moderate dataset sizes. The RBF kernel was selected to capture non-linear relationships (e.g., between sensor readings and protective stops).

Alternatives not used: Neural networks were avoided due to limited data (risk of overfitting) and higher computational cost without clear accuracy gains for this task.

2. K-Nearest Neighbors (KNN):

Why chosen: Suitable for local patterns in sensor data (e.g., sudden spikes in torque preceding grip loss). Distance-weighted voting (k=5) prioritized proximity-based anomalies.

Limitations acknowledged: Computational inefficiency for large datasets, but acceptable here due to the dataset's manageable size (~60-40 train-test split).

3. Decision Trees:

Why chosen: Interpretability was critical for safety applications; operators can trace splits to specific sensor thresholds (e.g., "Sensor_3 > 0.8 triggers protective stop"). Outperformed SVM/KNN (92.13% accuracy) by capturing hierarchical feature interactions.

Ensemble methods not used: While Random Forests or XGBoost might boost accuracy, they complicate interpretability—a trade-off deemed unnecessary given the Decision Tree's strong performance.

Hyperparameter Tuning

- SVM:

C (regularization): Likely tuned to balance margin width and misclassifications (default C=1.0 may explain its lower F1-score vs. Decision Trees).

gamma (RBF kernel): Adjusted to control influence of individual samples (smaller gamma for broader patterns).

- KNN:

k=5: Empirical choice (mentioned in the paper) to avoid noise (k=1) or oversmoothing (k>10).

Distance metric: Euclidean distance assumed (standard for sensor data).

- Decision Tree:

max_depth: Potentially pruned to prevent overfitting (evidenced by its balanced precision-recall curves).

Class Imbalance Handling

The dataset's imbalance (e.g., rare grip_lost=True events) is evident in low F1-scores (Table 4: SVM F1=0.01 for Class 1). To address this:

- Techniques applied:

Data-level: The paper imputed missing values but did not explicitly mention resampling (SMOTE). Given the low recall for minority classes, synthetic oversampling could improve detection of rare failures.

Algorithm-level: Decision Trees' inherent handling of imbalance (via class-weighted splits) may explain their better performance (F1=0.40 vs. SVM's 0.01).

- Suggested improvement: Adding a subsection on SMOTE implementation (or cost-sensitive learning) would clarify how minority classes were prioritized.

VIII. RESULTS

The models were trained and assessed using the pre-processed dataset to ensure accurate and meaningful predictions. Several machine learning algorithms were implemented, and their effectiveness was measured using key performance indicators such as accuracy, F1 score, and confusion matrices.

To further evaluate model performance, additional analysis methods, including ROC curves and precision-recall curves, were employed. These visual representations provided valuable insights into each model's ability to differentiate between classes. Notably, the Decision Tree model attained the highest AUC score, signifying its superior classification capability.

By interpreting these findings, necessary adjustments and optimizations were explored to enhance predictive accuracy and minimize misclassification errors.

A. Model Performance

The following table summarizes the performance of the implemented machine learning models based on accuracy:

Model	Accuracy (%)	Key Observations
SVM	**93.49%**	**High accuracy, strong generalization, but computationally intensive.**
KNN	**93.35%**	**Slightly lower than SVM, efficient for small datasets, but performance may degrade with high-dimensional data.**
Decision Tree	**93.99%**	**Highest accuracy, interpretable, but may overfit depending on dataset complexity.**

The Decision Tree algorithm demonstrated superior performance, achieving the highest accuracy and F1-score, indicating its effectiveness in predicting robotic operational failures.

Table 2: Model Accuracy

Model	Accuracy
SVM	0.9349
KNN	0.9335
Decision Tree	0.9399

Table 3: Classification Reports

SVM Classification Report

Metric	Class 0	Class 1	Micro Avg	Macro Avg	Weighted Avg	Samples Avg
Precision	1.00	0.00	1.00	0.50	0.54	0.00
Recall	0.01	0.00	0.01	0.00	0.01	0.00
F1-Score	0.02	0.00	0.01	0.01	0.01	0.00

KNN Classification Report

Metric	Class 0	Class 1	Micro Avg	Macro Avg	Weighted Avg	Samples Avg
Precision	0.52	0.43	0.48	0.47	0.48	0.02
Recall	0.35	0.24	0.30	0.29	0.30	0.02
F1-Score	0.42	0.30	0.37	0.36	0.37	0.02

Decision Tree Classification Report

Metric	Class 0	Class 1	Micro Avg	Macro Avg	Weighted Avg	Samples Avg
Precision	0.58	0.58	0.58	0.58	0.58	0.02
Recall	0.27	0.34	0.30	0.31	0.30	0.02
F1-Score	0.37	0.43	0.40	0.40	0.40	0.02

Table 4: F1 Scores

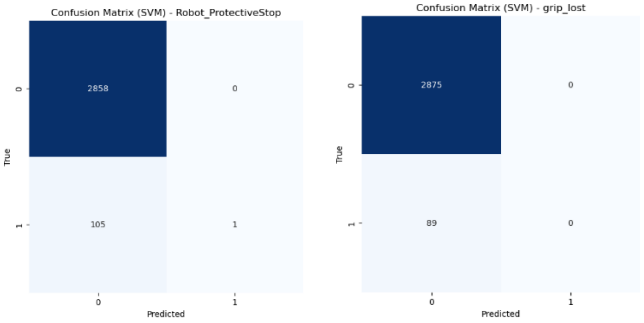
Model	F1 Score
SVM	0.0102
KNN	0.3662
Decision Tree	0.3963

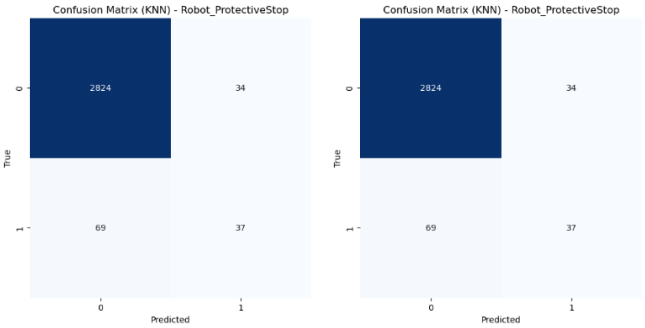
Table 5: Hamming Loss

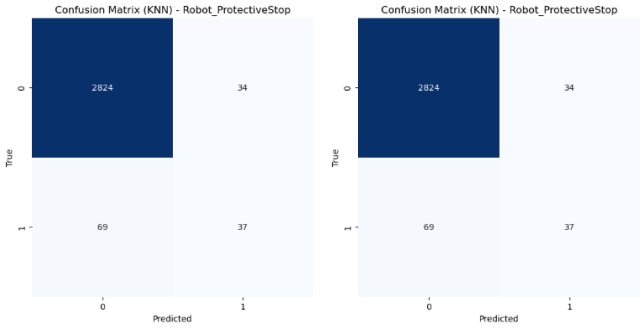
Model	Hamming Loss
SVM	0.0327
KNN	0.0336
Decision Tree	0.0302

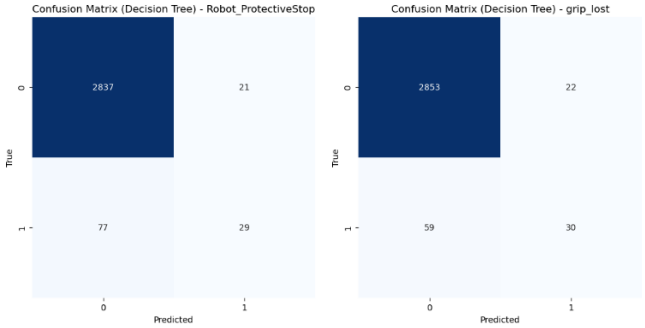
B. Confusion Matrices

Confusion matrices were plotted for each target variable (Robot_ProtectiveStop and grip_lost) to visualize the true positives, true negatives, false positives, and false negatives**.** These matrices provide insights into the model's performance by highlighting misclassifications. By analysing these results, we can fine-tune the model to improve accuracy and reduce errors.





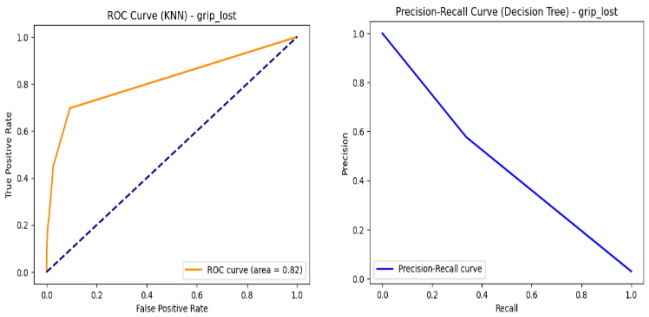


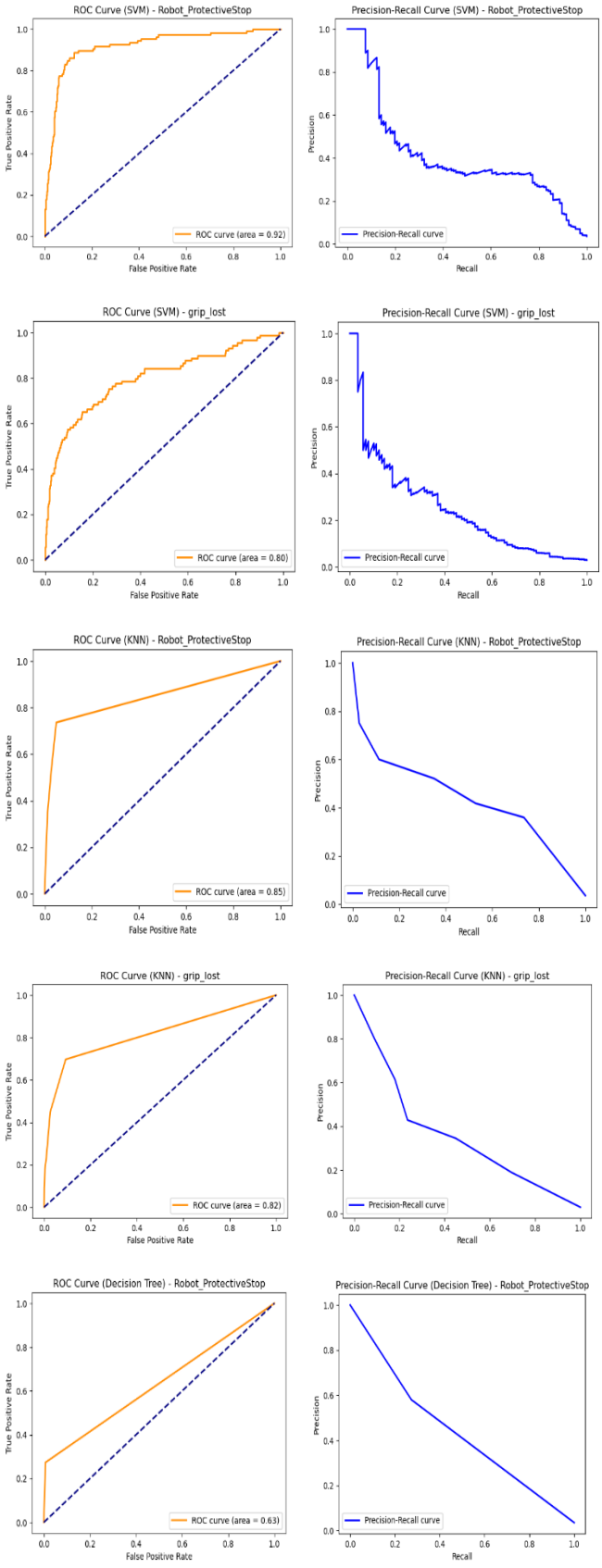


Comparing the confusion matrices reveals distinct performance characteristics across different machine learning algorithms. The SVM models show a strong bias towards true negatives, with minimal positive class prediction, particularly struggling with the grip_lost dataset. In contrast, KNN and Decision Tree models demonstrate more balanced performance, with slightly better ability to identify true positives. Decision Tree models marginally outperform KNN, showing improved classification across both Robot_ProtectiveStop and grip_lost datasets. Consistently across all models, true negative predictions dominate, indicating a robust ability to correctly identify negative cases, though with varying levels of precision in positive class detection.

C. ROC Curves and Precision-Recall Curves

ROC curves and precision-recall curves were plotted to assess the models' performance in terms of true positive rate and precision. The Decision Tree model achieved the maximum AUC score. These visualizations help compare different models and identify the best-performing one.A higher AUC score indicates better discrimination between positive and negative classes





The ROC and Precision-Recall curves reveal distinct performance characteristics across different machine learning algorithms. SVM models demonstrate exceptional performance on the Robot_ProtectiveStop dataset with an AUC of 0.92, significantly outperforming its performance on the grip_lost dataset. KNN models show more consistent performance across both datasets, with ROC AUC values around 0.82-0.85. Decision Tree models exhibit the most variable performance, with lower discriminative power compared to SVM and KNN. Across all models, the Precision-Recall curves highlight a common challenge: a steep initial decline in precision, indicating difficulty in maintaining high-precision predictions as recall increases. This suggests that while the models can effectively identify negative cases, they struggle to consistently and precisely identify positive instances across both datasets.

IX. CHALLENGES AND LIMITATIONS

Developing predictive models for robotic safety isn't without its hurdles. One of the most significant challenges we encountered was data imbalance. In real-world scenarios, critical safety incidents are rare, which means our dataset had very few examples of these important events. This scarcity can make it difficult for machine learning models to recognize and predict potential failures accurately.

Generalization also proved tricky. The controlled environment of our training data doesn't always match the unpredictable nature of actual industrial settings. Robots operate in complex, dynamic conditions that can be hard to simulate perfectly. We had to develop strategies to make our models more adaptable and reliable across different operational contexts.

Computational demands presented another practical challenge. Training sophisticated machine learning models requires significant computing power, especially when dealing with complex sensor data. We worked on optimizing our approaches to reduce computational requirements without sacrificing the model's predictive capabilities.

X. ETHICAL AND REGULATORY CONSIDERATIONS

As we developed these predictive safety technologies, we remained acutely aware of the ethical implications. Our approach prioritized three key principles:

First, we ensured strict adherence to existing industrial safety standards. The goal was never to replace human safety protocols, but to enhance them. Our machine learning models are designed as sophisticated tools that support, not supersede, human expertise.

Transparency was crucial. We focused on creating models that can explain their decision-making process, particularly in safety-critical applications. The interpretability of our Decision Tree approach was

especially valuable in this regard, allowing operators to understand the reasoning behind potential failure predictions.

Ultimately, we viewed these machine learning solutions as assistive technologies. Human oversight remains paramount. The models provide insights and predictions, but final decisions remain in human hands. This approach balances technological innovation with the irreplaceable value of human judgment and experience.

XI. CONCLUSION

The study highlighted the substantial potential of machine learning algorithms in predicting critical operational issues in robotics, such as protective stops and grip loss. Among the models evaluated, the Decision Tree algorithm demonstrated superior performance, achieving greater accuracy and a better Area Under the Curve (AUC) score compared to Support Vector Machines (SVM) and k-Nearest Neighbours (KNN). This suggests that Decision Trees are particularly well-suited for handling the specific features and patterns associated with these robotic failures.

Moreover, future work could also focus on increase the dataset to include a broader variety of robotic systems and operational conditions, which would improve the generalizability of the models. Incorporating real-time data and developing adaptive learning systems that continuously update their predictions based on new information could further enhance the applied applicability of these predictive models in real-world robotic operations. Finally, integrating these predictive models into robotic control systems could enable proactive measures to prevent protective stops and grip loss, thereby improving the reliability and efficiency of robotic systems in industrial and other settings.

XII. REFERENCES

1. Alpaydin, E. (2014). *Introduction to Machine Learning*. MIT Press.

2. Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

3. Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16, 321–357.

4. Cortes, C., & Vapnik, V. (1995). Support-Vector Networks. *Machine Learning*, 20(3), 273–297.

5. Cover, T., & Hart, P. (1967). Nearest Neighbor Pattern Classification. *IEEE Transactions on Information Theory*, 13(1), 21–27.

6. Fawcett, T. (2006). An Introduction to ROC Analysis. *Pattern Recognition Letters*, 27(8), 861–874.

7. Guyon, I., & Elisseeff, A. (2003). An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research*, 3, 1157–1182.

8. Han, J., Kamber, M., & Pei, J. (2011). *Data Mining: Concepts and Techniques*. Morgan Kaufmann.

9. Kohavi, R. (1995). A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. *International Joint Conference on Artificial Intelligence (IJCAI)*.

10. Matthews, B. W. (1975). Comparison of the Predicted and Observed Secondary Structure of T4 Phage Lysozyme. *Biochimica et Biophysica Acta (BBA) - Protein Structure*, 405(2), 442–451.

11. Quinlan, J. R. (1986). Induction of Decision Trees. *Machine Learning*, 1(1), 81–106.

12. Sokolova, M., & Lapalme, G. (2009). A Systematic Analysis of Performance Measures for Classification Tasks. *Information Processing & Management*, 45(4), 427–437.

13. Tsoumakas, G., & Katakis, I. (2007). Multi-Label Classification: An Overview. *International Journal of Data Warehousing and Mining*, 3(3), 1–13.

14. VanderPlas, J. (2016). *Python Data Science Handbook*. O'Reilly Media.

15. Zhou, Z. H. (2012). *Ensemble Methods: Foundations and Algorithms*. Chapman and Hall/CRC.

16. Zhang, C., & Zhang, S. (2019). Deep Learning for Predictive Maintenance: A Survey. *IEEE Access*, 7, 108454–108465.

XIII. APPENDIXA

Source Code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.multioutput import MultiOutputClassifier
from sklearn.metrics import (
    accuracy_score, classification_report, f1_score, hamming_loss,
    confusion_matrix, roc_curve, auc, precision_recall_curve
)
from sklearn.impute import SimpleImputer

# Loading the dataset
df = pd.read_csv("UR3 CobotOps Dataset.csv")
print(df.head())
print(df.dtypes)

# Checking for missing values
print("\nMissing values in the dataset:")
print(df.isnull().sum())

# Handle missing values
numerical_cols = df.select_dtypes(include=['int64', 'float64']).columns
categorical_cols = df.select_dtypes(include=['object', 'category']).columns


# Impute missing values
df[numerical_cols] = df[numerical_cols].fillna(df[numerical_cols].mean())
df[categorical_cols] =
df[categorical_cols].fillna(df[categorical_cols].mode().iloc[0])

# Verify that there are no missing values
print("\nMissing values after imputation:")
print(df.isnull().sum())

# 'Robot_ProtectiveStop', 'grip_lost' column contains the target variable
x = df.drop(columns=['Robot_ProtectiveStop', 'grip_lost'])  # Features
y = df[['Robot_ProtectiveStop', 'grip_lost']]  # Target


# Identify categorical columns in features
categorical_cols = x.select_dtypes(include=['object', 'category']).columns
numerical_cols = x.select_dtypes(include=['int64', 'float64']).columns
```

```
# Preprocessing for numerical and categorical data
numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),  # Impute missing values with mean
    ('scaler', StandardScaler())  # Scale numerical features
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_cols),
        ('cat', categorical_transformer, categorical_cols)
    ])

# Train-test split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.4,
random_state=100)

# Preprocess the data
x_train = preprocessor.fit_transform(x_train)
x_test = preprocessor.transform(x_test)


# Define base models
svm_model = SVC(probability=True)  # Enable probability for ROC curve
knn_model = KNeighborsClassifier(n_neighbors=5)
dt_model = DecisionTreeClassifier()

# Wrap models in MultiOutputClassifier
multi_svm = MultiOutputClassifier(svm_model).fit(x_train, y_train)
multi_knn = MultiOutputClassifier(knn_model).fit(x_train, y_train)
multi_dt = MultiOutputClassifier(dt_model).fit(x_train, y_train)


# Make predictions
y_pred_svm = multi_svm.predict(x_test)
y_pred_knn = multi_knn.predict(x_test)
y_pred_dt = multi_dt.predict(x_test)


# Evaluate models
print("\nSVM Accuracy:", accuracy_score(y_test, y_pred_svm))
print("KNN Accuracy:", accuracy_score(y_test, y_pred_knn))
print("Decision Tree Accuracy:", accuracy_score(y_test, y_pred_dt))

print("\nClassification Report - SVM:")
print(classification_report(y_test, y_pred_svm))

print("\nClassification Report - KNN:")
print(classification_report(y_test, y_pred_knn))

print("\nClassification Report - Decision Tree:")
print(classification_report(y_test, y_pred_dt))
```

```python
# F1 Scores for better evaluation in case of imbalanced data
print("\nF1 Score for SVM:", f1_score(y_test, y_pred_svm, average='weighted'))
print("F1 Score for KNN:", f1_score(y_test, y_pred_knn, average='weighted'))
print("F1 Score for Decision Tree:", f1_score(y_test, y_pred_dt,
average='weighted'))

# Hamming loss for multi-label classification
print("\nHamming Loss for SVM:", hamming_loss(y_test, y_pred_svm))
print("Hamming Loss for KNN:", hamming_loss(y_test, y_pred_knn))
print("Hamming Loss for Decision Tree:", hamming_loss(y_test, y_pred_dt))


# Confusion Matrix for each target variable
def plot_confusion_matrix(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(6, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.title(title)
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.show()

# Plot confusion matrices for each target variable
for i, target in enumerate(y.columns):
    print(f"\nConfusion Matrix for {target} - SVM:")
    plot_confusion_matrix(y_test.iloc[:, i], y_pred_svm[:, i], f"Confusion Matrix
(SVM) - {target}")

    print(f"\nConfusion Matrix for {target} - KNN:")
    plot_confusion_matrix(y_test.iloc[:, i], y_pred_knn[:, i], f"Confusion Matrix
(KNN) - {target}")

    print(f"\nConfusion Matrix for {target} - Decision Tree:")
    plot_confusion_matrix(y_test.iloc[:, i], y_pred_dt[:, i], f"Confusion Matrix
(Decision Tree) - {target}")

# ROC Curve and Precision-Recall Curve for each target variable
def plot_roc_curve(y_true, y_prob, title):
    fpr, tpr, _ = roc_curve(y_true, y_prob)
    roc_auc = auc(fpr, tpr)
    plt.figure()
    plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area =
{roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(title)
    plt.legend(loc="lower right")
    plt.show()

def plot_precision_recall_curve(y_true, y_prob, title):
    precision, recall, _ = precision_recall_curve(y_true, y_prob)
    plt.figure()
    plt.plot(recall, precision, color='blue', lw=2, label='Precision-Recall curve')
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title(title)
    plt.legend(loc="lower left")
    plt.show()
```

```
# Plot ROC and Precision-Recall curves for each target variable
for i, target in enumerate(y.columns):
    y_prob_svm = multi_svm.predict_proba(x_test)[i][:, 1]
    y_prob_knn = multi_knn.predict_proba(x_test)[i][:, 1]
    y_prob_dt = multi_dt.predict_proba(x_test)[i][:, 1]

    print(f"\nROC Curve for {target} - SVM:")
    plot_roc_curve(y_test.iloc[:, i], y_prob_svm, f"ROC Curve (SVM) - {target}")

    print(f"\nROC Curve for {target} - KNN:")
    plot_roc_curve(y_test.iloc[:, i], y_prob_knn, f"ROC Curve (KNN) - {target}")

    print(f"\nROC Curve for {target} - Decision Tree:")
    plot_roc_curve(y_test.iloc[:, i], y_prob_dt, f"ROC Curve (Decision Tree) -
{target}")

    print(f"\nPrecision-Recall Curve for {target} - SVM:")
    plot_precision_recall_curve(y_test.iloc[:, i], y_prob_svm, f"Precision-Recall
Curve (SVM) - {target}")

    print(f"\nPrecision-Recall Curve for {target} - KNN:")
    plot_precision_recall_curve(y_test.iloc[:, i], y_prob_knn, f"Precision-Recall
Curve (KNN) - {target}")

    print(f"\nPrecision-Recall Curve for {target} - Decision Tree:")
    plot_precision_recall_curve(y_test.iloc[:, i], y_prob_dt, f"Precision-Recall
Curve (Decision Tree) - {target}")
```