Intent Classification during Human-Robot Contact

by

Wesley Fisher

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Masters of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2020

© Wesley Fisher 2020

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Robots are used in many areas of industry and automation. Currently, human safety is ensured through physical separation and safeguards. However, there is increasing interest in allowing robots and humans to work in close proximity or on collaborative tasks. In these cases, there is a need for the robot itself to recognize if a collision has occurred and respond in a way which prevents further damage or harm. At the same time, there is a need for robots to respond appropriately to intentional contact during interactive and collaborative tasks.

This thesis proposes a classification-based approach for differentiating between several intentional contact types, accidental contact, and no-contact situations. A dataset is developed using the Franka Emika Panda robot arm. Several machine learning algorithms, including Support Vector Machines, Convolutional Neural Networks, and Long Short-Term Memory Networks, are applied and used to perform classification on this dataset.

First, Support Vector Machines were used to perform feature identification. Comparisons were made between classification on raw sensor data compared to data calculated from a robot dynamic model, as well as between linear and nonlinear features. The results show that very few features can be used to achieve the best results, and accuracy is highest when combining raw data from sensors with model-based data. Accuracies of up to 87% were achieved. Methods of performing classification on the basis of each individual joint, compared to the whole arm, are tested, and shown not to provide additional benefits.

Second, Convolutional Neural Networks and Long Short-Term Memory Networks were evaluated for the classification task. A simulated dataset was generated and augmented with noise for training the classifiers. Experiments show that additional simulated and augmented data can improve accuracy in some cases, as well as lower the amount of real-world data required to train the networks. Accuracies up to 93% and 84% we achieved by the CNN and LSTM networks, respectively. The CNN achieved an accuracy of 87% using all real data, and up to 93% using only 50% of the real data with simulated data added to the training set, as well as with augmented data. The LSTM achieved an accuracy of 75% using all real data, and nearly 80% accuracy using 75% of real data with augmented simulation data.

Acknowledgements

I would like to thank the following people for their support throughout this project, my degree, and throughout my life. I would like to thank:

My supervisors, Dana Kulić and Stephen Smith for their support and guidance. Your guidance in both research and writing has been greatly appreciated. This experience has been a wonderful introduction into the world of research and academics.

My thesis readers, Christopher Nielsen and James Tung, for their time in reviewing my thesis. I am grateful to have had both of you as professors, as well as to have your feedback on my work.

My professors at UW for providing me with inspiration and education, which has lead to this point.

Everyone at the Adaptive Systems Lab, both for direct assistance, and for creating a welcoming and insightful environment.

The staff of the UW Robohub, including Brandon DeHart, Alex Werner, Marie Charbonneau, and Robert Wagner. Your support, assistance, and company has made this a wonderful time.

The students I have worked with or spent time with at UW: Sagar Rajendran, Ilknur Umay, fellow TAs, and those in student teams.

My friends elsewhere. Keeping in touch has helped to keep my sane, and made this experience more enjoyable.

Finally, I would like to thank my family for all of their love and support throughout these many years.

And thank you to everyone who I have not been able to mention here, but who has helped me in some form.

Dedication

This work is dedicated to my parents Brian and Bonnie and my brother Mitch, for all of their love and support throughout many years. I am where I am because of all you have been able to provide me.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Robots have been widely used in the manufacturing industry, and have replaced humans in performing difficult or dangerous work [61]. In such industrial settings, human safety is ensured via physical separation of humans and robots [61]. However, there is now a desire to allow robots to act alongside humans, for the purposes of performing collaborative tasks, or providing care to humans [29] [61]. In situations such as this, old methods of providing safety by means of separation are no longer applicable, and new methods and standards must be adopted [61] [29].

When collaborating with a human, a robot needs to avoid unintended collisions, but also the ability to respond to an intentional interaction, as opposed to simply treating an interaction as a collision. Previous research has developed collision-detection systems, which can increase safety by allowing a robot to react appropriately to an unintended contact [39] [70]. Previous works have also attempted to distinguish intentional interaction from unintentional contact [38] [18]. These works have been limited to the detection of collision, to distinguishing unintentional collisions from intentional interactions, or to extracting information about an occurring collision. They have not investigated determining the human intent behind the interaction, which involves classifying the situation as one of many types of intentional interactions.

This thesis investigates methods of classifying different intentional interaction types between a human and an industrial manipulator robot, using Machine Learning classifiers. The interaction types considered include: a rapid shove of the robot as if to prevent or stop a collision, gradual adjustments to the robot, as if to correct for error or adjust motion, light accidental contacts, and nominal operation. The ability to classify interaction in this way can allow for improved safety and collaboration, as a robot will be able to respond appropriately based on understanding the human's intent during the contact. The classifiers are developed and tested in simulations and a physical interaction dataset.

First, a Support Vector Machine classifier is developed. An investigation into feature selection is performed, which demonstrates the usefulness of combining model-based and non-model-based features. A method used by [39], involving preforming classification for each joint, is compared against whole-arm classification. Finally, sources of error are isolated.

Second, classifiers based on two different neural network architectures are developed. This thesis then investigates augmenting the training process of neural network classifiers with data generated in simulation, to reduce the amount of real-world data needing to be gathered.

1.1 Contributions

This thesis makes the following contributions:

Support Vector Machine, Convolutional Neural Network, and Long Short-Term Memory network classifiers are developed to distinguish between types of physical human-robot interaction. Physical and simulation datasets are generated to evaluate the classifiers performance.

The benefits of combining model-based and non-model-based features are demonstrated. Feature exploration is performed, distinguishing between input data taken from robot sensors, data calculated from a robot model, and data with additional transformations performed. The results of the experiments performed demonstrate that combining model-based and sensor-based features allows for increased performance. This strategy of combining raw- and model-based features may be used with future classifiers to achieve better results.

Support Vector Machines using data from the entire robot arm, or the use of an ensemble of classifiers each trained on individual joints, as seen in [39] are compared. Classifiers that act on data from individual joints, as opposed to the entire arm, are shown to provide little or no additional benefits. A modification of this method to train multiple classifiers on data from 3 joints only reaches the classification accuracy of a classifier trained on data from the entire arm, but does not surpass it.

Improved neural network training through use of a simulated dataset, and ability to train with less real-world data is demonstrated. A simulated dataset is generated to mimic a real-world dataset, and used to augment the training process of neural networks with limited real-world data. Experiments demonstrate the ability of simulation data to improve classification accuracy, and allow training with less real-world data. This method may be beneficial in other situations where real-world data is sparse.

1.2 Thesis Outline

This thesis is organized as follows:

Chapter 2 presents background information and a review of literature relevant to this thesis.

Chapter 3 presents the Machine Learning problem formulation, and common elements of methodology used in this thesis.

Chapter 4 presents the development and testing of Support Vector Machines in classifying human-robot contact.

Chapter 5 presents the development and testing of a convolutional and a Long Short-Term Memory neural network for classifying human-robot contact, as well as the generation of a simulated dataset.

Chapter 6 presents the results of the tests described in Chapter 4 and Chapter 5.

Chapter 7 presents final conclusions, limitations and suggestions for future work.

Chapter 2

Background and Related Work

This chapter reviews the background concepts related to this thesis, including rigid body dynamics, estimation of external torques, and an introduction to Support Vector Machines, Convolutional Neural Networks, and Long Short-Term Memory Networks.

Next, previous works related to collision detection during human-robot interaction are reviewed.

Finally, an overall summary is provided, identifying the open challenges and limitations of current work.

2.1 Robot Modelling and State Estimation

This section reviews the rigid body dynamics and methods used to estimate external torque acting on a robot. Where not otherwise specified, [65] was used as a reference when compiling this information.

2.1.1 Robot Arm Kinematics

Consider a rigidly mounted robot arm, whose configuration is specified by a set of joint angles. A frame is rigidly attached to each link. Joint angles can then be used to calculate transformations between frames of the robot. Figure 2.1 [65] provides an illustration of several links of a robot arm, along with the x and z-axes of the frames attached to each link.

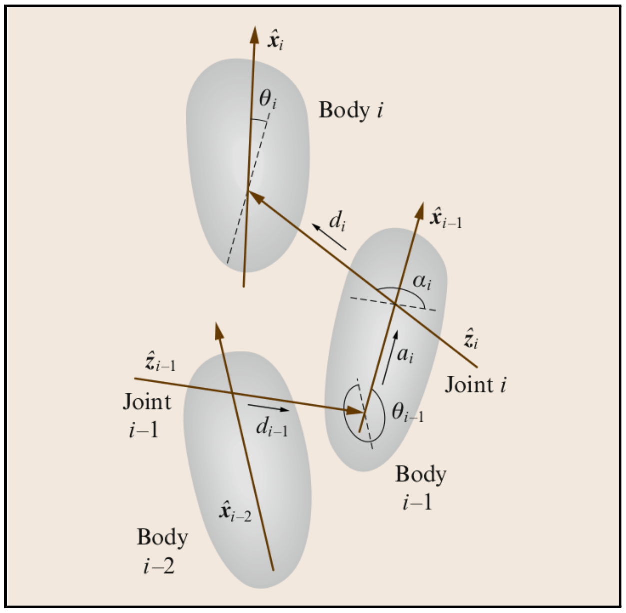

Figure 2.1: Robot bodies and frames connected via joints [65]

Transformations between frames of a robot can be computed through the use of homogeneous transformations, which take the form:

$$T_i^j = \begin{bmatrix} R_i^j & p_i^j \\ 0^R & 1 \end{bmatrix} \tag{2.1}$$

with:

- T_i^j the transformation from frame i to j
- R_i^j the rotation from frame i to frame j
- p_i^j the translation between from frame i to frame j.

One way to calculate these transformation is through the use of Denavit-Hartenburg parameters, discussed in detail in [65] and also shown in Figure 2.1. These are parameters which depend on the fixed geometry of the robot, and dynamic joint angles or positions. For the parameters of joint i [65]:

- a_i: distance from $\hat{z}_{i-1}$ to $\hat{z}_i$ along $\hat{x}_{i-1}$

- α_i: angle from $\hat{z}_{i-1}$ to $\hat{z}_i$ about $\hat{x}_{i-1}$
- d_i: distance from $\hat{x}_{i-1}$ to $\hat{x}_i$ along $\hat{z}_i$
- θ_i: angle from $\hat{x}_{i-1}$ to $\hat{x}_i$ about $\hat{z}_i$

the transformation can be calculated [65]:

$$T_{i-1}^{i} = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 & a_i \\ \sin(\theta_i)\cos(\alpha_i) & \cos(\theta_i)\cos(\alpha_i) & -\sin(\alpha_i) & \sin(\alpha_i)d_i \\ \sin(\theta_i)\sin(\alpha_i) & \cos(\theta_i)\sin(\alpha_i) & \cos(\alpha_i) & \cos(\alpha_i)d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

These transformations can be composed. The transformation of frame n with respect to the robot's base frame, 0 is [65]:

$$T_0^n = \Pi_{i=1}^{n} T_{i-1}^{i}. \tag{2.3}$$

Forward Kinematics

Forward kinematics is the calculation of positions and orientations of part of a robot, in the robot's base frame, given the joint angles. This can be accomplished by taken the given joints angles, calculating transformation matrices according to Equation (2.2), and then composing these using Equation (2.3) [65]. The output of this, in general, includes 3 position components, and a rotation matrix R.

Instantaneous Kinematics

One can take the forward kinematic equation for a given point k on a robot, expressing the rotation in Euler angles, and differentiate it with respect to time, obtaining equations in the form [65]:

$$v_0^k = J(q)\dot{q} \tag{2.4}$$

Where v_0^k is a vector composed of 3 linear and 3 angular velocities of point k with respect to the robot's base frame, given the n-dimensional vector $\dot{q}$ of current joint velocities of the robot, and $J(q)$ is a $6 \times n$ matrix, referred to as the Jacobian. The matrix $J(q)$ typically contains nonlinear equations of joint angles, as the forward kinematics will often contain multiples of trigonometric functions of joint angles. Elements of $J(q)$ can be written, for element i of v_o^k and joint j [65]:

$$J_{ij} = \frac{\partial v_i}{\partial q_j}. \tag{2.5}$$

Note that if the point k on the arm is located before some joints, the columns of $J(q)$ corresponding to those joints will be zero.

The instantaneous kinematics can be inverted to obtain [65]:

$$\dot{q} = J^{-1}(q)v_0^k. \tag{2.6}$$

This formulation permits the joint velocities required to obtain velocity v_0^k at point k to be computed. In general, $J(q)$ may not be square, full-rank, or well-conditioned. For example, if $J(q)$ is full row-rank, but not full column-rank, many right-inverses exist. In these cases, numerical inversion or pseudo-inverses may be required [65].

2.1.2 Forces

If the robot is at rest and in contact with the environment, the Jacobian of the contact point can be used to calculate the forces applied at that point by the robot, given the current joint torques τ [65]:

$$f = (J(q)^T)^{-1}\tau. \tag{2.7}$$

In Equation (2.7), f is a 6-dimensional vector of 3 forces and 3 moments acting at the point for which $J(q)$ is calculated. The numerical issues discussed in the previous section also apply here. For example, if $J(q)^T$ is full column-rank, but not full row-rank, many left-inverses of $J(q)^T$ exist.

This equation can also be inverted [65]:

$$\tau = (J(q)^T)f. \tag{2.8}$$

2.1.3 Rigid Body Dynamics

The dynamics model used in this research [65] takes the form:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau_m - \tau_F + \tau_{ext} = \tau_{tot}. \tag{2.9}$$

where:

- q is a vector of joint angles, and $\dot{q}, \ddot{q}$ are its derivatives
- $M(q)$ is the inertia matrix, which is symmetric and positive-definite
- $C(q,\dot{q})\dot{q}$ is a vector of centripetal and Coriolis terms
- $g(q)$ is a vector of gravitational terms
- τ_m is the torques applied by the robot motors
- τ_F is torques induced by friction
- τ_{ext} is torques induced on the robot joints by external forces

An important property of this equation [65], is the skew-symmetry of:

$$\dot{M}(q) - 2C(q,\dot{q}). \tag{2.10}$$

It should be noted that the rigid body dynamics equation is highly nonlinear. In addition, it requires joint acceleration values to compute τ. This can cause problems due to sensor noise and numerical differentiation, as noted by [42].

2.1.4 External Torque Estimation through Observers

Haddadin et al. [42] present and compare several methods used to estimate the external torques acting on a robot. These external torque estimates are widely used as collision-detection signals, as will be discussed in Section 2.3.1. This section reviews a basic method of estimating external torques, and a more advanced method, both described in detail in [42].

Direct Estimation of External Torque

The most straightforward way of estimating the external torque acting on a robot is through the use of the rigid body dynamics equation presented in Section 2.1.3:

$$\hat{\tau}_{ext} = \hat{M}(q)\ddot{q} + \hat{C}(q, \dot{q})\dot{q} + \hat{g}(q) - \tau_m \tag{2.11}$$

With the hat-notation representing estimations of values.

Haddadin et al. [42] notes that while this method is direct, it is susceptible to noise introduced through differentiation and double-differentiation of joint position measurements.

Haddadin et al. [42] [40] present a modification to the above method, which uses the desired joint trajectory q_d and derivatives to compute the expected torque:

$$\hat{\tau}_{ext} = \hat{M}(q_d)\ddot{q_d} + \hat{C}(q_d, \dot{q_d})\dot{q_d} + \hat{g}(q_d) - \tau_m \tag{2.12}$$

With a well-tuned controller, [42] argues that the desired trajectory will approximate the actual trajectory well. This presents the advantage that the joint velocity and acceleration are not subject to sensor noise, or calculated numerically. However, this method becomes dependent on the controller maintaining a trajectory close to the desired trajectory. It will also not be valid if a collision moves the robot away from the planned trajectory.

Takakura et al. [70] and Yamada et al. [76] also make use of an inverse dynamics model to estimate the external torque acting on the robot. Both thresholding and comparing the disturbance observer values to those from the last execution are suggested as methods to detect a collision [70].

This method may provide an effective baseline for estimating the external torque on a robot. However, the effect of noise added through differentiation, and any errors in the dynamic model used, will decrease the quality of the estimation.

Momentum Observer

The final strategy discussed by [42] and also presented in [25] [27] [5], is the momentum observer. The momentum observer makes use of the generalized momentum p of the robot:

$$p = M(q)\dot{(q)}. \tag{2.13}$$

The observer makes use of several properties of the rigid body dynamics equation to create an observer with gain parameter Ko and dynamics:

$$\dot{\hat{p}} = \tau_m + C^T(q, \dot{q})\dot{q} - g(q) + r, \quad p(0) = p_0 \tag{2.14}$$

$$\dot{r} = Ko(\dot{p} - \dot{\hat{p}}) \tag{2.15}$$

And the final form:

$$r = K_O(p(t) - \int_0^t \tau_m + C^T(q,\dot{q})\dot{q} - g(q) + r ds - p_0) \tag{2.16}$$

This observer serves as a stable first-order filter of the external torque acting on the robot, with dynamics:

$$\dot{r} = K_O(\tau_{ext} - r) \tag{2.17}$$

As an extension to the joint velocity observer, the momentum observer avoids the need for inversion of the mass matrix. Compared to using inverse dynamics, it does not require the double-differentiation of joint positions to calculate acceleration.

2.2 Background on Classifiers

This section presents a brief overview of the classifier methods used in this thesis, which include Support Vector Machines, Convolutional Neural Networks, and Long Short-Term Memory Networks.

2.2.1 Support Vector Machines

Support Vector Machines (SVMs) are a supervised machine learning method. In this thesis, they are used to perform classification.

Given a dataset consisting of pairs of feature vectors and labels (x_i, y_i), with $y_i \in [-1, +1]$ defining the class of sample i, a SVM learns a weight vector w and bias scalar b. These define a hyperplane in feature-space which separates the data samples with maximum margin between samples and the hyperplane, such that [14]:

$$\begin{aligned} x_i^T w + b > 0, &\quad y_i = +1 \\ x_i^T w + b < 0, &\quad y_i = -1. \end{aligned}$$

This is done through the use of the Lagrangian function [14]:

$$L = 0.5||w||^2 - \sum_{i=1}^{l} \alpha_i y_i (x_i^T w + b) + \sum_{i=1}^{l} \alpha_i \tag{2.18}$$

with lagrange multipliers α_i.

This formulation is feasible if the data is linearly separable in feature-space [14]. When the data is not linearly separable, the SVM solves for the optimal separating hyperplane and margins by introducing positive slack variables ζ. The equations above then become [14]:

$$\begin{aligned} x_i \cdot w + b &\geq +1 - \zeta_i, \quad y_i = +1 \\ x_i \cdot w + b &\leq -1 + \zeta_i, \quad y_i = -1 \\ \zeta_i &\geq 0, \forall i \end{aligned}$$

and

$$L = 0.5||w||^2 + C \sum_i \zeta_i - \sum_{i=1}^{l} \alpha_i \{ y_i (x_i \cdot w + b) - 1 + \zeta_i \} + \sum_{i=1}^{l} \mu_i \zeta_i \tag{2.19}$$

with Lagrange multipliers μ_i and coefficient C chosen by the user to penalize errors in classification.

Figure 2.2 illustrates a the solution to a SVM with non-separable data.

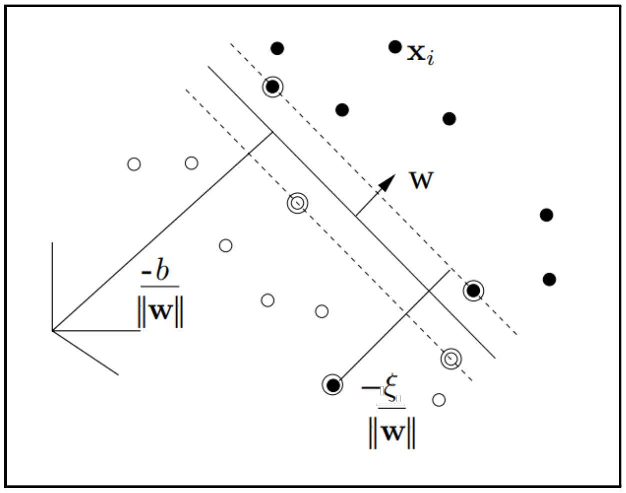

Figure 2.2: Illustration of SVM hyperplanes with non-separable data. Adapted from [14]

SVMs can perform well in high-dimensional spaces and tend to avoid overfitting [39] [14]. SVMs are able to ignore any vectors not supporting the decision boundary, and can function even in cases where the dimensionalty of vectors is grater than the number of samples [10]. They can also be extended through the use of nonlinear kernel functions [14].

2.2.2 Deep Neural Networks

Deep neural networks are a type of machine learning method. They are composed of layers of connected neurons. A deep neural network consists of an input layer, which receives numerical input data, hidden layers of neurons, and an output layer, which outputs the network prediction.

A single neuron in a network is connected to those in the preceding layer. The neuron has a numerical weight assigned to each of these connections, as well as a bias term. When doing a forward calculation of the network, the outputs of proceeding neurons are multiplied with their corresponding weights, and summed together with the bias. A nonlinear function is then calculated on this sum, and the output used as the output of the neuron. During training, the weights and bias terms of each neuron are adjusted. Figure 2.3 illustrates a single neuron in layer j of a network, with inputs, weights, and output shown.

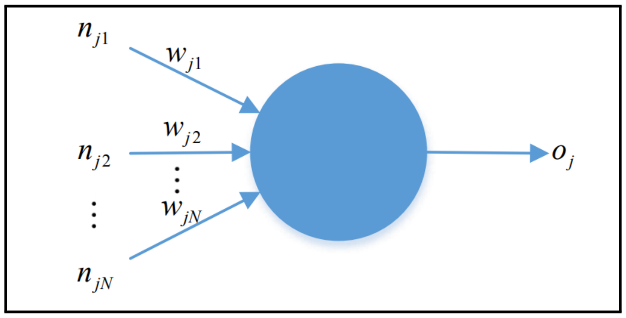

Figure 2.3: Illustration of single neuron in a Deep Neural Network [17]

Many neurons are placed in parallel to make up a layer. In dense neural network layers, each neuron in a given layer is connected to all neurons in the preceding and succeeding layers. Figure 2.4 illustrates this.

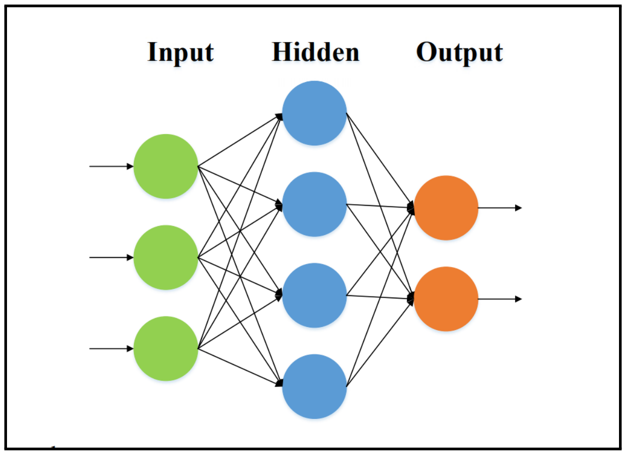

Figure 2.4: Illustration of layers in a Deep Neural Network [17]

2.2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a class of deep learning method, designed to work with data which has a grid-like structure, such as images. A CNN typically consists of convolutional layers, pooling, and dense layers [77].

A convolutional layer consists of an application of kernels, or arrays of numbers. These numbers are arranged in a small grid. This grid is placed over and moved along the input grid of data. At each location, the element-wise product is taken between overlapping kernel values and input data values [77]. The sum of these values is then fed to the output of the layer, and an activation function is applied. Applying multiple kernels leads to multiple output layers stacked depth-wise. During training, the values in each kernel are learned to identify and extract features. This is illustrated in Figure 2.5.

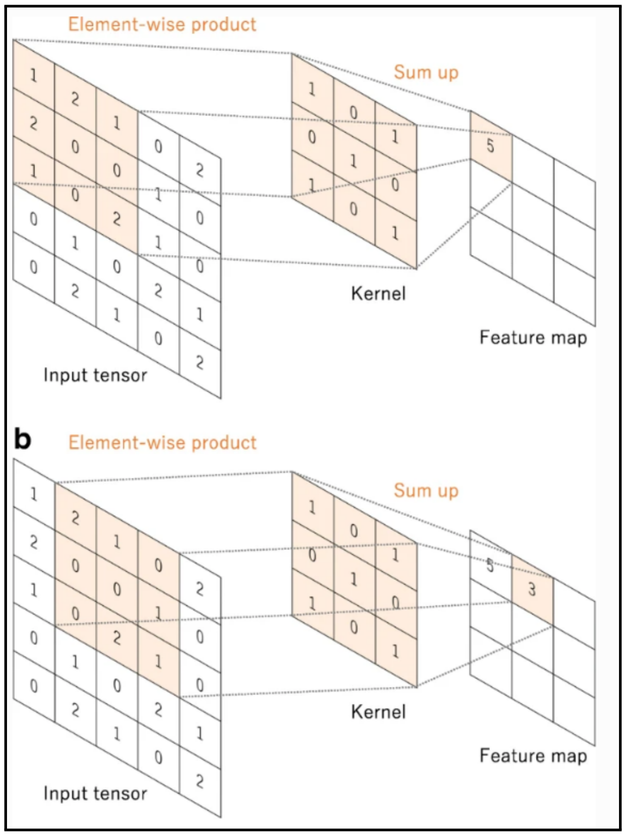

Figure 2.5: Illustration of applying a kernel to an input grid of data [77]

Pooling layers serve to reduce the size of the data at various layers of the network. This is done by combining various nearby values in the input layer to the pooling layer. For example, with a 'max pooling' layer of 2x2 pooling, each 2x2 grid of data will be reduced to a 1x1 element containing the maximum value [77]. Figure 2.6 illustrates this.

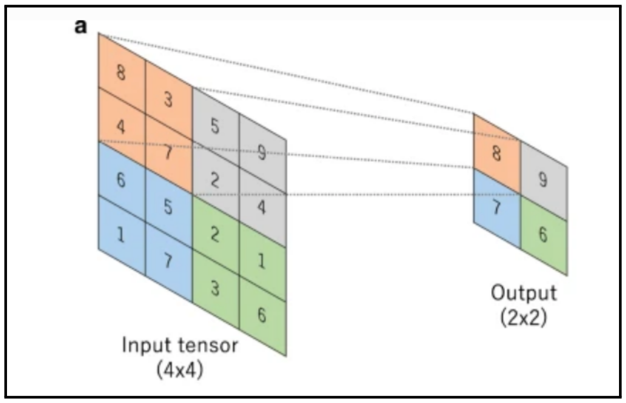

Figure 2.6: Illustration of 2x2 max pooling layer [77]

The application of repeated layers of convolution and pooling can lead to the extraction of a hierarchy of features at various levels. Often, the last of these layers is connected to one or more dense layers of the network, which provides the final classification prediction [77].

2.2.4 Long Short-Term Memory Networks

Long Short-Term Memory (LSTM) networks are a type of network which work with time-series input data, and hold a memory of previous states within the timeseries. LSTM neurons contain an internal state, as well as internal 'gates' connected to the input and output, as well as a forgetting gate. This combination of gates an internal states is able to allow a LSTM neuron to learn how to remember or forget information [67]. Figure 2.7 and Figure 2.8 show two different depections of an LSTM neuron. Figure 2.7 shows a more complicated arrangement of an LSTM into a memory cell, while Figure 2.8 shows an unrolled LSTM neuron. LSTMs can then be arranged into networks.

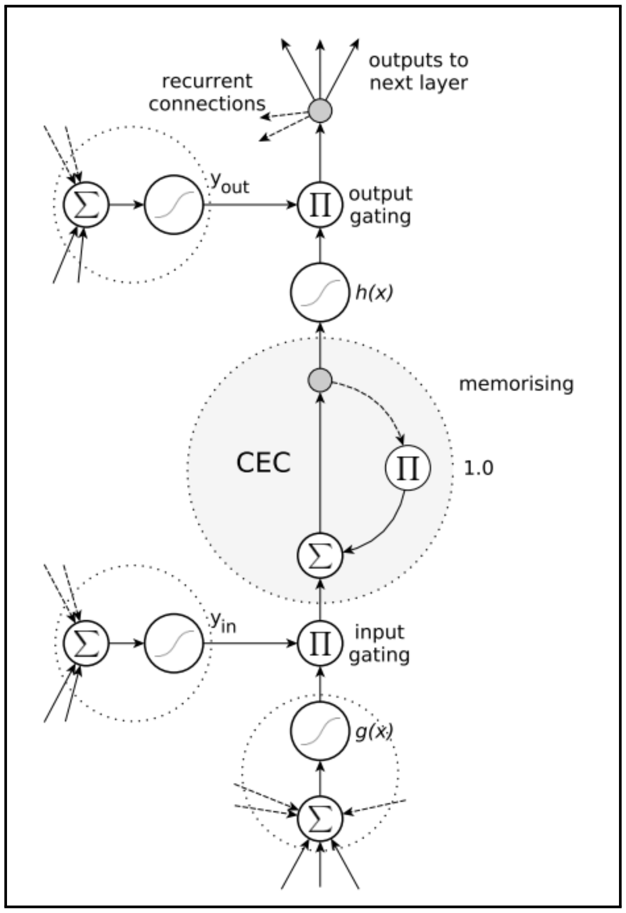

Figure 2.7: Illustration of LSTM memory cell [67]

Figure 2.7 shows some internal structures of an LSTM cell. A memory block contains one or more constant error carousels (CECs) which have recurrent connections. Updates to the cell are controlled by an input gate. The inner state is updated by taking element-wise multiplication of inputs and input gate elements, and adding these to the internal state. An output gate controls connections to the next layer in the network. The internal state is multiplied element-wise with output gate values, and taken as inputs to the next layer.

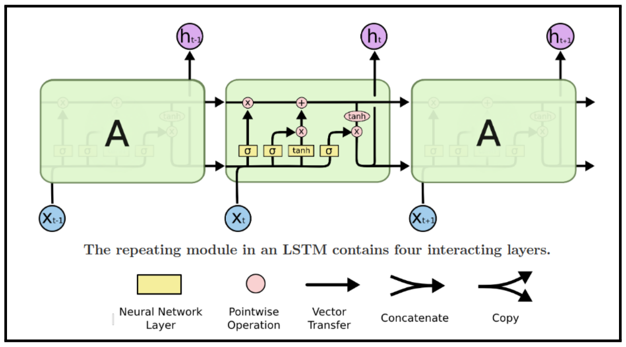

Figure 2.8: Illustration of LSTM neuron. Adapted from [54]

2.3 Related Works

This section reviews previous work in collision and contact detection during physical human-robot interaction. A large body of work exists on collision detection, which is critical for ensuring safety. Other works attempt to extract additional information from contact, or determine some form of intent from a contact.

This section first reviews previous works which use model-based methods, followed by those which use machine learning methods, and presents a final summary of identified trends.

The goal of this thesis is to be able to classify human intent, so that the robot can ultimately adapt its control strategy accordingly. Various papers discuss the benefits of being able to modify control strategies based on context, including examples discussed by [29] and [3].

Ajoudani [3] reviews recent works on human-robot interfaces. They note the benefits of using multiple sensors to allow operation in a variety of contexts, and note that force-torque sensors are commonly used in collaborative tasks. They also describe benefits of a robot platform capable of adapting its control scheme given a changing situation. Finally, [3] notes the capability of humans to adapt to changing robot dynamics, which suggests a human can learn to make use of a system which determines intent and responds accordingly.

Argall and Billard [6] review a number of works on tactile human-robot interaction. They classify work on tactile sensing into three domains: 'responding to disturbance', 'state classification', and 'behavioural selection'. The first and third apply most directly to this thesis. The papers in these areas tended to use contact location information, and the magnitude of applied forces. A similar pattern was seen when grouping works that performed 'behaviour execution' and 'behaviour guidance', which correspond to the work in this thesis.

2.3.1 Model-Based Methods

This subsection discusses model-based methods which have been identified in previous works. The majority of these methods have used model information to estimate external torques, and used these torque measurements in various ways, especially for collision detection.

Estimating External Torques

Many papers suggest detecting a collision by estimating the external torque acting on a robot, and indicating a collision has occurred if this surpasses some value. Some works use these estimates directly to detect collisions, including [42], [76], [70], [5] , while others use these estimates in machine learning models, including [39], [60].

In general, thresholds set for these purposes must be able to accommodate noise from sensors, model error, or other sources of error [42], [19]. [70] suggests comparing estimates of external torque to readings from past periods in repetitive tasks.

The momentum observer appears to be the most promising of the observers presented in [42], and is the primary observer investigated in this work. This observer is also used in [25], [5], [13], [37], [39], [19], [26], [28].

Kouris et al. [46] make use of the Fast Fourier Transform, applied to either end-effector force readings or external joint torque readings, to detect collisions. The FFT is calculated on a fixed-size window of data every time a new sample is received from the sensors, for frequency values between 10 and 100 Hz. The rate of change in the magnitude of the FFT calculated between the two windows is used as a collision-detection signal.

De Luca at al. [25], used the calculated residual vectors in a control law to respond to the collision. These were found to cause the robot to retreat from collision rapidly. Smilarly, Gaz et al. [37] use calculated residual vectors as part of their control law. Similar information is recovered by the method in [11].

Buodonno and De Luca [13] use a combination of a force-torque sensor on the base of the robot, and the momentum observer (sometimes replaced with a joint acceleration estimate) to determine the location and magnitude of a contact force. This is done through a geometric approach, using knowledge of the robot's kinematic and dynamic model. This process can be used without some combination of these sensors if assumptions are made about the contact force [13].

Mikhel et al. [52] investigates two methods for determining the location of a contact on a robot, both of which use measurements/estimates of external joint torques. The first method attempts to solve for two contact forces on the robot which minimize the error between the expected and measured external forces. This method is limited to only 2 possible contacts, and does not consider the shape of the robot. The second method considers a grid of points on the robot's surface. For each point, the optimal force applied to the point which would match the observed external torques is calculated, through formulation as a quadratic optimization problem. The point/force with the greatest probability is chosen. This method has the advantage that it determines nearly the exact location on the robot of a contact, an can be expanded to multiple contact points, but can be more computationally expensive. Of note is that both methods only make use of external torque estimates.

Haddadin et al. [42] note that a collision will not introduce external torques onto robot joints further away from the robot base than the point of collision. Thus, comparing estimates of external torque to find the 'farthest' link with an external torque present will allow one to identify that link as the link in collision. [42] then discussed geometric approaches for identifying the specific location of the contact, when possible. This provides a simple method of isolating the colliding link, and collision location using only estimates of external torques.

Modifications to Observers

Haddadin et al. [40] discuss two modifications to the momentum observer.

The first is to apply a high-pass filter to the residuals, in order to eliminate modelling error. Haddadin et al. argue that the robot dynamics would contain low-frequency signals, relative to torques generated by impact. These signals would then be removed by high-pass filtering. Similar strategies are also used by [38].

The second is to use the difference between momentum observer residuals and estimates of external torque made using the desired trajectory as the signal of interest. Once again, [40] notes that this requires a stiff controller in order to function well.

De Luca and Flacco [26] modify the momentum observer by removing the forces measured by a force-torque sensor located on the robot's end effector. This allows forces applied at the end-effect to be considered intentional, and discarded from the momentum observer residuals, while other external forces can still be detected. This would not, however, be useful for detecting collisions occurring at the end-effector.

Vorndamme et al. [73] use estimates of external joint torques combined with force-torque sensors to detect collisions on a humanoid robot. [73] considers situations with force-torque sensors placed on the middle or ends of the robot limbs. Collision detect is done via thresholding, both of external joint estimations, and external sensed wrenches at the force-torque sensors (corrected for any robot dynamics). The methods used by [73] require either the measurement or estimation of joint acceleration values. The collision detection method itself does not appear to have been tested independently, although simulated tests were run on collision isolation (described below). High accuracy in localization is reported when the Jacobian of the contact point is high-rank, with decreasing accuracy at decreasing ranks.

Cho et al. [18] use a modified version of the momentum observer [19] to distinguish intended contacts and collisions. This new observer takes the form of a high-pass filtered version of the momentum observer residuals. The justification offered is similar to that from [38]: external forces are likely to be applied at much slower rates during interactions than during collisions. [18] note that this method in and of itself will not allow a robot to perform a tasks which includes high-frequency forces, such as many manufacturing tasks. In these cases, more information may be required. Unlike [38], this method also relies on an accurate dynamic model of the robot.

Energy Observer

Haddadin et al. [42] note that collisions should be expected to change the energy of a robotic system, making the observation of energy in the system a potentially useful method of detecting collisions. The initial equation of the energy observer was presented as:

$$r(t) = Ko(\hat{E}(t) - \int_0^t (\dot{q}^T \tau_m + r)ds - \hat{E}(0)), \quad r(0) = 0 \tag{2.20}$$

With [42]:

- residual values r
- gain Ko
- Estimate of total energy $\hat{E(t)}$

[42] modified the equation to take the form:

$$r(t) = k_o(\hat{T}(t) - \int_0^t (\dot{q}^T(\tau_m + \hat{g}(q)) + r))ds - \hat{T}(0)) \tag{2.21}$$

With:

- Estimate of kinetic energy $\hat{T}(t)$
- Estimate of gravity vector $\hat{g}(q)$

Haddadin et al. [42] note that this method will not detect collisions in cases where the robot is not in motion (ie: $\dot{q} = 0$). In addition, collisions where the external force lies perpendicular to the robot's motion at that point will not be detected. However, [42] note that the estimation of power transferred to/from the robot may still be useful when making decisions related to the collision. The measurement of kinetic energy in the robot may be useful in determining what type of interaction has occurred. However, the inability to detect external forces when the robot is not moving, or is otherwise perpendicular to the robot's motion, presents a large weakness to this method. As suggested by [42], this method may be most useful for providing additional information after the detection of a collision or interaction.

2.3.2 Machine Learning Methods

This section reviews works which use machine learning in physical human-robot interaction, particularly detecting collisions, and extracting information about collision events. The works are divided by the type of model used. A summary of these methods is presented at the end of the section, including details on the inputs and outputs of the models, dataset sizes, and accuracies.

Support Vector Machines

Golz et al. [39] use a support vector machine (SVM) to distinguish between collisions and interactions. To do this, several nonlinear features are calculated from estimates of external joint torque, and used as inputs to the SVM. A SVM was trained for each joint, with the predictions from all joints accumulated for a final prediction. SVMs perform well in high-dimensional spaces, do not require large amounts of data for training, and can provide fast classification [39], making them useful for live classification and rapid prototyping. [39] trained SVMS to both distinguish between collision and interaction, and to classify a body part that the robot collided with. This paper lists many nonlinear features which may be useful in performing classification. However, the use of each individual feature is not justified.

While works such as [31], [75], [60] attempt to detect collisions on the basis of the entire arm, [39] performs classification on a per-joint basis and amalgamates results.

Losey et al. [50] review attempts at intent detection, focusing on healthcare applications. Among these, they provide an example of an SVM being used in the control of prosthetics.

Hidden Markov Models

DiLello et al. [31] use a sticky-Hierarchical Dirichlet Process Hidden Markov Model to learn expected force/torque sensor readings for a particular task. In this case, [31] took the measurements of each state to be the force and torque measurements of the robot end-effector, along with their first derivatives. This method is able to detect collisions by computing the probability of observing a given set of sensor measurements over a nominal execution period. If this probability is below some threshold, this can indicate a collision has occurred [31]. Classes are further used to predict the type of object collided with. The use of derivatives supports the idea that the frequency or speed of events carries important information.

Wu et al. [75] used a similar approach; a HMM was used to model sensor observations while completing a task. In this case, however, the cartesian position of the robot was combined with end-effector wrench measurements as the measurement from each state. [75] also detected anomalies based on the cumulative log-likelihood of each action. If the log-likelihood drops below a threshold, it is considered an anomaly.

A weakness of both of these methods is that "Models are trained for individual skills..." [75]. This presents limitations to making this method scale and generalize to new tasks. In addition, both of these methods made use of force-torque sensor measurements. These are sensors that not all robot platforms possess.

Neural Networks

Popov et al. [60] designed several neural networks for use in collision detection. The first network distinguished between collision and non-collision. It used 7 input neurons, each receiving estimated external joint torque values, and a layer with 14 hidden neurons. As shown in [60], the network was able to out-perform a simple thresholding method in terms of classification accuracy.

The improved performance from using a neural network suggests that there may be similar improvements to be seen using more advanced ML solutions for collision classification. It is also worth noting that only estimated external joint torques were used as input. Different values, or filtered values, may be able to provide more useful information if torques alone do not contain enough information.

Next, Popov et al. [60] also used two neural networks to identify the link, and the location along the link, where a collision was occurring. These networks contained different numbers of hidden layers, but both accepted external torques as input. Once again, improvement was seen compared to a thresholding solution similar to that described by [42]. However, this method only isolates the collision location in terms of a scalar distance from the base of the robot, and will not identify the exact location in the way the geometrical approaches in [42] might.

Finally, [60] used elements of the Fast Fourier Transform of the external torques as input to a network. This network was able to distinguish between hard and soft collision objects. It is also noteworthy that [60] used results of the FFT calculated on external joint torques as opposed to the torques themselves. This suggests the torque values alone did not provide the best information for the classification, and supports the use of other nonlinear features in [39], and their investigation in this thesis.

Lu et al. [51] used a neural network to detect collisions, and predict contact forces and locations on a robot. The neural network takes readings from two force-torque sensors and robot joint angles across a short time window as inputs. The outputs are the position and contact force of contacts on various links of the robot. When compared against a model-based approach, the neural-network based approach had lower error. However, [51] notes that a large amount of training data is required. In addition, the circumstances this was tested under were limited, with only one or two joints of the robot in motion. A model-based approach to determine the contact location was also developed, which produced a higher error in estimating the contact position/force.

Sharkawy and Aspragathos [64] use a neural network to detect collisions. This network is trained to use errors in joint position and joint velocities to predict the external torques acting on the robot. This network was trained against external torques estimated by a Kuka LWR controller. It's notable that this method was able to bypass reliance on a robot model, and this method also demonstrates the use of joint velocity in collision detect as a useful signal. However, it relied on information from joint torque sensors to train the network. The network was also only trained to perform this classification with one joint of the robot active.

Briquet-Kerestedjian et al. [12] use a neural network to distinguish collisions from interactions or no-contact situations, and localize contact to the upper or lower arm. This network was trained on joint velocity data, and estimates of external joint torques. The network was designed with sub-networks for each joint's input data, with 10 hidden-layer neurons for each joint fed into the output layer. Data was sampled with a period of 40ms, and a history of 40 samples, or 160ms was used for training and classification. This work can provide a useful reference for possible rates and timeframe of data required to perform good estimation. While the frequencies that might be needed to capture and classify events will vary depending on robot, situation, and motion, this provides a starting point of values to consider. It is also notable that [12] made an attempt to avoid model-based and frequency-based features, such as those used by Golz et al. [39].

Cremer et al. [24] perform intent estimation by using a neural network with radial basis functions to estimate a human's desired trajectory for the robot in task-space. Inputs to the network were the force applied by the human, and the current end-effector position and velocity in cartesian space. Weights for the network were learned online, and allowed for smooth guiding of a robot arm, along with a learned inner-loop controller. It is motivating to see the cartesian pose and velocity used as valuable inputs, along with the external force applied to the robot,as in [75]. However, the goal of this project will be classification of intent, as opposed to inferring an intended trajectory of the robot.

Learning Expected Error

Bolotnikova et al. [11] trained a binary tree predictor to estimate the expected tracking error when following a trajectory. The difference between measured and expected error is used as a collision detection signal, indicating a collision when it passes a given threshold. This method also demonstrates ML algorithms used to compensate for a lack of sensors. However, this method was tested in relatively limited circumstances.

The model used by [11] was trained through motions of each joint, while the joints immediately before and after it were also moved. The authors did not specify if a separate model was trained for each joint, or if this simply represented different testing scenarios.

Summary of Machine Learning Methods

Table 2.1 presents a summary of the machine learning methods.

Table 2.1: Related works using machine learning methods

Reference	Model	Model Input	Classes	Dataset Size	Accuracy
Golz et al. [39]	SVM	External Torques Nonlinear Features	Collision/Interaction	65 Samples	97%
DiLello et al. [31]	HMM	Force-Torque Derivatives	Nominal Operation 4 Objects	10-Trial Batches	91%
Wu et al. [75]	HMM	Force-Torque Cartesian Position	5 Tasks	10 Trials	85-88% Task Success
Popov et al. [60]	NN	External Torques FFT Values	Collision/Nominal Contact Link (7) Hard/Soft	103 min 300 Traj. 500 Events	99.2% 94.3% 89.5%
Lu et al. [51]	NN	Force-Torque x 2 Joint Angle History	Collision Forces+ Collision Positions on each link (7x6)	Unknown	<4N Forces <0.022m Pos
Sharkawy and Aspragathos [64]	NN	Joint Position Error Joint Velocity Measured Torques	External Torques (7)	56358 In/Out Pairs	84%
Briquet-Kerestedjian et al. [12]	NN	Joint Velocity External Torques (Histories)	Lower/Upper Arm $\times$ Collision/Interaction +Nominal	310,938 Samples	93.04% Collision 100% Non-Collision
Cremer et al. [24]	NN	End Effector Position End Effector Velocity End Effector Force	Desired Trajectory Human Stiffness/Damping	360 Trials Multiple segments	Good Tracking
Bolotnikova et al. [11]	Binary Tree Predictor	Desired Velocity Desired Acceleration Desired Load Torque	Position Error (Multiple £'s of Joints)	7 Recordings	84.6%

Table 2.1: Related works using machine learning methods

Reference	Notes
Golz et al. [39]	Per-Joint Classification
DiLello et al. [31]	Uses F/T sensor Speed-like informaton used
Wu et al. [75]	Trained for each action
Popov et al. [60]	Use of torques and features Multiple NNs used
Lu et al. [51]	Claims to require large amount of data
Sharkawy and Aspragathos [64]	External torques needed for training
Briquet-Kerestedjian et al. [12]	Per-Joint sub-networks
Cremer et al. [24]	Online training
Bolotnikova et al. [11]	Perhaps per-joint classification

2.3.3 Other Methods

This subsection reviews works which do not clearly fall into the previous categories. These papers use rates of change of raw sensor values, or make use of external sensors, without using them within an ML model.

Geravand et al. [38] use motor currents to detect collisions. They do this by measuring the currents, and applying both high-pass (HPF) and low-pass (LPF) filters. The reasoning behind this is that a LPF will remove high-frequency noise present in sensor measurements, but retain the effects of low-frequency modelling error and interactions. In contrast a HPF will remove low-frequency components of interactions, but retain high-frequency collisions. A connection can be made to [40], where filtering is suggested as a means to ignore torque estimate errors introduced by modelling error.

Je at al. [45] proposes a collision signal based on the change in recent motor currents. This was shown to allow collisions to be detected on a 2-dof robot arm, without any knowledge of the robot model needed. However, this was not tested on a more complicated arm, and may not provide enough information to distinguish other information about an interaction.

This use of motor currents provides a potentially useful method of detecting collisions when joint torque sensors are not available, such as in the TALOS humanoid robot [56].

De Luca and Flacco [26] use a vision sensor to determine if a contact is likely intentional or not. This is done by visually estimating the location of a contact, and determining if it is within an allowed or expected interaction zone on the robot. This method would, however, require a vision sensor in the workspace, and would rely on a pre-known model of how humans are likely to interact with the robot.

2.3.4 Sensors Used

Table 2.2 presents a summary of different types of sensors used in the previous works presented. A large variety of sensors have been used in collision detection, ranging from simple joint position sensors, to external force-torque sensors mounted on a robot. The most common are sensors mounted at robot joints, such as joint position and torque sensors.

Table 2.2: Sensors used in collision detection

Sensor Type	Used In	How Used	Advantages and Disadvantages
Position and Velocity	[44], [28] [25], [11] [51], [64] [12], [11]	Compare vs expected Used in other calculations	Simple sensors Common Requires displacement
Motor Current	[38], [45]	Filtering Changes in current	May be present when torque sensors are not [56]
Torque Sensors	[42], [76] [70], [5] [39], [60]	Estimation of external forces Used with inverse dynamics	Direct measurement of joint forces Can integrate force and motion control [30] Not always present [56] Additional system complexity and cost [30]
Force-Torque	[66], [75] [31], [51] [24]	End-effector or base-mounted Model-based or ML processing	Directional information May not be present May miss forces depending on placement in arm
Skin	[68], [21]	Localized detection	Can localize detection Force Direction Variable Placement Typically small in size Risk of damage during collision
Capacitive	[47]	Sensors placed inside arm Localized detection	Few required Safe from damage Required designing sensors for use inside specific arm
Vision	[57], [32], [26]	Collision avoidance Identifying intentional contacts	Can be used preemptively Must be installed Large amount of processing required

2.3.5 Summary

A large number of model-based methods rely on the estimation and thresholding of external joint torques. These are in turn often estimated through the use of the momentum observer. This method is likely be be useful in providing contextual information to the robot, and may be modified in many ways.

A variety of machine learning methods have also been proposed. Many make use of raw sensor information to detect collisions, or the calculation of external joint torques. Some methods make additional use of derivatives of sensor values, while others calculate nonlinear features for classification.

Distinguishing collisions from interactions can be done simply using filtering, or as part of machine learning classification. The ability to make this distinction using filtering is likely a result of the different events occurring at different frequencies, allowing signals induced by one or the other to be filtered out. The methods used to distinguish collisions from interactions tend to focus on just these two classes, and not varying types of interaction, which this thesis explores.

Both model-based and machine-learning-based methods have been used for determining the location of a collision on a robot. Geometric approaches are greatly aided with the presence of additional sensors, such as force-torque sensors. Machine learning methods tend to make use of these sensors as well. The location of a collision or contact on the robot may provide useful information in classifying an interaction. However, ML methods used to determine this tend to use simplified models or require many sensors, and full geometric methods may not have enough sensors present, or can be computationally intensive.

An estimate of the external force applied to the robot can be found using external torque estimates and Equation 2.7. To the author's knowledge, the transfer of this into work-space, in the absence of end-effector force-torque sensors, has not been investigated in the works discussed in the form of machine learning. While a neural network may be expected to learn applicable portions of this transformation, providing the network with pre-transformed inputs may be beneficial.

2.4 Limitations and Areas for Development

Beyond distinguishing collisions from interactions, few works attempt to classify forms of intent behind an interaction. Such knowledge may be useful in selecting an appropriate robot response. Whereas other works differentiated between collisions and interactions, this thesis will attempt to differentiate between interactions of different types.

A wide variety of sensors have been used, with position/velocity and torque sensors being the most common. The use of model-based features such as the momentum observer is also common. A comparative investigation into the values that provide the most useful information may be beneficial to future researchers, by allowing smaller sets of sensors to be selected with confidence that they will provide enough data.

Similarly, the use of different features should be compared. This can determine which features provide the most useful information. For example, [60] used the FFT in classifying hard/soft contacts, while the FFT was just one of many features used by [39]. A deeper analysis will reveal which features are most useful.

Previous works have shown both the use of raw sensor measurements, and of features calculated from a dynamics model of the robot. The use of raw sensor data compared to observer-based features is investigated in this thesis.

Chapter 3

Problem Formulation and Setup

This chapter describes the problem formulation and data generation/preparation for the methods described in Chapters 4 and 5.

3.1 Problem Formulation

The overall objective of this thesis is to develop ML methods to classify different types of physical interaction between a human and robot. Extracting additional information, such as interaction location, is considered out of scope for this thesis.

The classifier developed will be used to differentiate between one of four classes. These classes were chosen to cover a range of possible ways a human might interact with a robot, and which might require different responses. Each class is presented below with a motivating/illustrating scenario:

None: No interaction. Nominal operation of the robot arm

Bump: Light accidental contact. Might occur during close collaborative tasks. May or may not require a response from the arm

Adjust: Gentle adjustment of the robot to provide feedback. Might occur to guide the robot to correct errors in end-effector location, or to slightly manipulate trajectory during motion. Might be appropriate for the robot to adapt a compliant controller to be guided effectively

Shove: Vigorous movement of the arm to prevent collision, damage, or injury. Might occur if robot is moving rapidly and likely to damage the human or environment. Note that this class represents an intentional action meant to precede and prevent a collision. Might be appropriate for the robot to recoil or halt motion to prevent collision

More formally, following notation from [59], the problem is defined as estimating the label y_i of a sample of data x_i, with $y_i \in \{1, 2, 3, 4\}$ indexing each of the above classes, and $x_i \in X \subseteq \mathbb{R}^{s\times T}$, where s is the number of sensors recording data during the sample, and T is the number of timesteps recorded. The machine learning problem is to learn a mapping $g : X \to Y$. This is a supervised learning problem, with a dataset $D = \{(x_i, y_i), i = 1...N\}$, which will be divided into training and testing sets. Note that g may include both a ML model, as well as any processing of data.

Classifiers will be evaluated on their accuracy over a selection of M samples used as a testing set. To understand the classifier performance, the confusion matrix of the classifier $\mathbf{M}$ will also be used. Given a set of labels in the test set $\{y_i, i = 1...M\}$, and corresponding predictions $\{y_i^*, i = 1...M\}$, these are calculated as:

$$Acc = \frac{1}{M}\Sigma_{i=1}^{M}\mathbb{1}(y_i^* = y_i)$$

$$\mathbf{M}_{ij} = \Sigma_{i-1}^{M}\mathbb{1}\{y_i = i\}\mathbb{1}\{y_i^* = j\}$$

The time required to calculate features for classification is discussed in Appendix B. However, it is not a key consideration in this thesis.

3.2 Dataset Generation

This section discusses the generation of a dataset used for training and evaluating classifiers.

3.2.1 Robot Details

The Franka Emika Panda [33] is a robot arm with seven actuated joints, and a gripper attachment. Each joint includes position, velocity, and torque sensing. The Panda is designed to allow safe interaction with humans. An image of the Panda arm is shown in Figure 3.1.

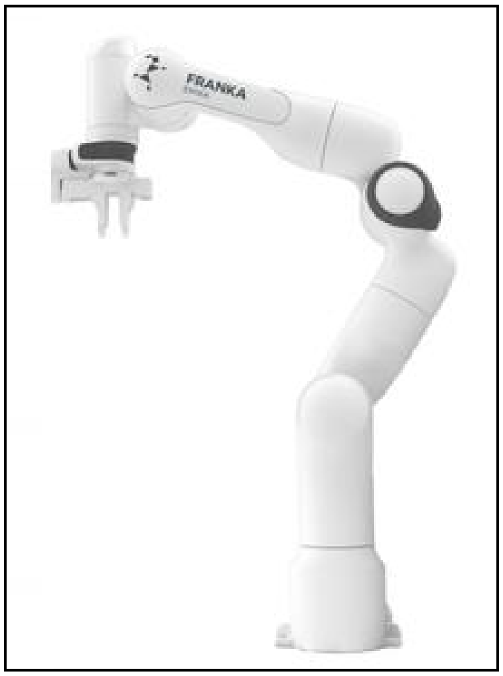

Figure 3.1: Panda robot arm [74]

When calculating dynamic forces acting on the Panda, dynamic parameters calculated using the methods provided in [36] were used.

3.2.2 Experimental Conditions

Two experimental conditions were used to generate datasets.

In the first condition, the robot held a static pose, with no intentional movement. The human then performed an action on the robot. Data gathered under this condition forms what is referred to as the 'static' dataset.

In the second condition, the robot moved between two given poses. The human performed an action on the robot in the middle of this motion. Data gathered under this condition forms what is referred to as the 'dynamic' dataset.

3.2.3 Data Recorded

During operation, data is recorded from the joint sensors of the Panda. These sensors provide information on joint position, velocity, and measured torque, as well as timestamps. The data was recorded at approximately 600 Hz. All of this information is collected into timeseries during data recording.

3.2.4 Sample Gathering

The datasets were generated by the author interacting with the Panda robot arm in various poses or during motion between poses. The poses of the robot arm were chosen to attempt to achieve wide coverage of the workspace, with joint values shown in Table 3.1. For the static dataset, these poses were used directly. For the dynamic dataset, the robot moved from pose A to the other poses.

Table 3.1: Joint positions of robot poses (units in radians)

Pose Name	q_1	q_2	q_3	q_4	q_5	q_6	q_7
A	0.0	-0.78	0.0	-2.36	0.0	1.57	0.78
B	-1.92	-0.26	2.72	-1.71	0.46	2.8	-1.0
C	0.24	1.0	0.37	-0.63	-0.34	2.90	0.84
D	-0.8	1.4	1.0	-1.5	-1.5	3.75	2.0
E	-1.0	0	1.8	-2.8	-0.1	3.0	0.75
F	-0.4	1.6	1.3	-1.9	-0.4	1.8	-0.8
G	0.05	0.32	0.34	-0.89	-1.52	3.2	2.3
H	0	1.0	0.85	-0.8	-1.83	2.5	1.0

For the purposes of this thesis, a 'sample' refers to one session of recording data during interaction between a human and the robot arm, during which sensor data on joint positions, velocities, and torques was gathered, along with timestamps. Along with the known interaction performed, these form the pair (x_i, y_i). Each sample gathered is transformed into one input/output pair for machine learning classifiers through the processing described below, and any additional processing needed for an individual classifier. The process for gathering a sample is as follows:

1. Place arm in position for a given pose, according to Table 3.1
2. Begin recording sensor information (joint positions, velocities, and torques)

3. Begin robot arm motion (if applicable)
4. Interaction performed by author
5. Robot motion halts
6. Data recording ended

Each sample was gathered with a specified action type. Typically, 3 repetitions of each of these cases were gathered. During each recording, an interaction type from Section 3.1 was pre-assigned, and performed by the author during data recording. An interaction time was also pre-planned and stored in meta-data.

While gathering the static dataset, interactions were performed with specified links of the arm. Each pose from Table 3.1 was used in generating this dataset.

While gathering the dynamic dataset on the Panda robot arm, the speed of motion between poses was also varied. In this dataset, there was no specification of a given link to interact with; the link was chosen during the action by the author, and was not recorded. Movements occurred from pose 'A' from Table 3.1 to every other pose.

3.3 Data Preprocessing

Before being used in classification, the data from each dataset is pre-processed.

3.3.1 Selecting Window of Data

The first step in data preprocessing is selecting a window of data from each sample in which the given interaction is known or estimated to have occurred. Appendix A describes in detail the development and testing of methods to identify the best size window. It was found that using the planned interaction time performed optimally for the dynamic dataset. A programmatic method performed better for the static dataset, as described in Appendix A.

3.3.2 Calculation of Features

The second step in data preprocessing is the calculation of the features to be used by a classifier. Two types of features are discussed in this thesis, and are presented in Table 3.2.

Table 3.2: Feature types used in this thesis

Type	Description	Examples
Timeseries Features	Data that exists at each timestep Recorded raw or calculated	Joint position $q(t)$ Joint torque $\tau(t)$
Scalar Features	Calculated from timeseries features May be a single value (ex: mean) May be a finite number of values (ex: FFT)	Mean of joint position $q(t)$ FFT coefficients of $\tau(t)$

Timeseries Features

Joint acceleration and power are calculated trivially from the raw data using finite differences and element-wise multiplication of velocity and torque, respectively. Together with the raw data, these are referred to as 'sensor-based' features.

Additional features are calculated using a dynamic model of the Panda robot arm, and are referred to as 'model-based' features. These are calculated using the Pinocchio rigid body dynamics library [16] [15]. Table 3.3 presents these features, and their method of calculation. Features are calculated for each joint i.

Table 3.3: Calculated timeseries features

Feature	Notation For Joint i	Calculation of Feature
Momentum Observer Residuals	$r_i(t)$	Equation (2.16)
Gravity Compensation Toque	$g_i(t)$	$g(q)$ in Equation (2.9)
Inverse Dynamics Torque	$\tau_{inv,i}$	τ_{tot} in Equation (2.9) With 0 external forces
External Power	$P_{ext,i}$	$P_{ext,i} = r_i(t) * v_i(t)$

The momentum observer residual and inverse dynamics torque features are considered based on their use in literature, such as [42], [76], [70], [5]. The gravity compensation torque and external power features were also considered, due to ease of implementation.

Finally, the momentum observer residuals are projected into the workspace, to estimate the 6-dof external force applied to the robot. This follows the equation:

$$f(t) = (J(q)^T)^{-1} r(t) \tag{3.1}$$

The sets of 3 forces, 3 moments, and 2 magnitudes of each, are termed 'workspace' timeseries features. These features are intended to capture information which may be relevant in workspace, and not joint-space.

Scalar Features

Scalar features are features calculated over a given timeseries feature. The majority of these features were chosen based on their use in [39]. Many of these features are calculated using the mean and standard deviation [23] of a timeseries feature, given below for a timeseries feature $s(t)$ with N timesteps:

$$\bar{s} = \frac{1}{N}\sum_{i=1}^{N} s_i \tag{3.2}$$

$$\sigma_s = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(s_i - \bar{s})^2} \tag{3.3}$$

Table 3.4 presents the scalar features used in classification, as well as methods for calculating them, with notation also taken from [39]. This considers each feature to be calculated from a timeseries feature $s(t)$ with N timesteps. Most features are calculated using the numpy [55] or scipy [71] libraries, or based on the definition provided.

Table 3.4: Calculated scalar features

Feature	Calculation of Feature
Mean	Equation (3.2)
Variance	σ_s^2
Range	$\max(s(t)) - \min(s(t))$
Delta	$s_N - s_0$
Skew	$\frac{\sum_{i=1}^{N}(s_i - \bar{s})}{N\sigma_s^3}$
Energy	$\sum_{t=0}^{N} s(t)^2$
Hjorth Mobility Hjorth Complexity [9] [39]	$Mob(s) = \sqrt{\frac{Variance(\dot{s})}{Variance(s)}}$ $Comp(s) = \frac{Mob(\dot{s})}{Mob(s)}$ [53] [39] Implemented with pyeeg [8]
Fast Fourier Transform	$FFT(s) = y$ such that $y[k] = \Sigma_{n=0}^{N-1} e^{-2\pi j \frac{kn}{N}} s[n]$ Calculated using scipy [71] fftpack [72]

Typically, every scalar feature is calculated for every timeseries feature.

3.4 Final Dataset Details

Table 3.5 provides the final details of the datasets generated.

Table 3.5: Details of generated datasets

Dataset	Samples	Poses	Actions	Links
Static	334	8 Poses 41-42 samples each	96 each of Bump/Adjust/Shove 46 None	72 samples for each link 46 no interaction
Dynamic	255	7 Movements 30-45 samples each	51 each of Bump/Adjust/Shove 102 None	Not Recorded

Figure 3.2 shows example force profiles taken during the generation of the static dataset. They show the magnitude of the estimated external force acting on the hand link of the

Panda robot arm. These estimates were generated by using the Momentum Observer given by Equation 2.16 to estimate external torques, and Equation 2.7 to calculate external forces. One can see that these profiles differ in both magnitude and shape of the profile. Note that some estimated external forces are expected even in the 'None' case, due to any existing modelling errors.

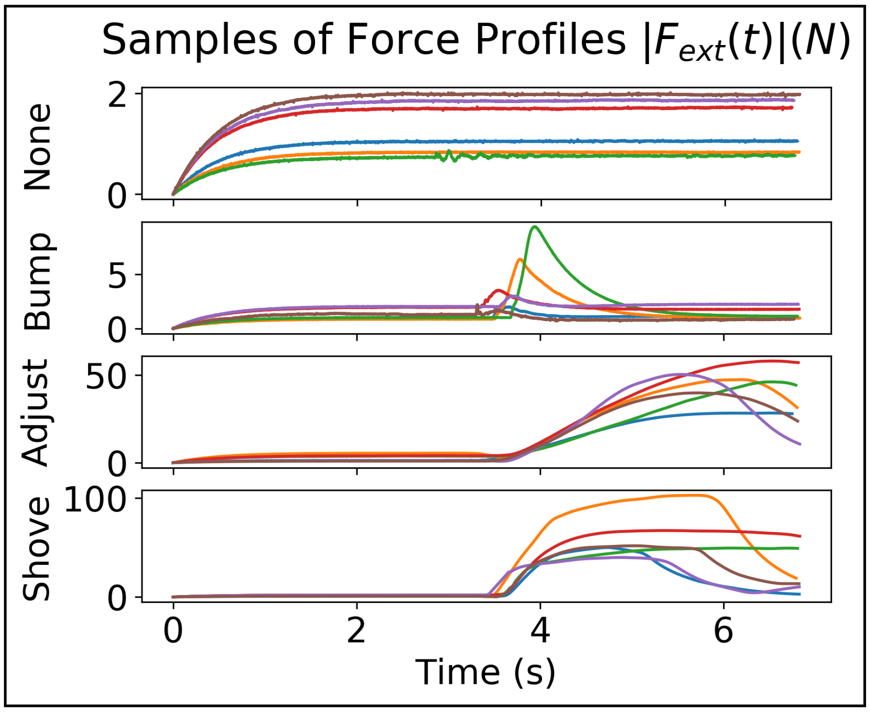

Figure 3.2: Samples of force profiles taken during actions of the provided classes. Each sample is shown on a separate line in the appropriate subplot.

Chapter 4

Support Vector Machine based human-robot contact classification

This chapter describes the use of a Support Vector Machine (SVM) to classify human-robot contact types. The use of a SVM is motivated by the work of [39], as well as the ability of SVMs to perform well with limited amounts of data.

The chapter describes the process used to identify the timeseries and scalar features which can be used by an SVM to reach optimal performance, showing that using a combination of sensor-based and model-based timeseries features, including frequency information, will lead to the best performance.

A classifier using information from all joints is compared to per-joint methods, motivated by [39], and [12]. A method to further increase performance over these approaches is introduced.

Section 4.1 presents the process for performing feature identification. Section 4.2 presents the process for testing per-joint classifiers, as used in [39]. Section 4.2.2 presents an extension to these per-joint methods. Finally, Section 4.3 presents common elements to testing and evaluating a SVM, which are used to evaluate each of the testing conditions.

4.1 Identifying Useful Features

Feature-identification tests are performed to identify features that will perform well when used to train an SVM. Features that lead to a high classification accuracy are desired. In addition, identifying a small number of features which can be quickly computed is also desired.

An evaluation of both timeseries and scalar features is described and presented in Appendix B. The results shown have identified several useful timeseries features, including joint torques, joint velocities, and momentum observer residuals. In addition, the Range scalar feature was shown to out-perform other features.

In order to test the benefits of comparing sensor- and model- based features, several combinations of timeseries features are tested. During each of these tests, the Range scalar feature will be used, as per Appendix B. The following timeseries features will be used in training and evaluating SVMs:

1. Joint velocities (sensor-based)
2. Joint torques (sensor-based)
3. Momentum observer residuals (model-based)
4. All of the above (combination)

The results of these tests are shown and discussed in Section 6.1.1.

4.2 Comparing Whole-Arm and Per-Joint Methods

In previous work, classifiers for each individual joint have also been reported, using SVMs in [39], and the use of sub-networks connected to individual joints in [12].

The work in this thesis generally performs classification on a whole-arm basis, combining information from every joint with one classifier. This section describes the comparison of the whole-arm approach, and per-joint approach. In addition, this section presents a new approach to per-joint methods.

4.2.1 Whole-Arm and Per-Joint Comparison

The process for comparing whole-arm and per-joint approaches takes 3 phases. The dataset used is held consistent across all phases.

First, a SVM classifier will be trained using data for the entire arm. This will be used for comparison later.

Next, 7 classifiers are trained, each using data related to only a single joint of the robot arm. In cases of using model-based information such as the momentum observer, values are calculated using information from the entire arm, and the element corresponding to each individual joint is extracted for use. These classifiers are trained using the procedure described in Section 4.3.3, with 25 classifiers trained to generate a distribution of accuracies. The trained classifier for each joint is used to perform classification on the entire dataset to produce probabilities for each class.

Finally, following the method of [39], each classifier is used to predict the probability that a sample $x_{i,j}$ belongs to each class, $p_j(class(x_{i,j}) = c)$, where i indexes the sample number, and j indexes each classifier. This uses the scikit-learn SVC's *predict_log_proba* function, which uses cross-validation to train a probability model for classification [63]. The probability predicted for each class is summed over the probabilities produced by each joint's classifier. The class c for which the total probability is greatest is then taken as the final prediction.

$$y_i^* = \underset{c}{argmax} \sum_{j=1}^{7} p_j(class(x_{i,j}) = c) \tag{4.1}$$

These predicted and true classes are then used in evaluating the accuracy of the classifier. The prediction process is outlined in Algorithm 1, and an illustration is provided in Figure 4.1. Note that lines 3, 4, and 5 of Algorithm 1 are also relevant to the training of each per-joint SVM; the SVMs are trained only on data from their corresponding joints.

Algorithm 1: SVM Per-Joint Method for Classification

```
Input  : Trained SVMs; testing dataset
Output: Predicted class $y_i^*$ for each sample in dataset
1  forall Sample $s_i \in dataset$ do
2      Extract label $y_i$ of $s_i$;
3      forall Joints $j \in \{1,2,3,4,5,6,7\}$ do
4          Extract timeseries features associated with joint $j$;
5          Calculate feature vector $x_{i,j}$;
6          Select trained $SVM_j$;
7          forall Class $c \in \{1,2,3,4\}$ do
8              Use $SVM_j$ to calculate probability $p_j(class(x_{i,j}) = c)$;
9          end
10     end
11     prediction $y_i^* = \underset{c}{argmax} \sum_{j=1}^{7} p_j(class(x_{i,j}) = c)$;
12 end
```

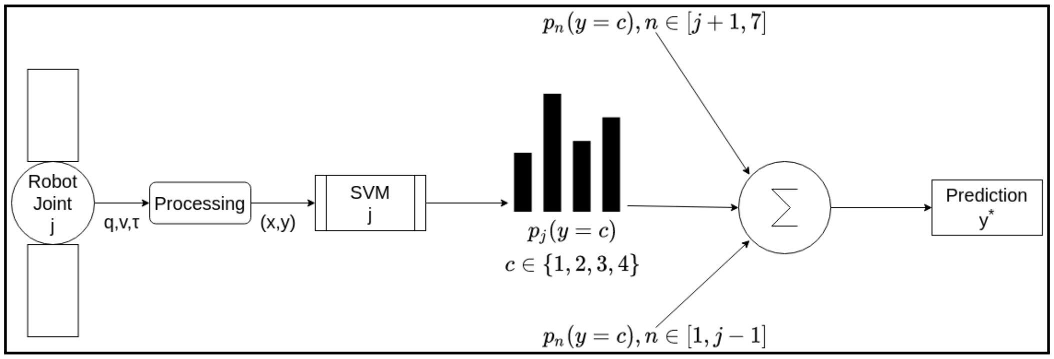

Figure 4.1: Illustration of Per-Joint SVM method

The results of these tests are shown and discussed in Section 6.1.2.

4.2.2 Extending Per-Joint Methods

This sub-section presents an extension to the per-joint method. It is believed that if features from each individual joint contain enough information to make a prediction, then using multiple joints may provide more information. Thus, a smaller number of more-accurate classifiers may provide better overall classification results.

This extension modifies the second phase of the process described above, particularly lines 3 and 4 of Algorithm 1. Instead of using data related to only a single joint of the robot arm, classifiers are trained with data from 3 consecutive joints of the robot arm (ie: classifier j uses data from joints $j-1, j, j+1$). This is repeated for all combinations of consecutive joints, for a total of 5 classifiers.

$$y_i^* = \underset{c}{argmax} \sum_{j=2}^{6} p_j(class(x_{i,j}) = c) \tag{4.2}$$

In the third phase of the process, the summation is performed over the 5 existing classifiers. The classification process is outlined in Algorithm 2, and illustrated in Figure 4.2.

Algorithm 2: SVM Modified Per-Joint Method for Classification

```
   Input   : Trained SVMs; testing dataset
   Output: Predicted lass $y_i^*$ for each sample in dataset
1  forall Sample $s_i \in dataset$ do
2      Extract label $y_i$ of $s_i$;
3      forall Index $j \in \{2,3,4,5,6\}$ do
4          Extract timeseries features associated with joint $j-1$;
5          Extract timeseries features associated with joint $j$;
6          Extract timeseries features associated with joint $j+1$;
7          Calculate feature vector $x_{i,j}$;
8          Select trained $SVM_j$;
9          forall Class $c \in \{1,2,3,4\}$ do
10             Use $SVM_j$ to calculate probability $p_j(class(x_{i,j}) = c)$;
11         end
12     end
13     prediction $y_i^* = \underset{c}{argmax} \sum_{j=1}^{7} p_j(class(x_{i,j}) = c)$;
14     compare $y_i$ and $y_i^*$
15 end
```

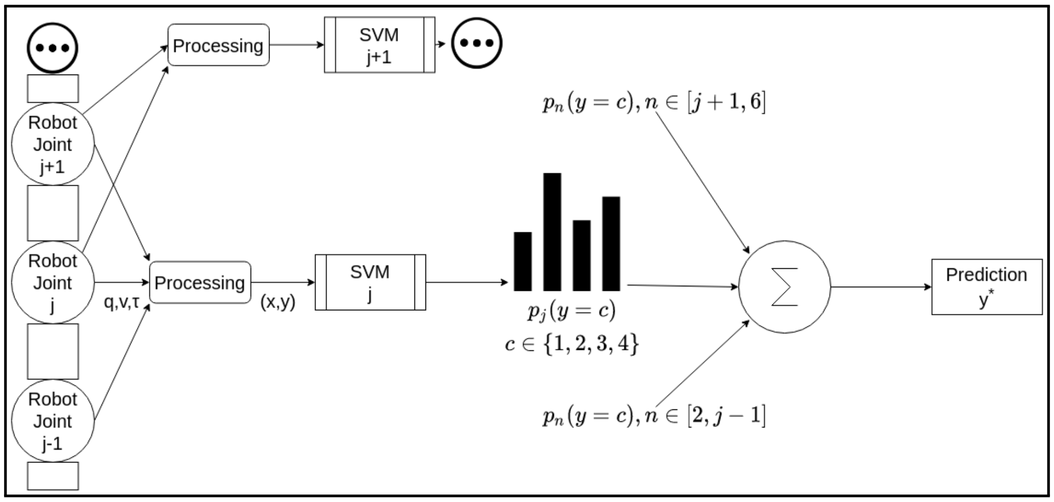

Figure 4.2: Illustration of Per-Joint SVM with new approach

The results of these tests are shown and discussed in Section 6.1.2.

4.3 Testing Methodology

This section describes common methods used in performing tests using SVMs.

4.3.1 Data Generation

The tests in this section use the datasets described in Section 3.2. During each test, the following process is followed to create a dataset useable for a SVM:

1. Select sample from dataset, $x \in X \subseteq \mathbb{R}^{s \times T}$
2. Extract window of data: $x \to x' \in \mathbb{R}^{s \times T'}, T' \leq T$
3. Select timeseries feature to be used: $x' \to x'' \in \mathbb{R}^{s' \times T'}, s' \leq s$
4. Calculate all scalar features and concatenate to form a feature vector: $x'' \to z \in \mathbb{R}^n$
5. Extract true class of sample from metadata, y

In this way, a dataset of input/output pairs (z_i, y_i) is generated. The feature vector z_i is used as input to the classifier.

4.3.2 SVM Implementation

SVMs are implemented using scikit-learn's [58] SVC implementation. In addition to this, scikit-learn's StandardScaler is used to scale training and testing data.

4.3.3 SVM Evaluation Procedure

Given a usable dataset, SVMs are trained on the data. In order to generate a robust understanding of training results, 25 SVMs will be trained. Each SVM will be assigned a random 80% of the dataset as training data, and 20% as testing data. The average accuracy of SVMs over the testing data, along with it's standard deviation, will be shown in the results.

Chapter 5

Neural Networks for Human-Robot Contact Classification

This chapter describes the use of neural networks to classify human-robot contact types, and the development of methods to augment their training data. Neural networks were chosen due to their improved expressive power over SVMs. However, neural networks often require large amounts of data to train. For example, [12] reported a training dataset of over 300,000 samples for classifying interaction type and location on a robot arm. This can present problems when it is difficult or time-consuming to gather sufficient data on a robot. This chapter presents a method of augmenting the training dataset of neural networks using data generated in simulation, in order to require less real-world training data.

This section describes the method of testing neural networks to be used, the design of a Convolutional Neural Network (CNN) and Long Short-Term Neural Network (LSTM), and the generation of a simulated dataset for use in testing.

Section 5.1 describes the process used to test training neural networks using simulated data, and the augmentation of this data. Section 5.2 describes the architectures of the networks used in each test. Finally, Section 5.3 discusses the generation of the simulated dataset used in these tests.

5.1 Training NNs with Simulated Data

Tests are performed to determine if simulated data can be used to reduce the amount of real-world data required to train the networks. A simulated dataset is generated as described below. Each network is tested under variations in the amount of real data used, the addition of simulated data, and the level of noise used to augment the simulated data. This follows the process outlined below:

Algorithm 3: Neural Network testing process

```
1 forall Percentage p in {50%, 75%, 100%} do
2     Extract p% of real dataset;
3     Train and evaluate network on real data only;
4     Add simulated data to training set; Train and evaluate network;
5     Add second copy of simulated data to training set; Train and evaluate network;
6     forall Noise level n in {1, 5, 10} do
7         Augment simulated data with noise magnitude n; Train and evaluate
            network;
8     end
9 end
```

This is intended to identify the ability to achieve good performance with less real-world data gathered and any optimal levels of noise needed for augmentation.

5.1.1 Reduction of Real Data

Tests are performed with various portions of the real-world data used by the neural networks. This is intended to show that the simulated data augments the training process such that a lesser amount of real-world data is needed. Elements of the dataset are excluded on a random basis.

5.1.2 Simulation Data and Augmentation

Tests are performed with and without simulated data added to the training dataset, as well as augmentation to the simulated data. This is indented to identify any augmentation levels which produce optimal results, and to separate the effect of the addition of data from the augmentation of data. The cases tested using the process from Algorithm 3 are:

Real: Real data only. Intended for use as a baseline comparison

1xSim: Simulation data added with no augmentation. Only one copy of the dataset is added. Intended to capture effect of adding simulated data.

2xSim: Simulation data added with no augmentation. The set of simulated data is added twice, to capture the effect of doubling the dataset. Intended to capture effect of increasing amount of simulated data without noise.

Augmenation: Simulation data added with augmentation at various joint torque noise levels. Dataset doubled as described in Section 5.3.5. Intended to capture effect of augmentation on performance.

The results of these tests are shown and discussed in Section 6.2.

5.2 Network Details

Two neural network types are developed and tested. These include a convolutional neural network (CNN), and long short-term memory network (LSTM). Both networks are implemented using the Keras library [20], and are not pre-trained.

CNN Architecture

The CNN network is designed to take an input of timeseries features, formatted in a 2D grid, and perform 2D convolutions during classification.

The network design was based on [35] and [7], which use CNNs to perform tactile object recognition and material classification, respectively. In particular, these works use similar amounts of data to the real dataset used in this thesis, and [35] uses a similar number of input channels.

Both examples used only two convolutional layers, the second having twice the filters of the first. An initial network was created with two convolutional layers of 32 and 64 kernels, and a single dense layer of 32 neurons. Max pooling in the time dimension was used after each convolutional layer. Batch normalization, leaky ReLU, and Dropout is added after every pooling and dense layer.

A grid-search is performed by modifying the architecture in the following ways:

Depth: Multiply number of layers in the network. Selected from [1, 2, 4]

Width: Multiple number of neurons in each layer. Selected from [0.5, 1, 2]

Dropout: Dropout rate in network. Selected from [0.00, 0.10, 0.25]

Inputs: Input timeseries features. Selected from [Raw sensor data, joint torques/velocities/residuals]

The grid-test identified the following values as the optimal parameters, or modifications to parameters, from the base network, shown in Table 5.1:

Table 5.1: Optimal parameters adjustments for CNN

Parameter	Value
Depth Multiplier	4
Width Multiplier	2
Dropout Rate	0.0 (0.02 used)
Inputs	Joint positions, velocities, and torques (Raw sensor data)

The optimal results appear to come from both a deep and wide network, as well as operating on raw sensor data.

The final network structure is shown in Figure 5.1.

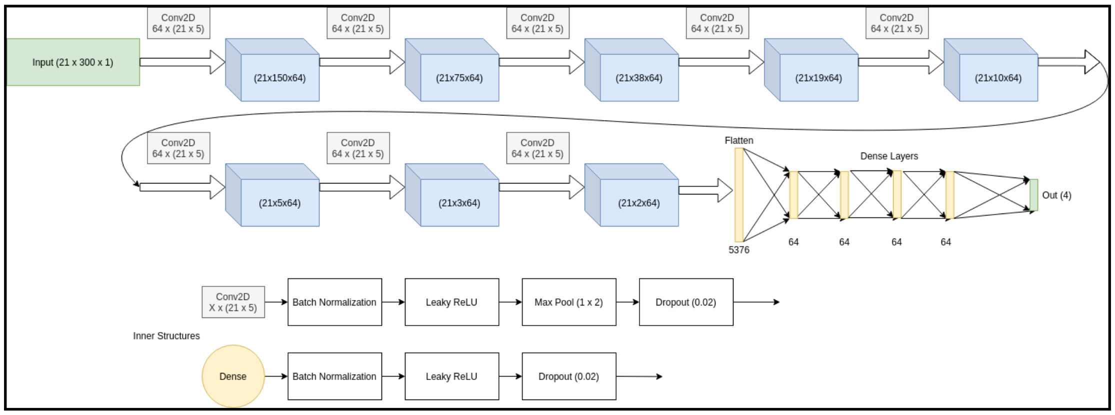

Figure 5.1: Architecture of CNN used

The width of data in each layer is only approximate, as data size depends on the particular dataset used. The data is sized to fit the largest sample of data in the dataset, with other samples zero-padded to match this size. With a window of 0.5s and a frequency of approximately 600 Hz, each sample is expected to be approximately 21 inputs by 300 timesteps in size.

LSTM Architecture

The LSTM network is designed to accept timeseries features as direct input. The network design was based on [2] and [49], which perform collision detection and inverse dynamics estimation, respectively. In particular, [49] estimates the inverse dynamics of a 7-dof robot arm.

[2] uses only 2 LSTM hidden layers, and [49] tests several combinations, with results favoring fewer, wider layers. An initial network was created with two LSTM layers of 32 and 16 layers, followed by a dense layer of 16 neurons. Batch normalization, leaky ReLU, and Dropout is added after every layer.

The same grid search as for CNN was performed for the LSTM. The grid-test identified the following values as the optimal parameters, or modifications to parameters, from the base network, shown in Table 5.2:

Table 5.2: Optimal parameter adjustments for LSTM

Parameter	Value
Depth Multiplier	2
Width Multiplier	2
Dropout Rate	0.0 (0.02 used)
Inputs	Joint velocities, torques, and residuals (Same as SVM)

The optimal results come from a deeper network, as well as a wider network. It is notable that the features chosen are those which performed best for the SVMs, and differ from those used by the CNN. While an advantage of neural networks is the ability to learn complicated functions from raw input data, examples such as [60] have demonstrated the benefit of performing some feature selection to introduce more useful features to networks.

The final network structure is shown in Figure 5.2.

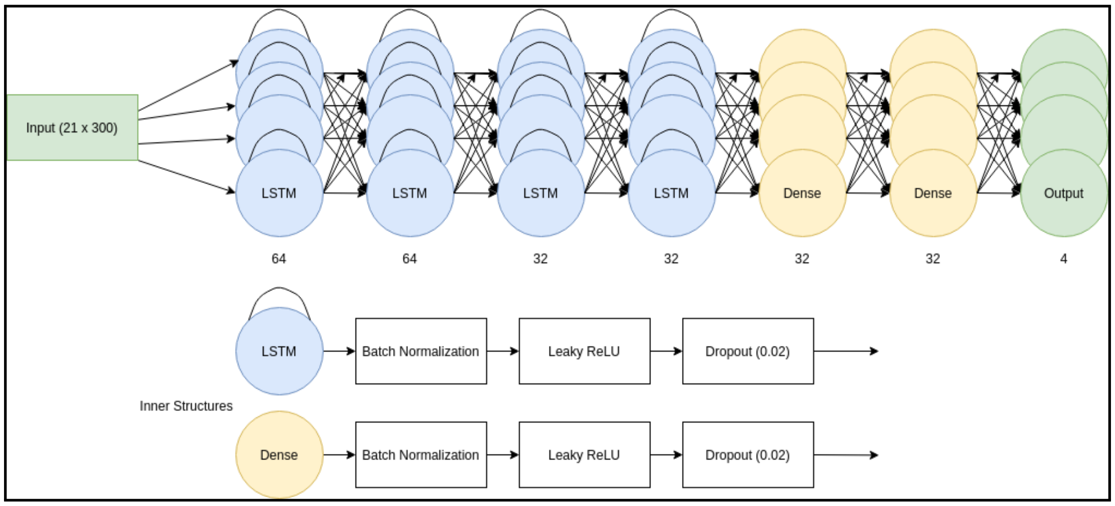

Figure 5.2: Architecture of LSTM used

The LSTM network did not require the consistent data sizing which the CNN required. However, samples are still expected to be 21 features by approximately 300 timesteps long.

5.2.1 Network Training and Testing

Both networks are trained using the Keras library. During training, data is generally used in a 70/30 train/testing split. The testing set is further divided into two sets comprising 15% of the total data each. The first is used as validation data during training, and used for early stopping and model checkpointing. The second is used to evaluate the performance of the final model. The choice of 15% of data for these sets was made to ensure both sets had a sufficient amount of data, as opposed to the 10% that each set would contain if the 80/20 division used for SVMs was repeated.

Training is done for up to 500 epochs, with early stopping active. Once training has completed, the last checkpointed model is loaded for testing.

Limited time and resources prevented performing a full k-fold cross-validation of the models. Thus, an attempt to approximate the performance of the network is made, and predict variability when training networks with random initialization of weights. It should be noted that this will likely have the effect of underestimating the effect of random initialization of network weights. During testing, random subsets of 40% of the final testing dataset are taken, and classification performance is evaluated. The average accuracy, and its standard deviation, are reported. It is hoped that this selection will provide a sufficient estimate of the variability of the network, as opposed to providing a single accuracy value based on the entire 15% of the test set.

5.3 Dataset Generation

In order to address the problem of limited data, a set of data produced in simulation is generated. Once again, two datasets are generated to mimic the static and dynamic datasets discussed in Section 3.2.

5.3.1 Simulation Dynamics and Control

The simulation is run at 1200 Hz, with joint velocities and positions updated in increments of 10 sub-steps within each period, and data recorded at 600 Hz. Dynamics are calculated using the Pinocchio rigid body dynamics library [16], [15]. The dynamics are updated using equations (2.9) and (2.8) to calculate the joint accelerations due to dynamics, external forces, and motor torques:

$$\begin{aligned} \tau_{dyn} &= C(q,\dot{q})\dot{q} + g(q) \\ \ddot{q} &= M(q)^{-1}(\tau_m + \tau_{ext} - \tau_{dyn}) \end{aligned} \tag{5.1}$$

With the acceleration values calculated, joint position and velocities are integrated for several sub-steps. Integration of joint positions is performed using the pinocchio library.

$$\begin{aligned} q{+} &= (dt_{sim}/n\_sub\_steps)\dot{q} \\ \dot{q}{+} &= (dt_{sim}/n\_sub\_steps)\ddot{q} \end{aligned} \tag{5.2}$$

The controls active on each joint were a combination of gravity compensation and PID control. Joint torque limits are enforced according to [34].

5.3.2 Robot Parameters

The parameters used for the Panda robot arm in simulation are those calculated in [36], using provided code. It should be noted that this identification did not include the hand of the robot, and an estimate of the hand's dynamic parameters was used.

5.3.3 Simulated Motion

During generation of the static dataset, the same poses used in generating the real dataset were used, as described in Table 3.1. This was done to achieve good correspondence with the real dataset. To augment the dataset, while still maintaining close correspondence to the real dataset, random Gaussian noise was added to joint positions. This noise had zero mean and a standard deviation of 0.1rad. Positions were randomly chosen during each simulated sample generation.

During generation of the dynamic dataset, the start pose and selection of end poses were also chosen to mimic the real-world dataset. Joint positions for both start and end pose were shifted by random noise with a standard deviation of 0.1 rad. A motion profile for each joint to follow was generated using a quintic trajectory, allowing for zero velocity and acceleration at each endpoint. Average joint speeds between 0.01 and 0.5 rad/s were chosen, compared to the limits of 0.1 and 0.3 rad/s of the real dataset.

The generation of each sample began with 3 seconds of holding the start position. A motion phase was executed for 4s for the static dataset, and as long as required for the dynamic dataset. External forces were applied starting at the midpoint of this period, delayed by a random time between 0 and 0.25 seconds. The correct starting time for application of external forces was recorded in metadata. Finally, an additional 5s of holding the end position was simulated and recorded, to allow and force application to finish.

5.3.4 Application of External Forces

A critical aspect of the simulation is the application of forces corresponding to correct physical interaction classes. In order to meet this requirement, force profiles were generated for each interaction type. Parameters of these profiles, such as duration and maximum value, were tuned by visual inspection so that estimates of external forces from real and simulated datasets would appear similar. Force profiles from real-world data were taken from the static dataset, and compared against the static simulated dataset. A selection of profiles is shown for each interaction type in Figure 5.3.

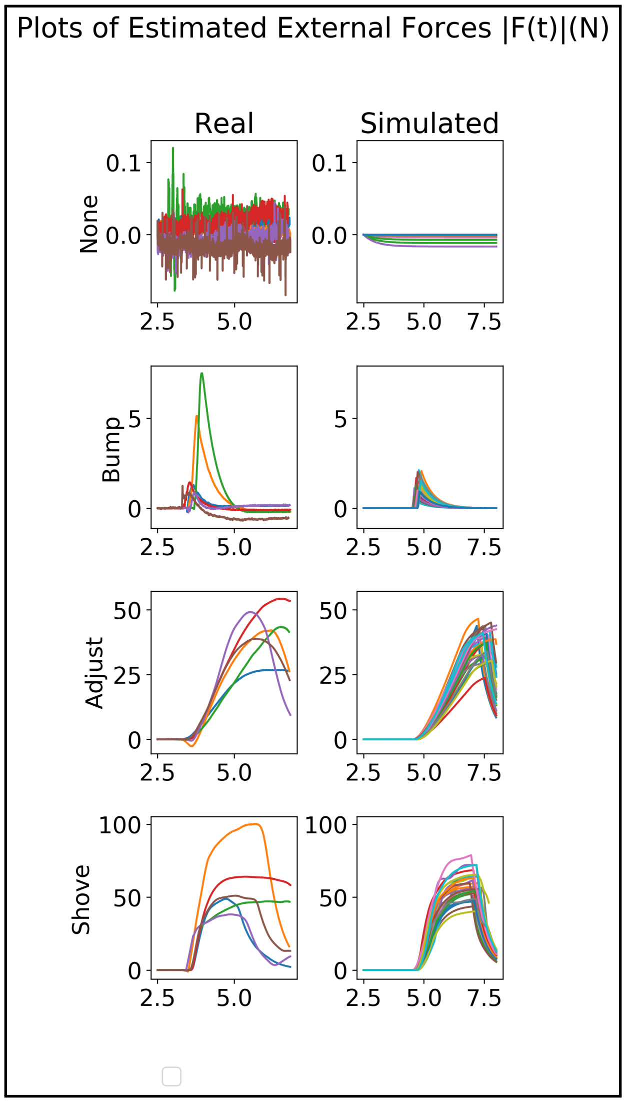

Figure 5.3: External force profiles for real and simulated datasets

It is not known why some of the 'Bump' profiles from the real-world dataset exhibit significantly larger peaks than most. This may be part of natural variation in the force applied by the author, caused by applying forces near a singularity of the robot, or due to an unknown factor. The simulated profiles appear to match well with the remaining force profiles of the 'Bump' class.

During the generation of a data sample, the created force profiles are used to generate the magnitude of an external force. This force is then applied in a random direction along the horizontal plane. The direction is chosen randomly for each sample, and held constant throughout the sample. The forces are applied to the end link of the simulated robot. This does not match the real dataset, which consisted of interactions with multiple links, and the effect of this limitation is discussed in Section 6.2.3.

5.3.5 Artificial Noise

As the simulation performed does not contain sensor noise, this noise is introduced into the data immediately before training. The noise is introduced in two forms:

Joint Torque Sensor Noise

The joint torque sensor readings were taken from static real-world dataset samples, in situations where no contact occurred. The standard deviation of these values across the entire sample was calculated, and these were averaged across all available samples. This was repeated for each joint to build a baseline of joint torque sensor noise.

These values are used to augment the simulated joint torque measurements. At each timestep, zero-mean Gaussian noise is added to the recorded torque value of each joint. The standard deviation of this noise is a multiple of the real-world torque sensor standard deviations. Various multiples are used and tested.

Momentum Observer Augmentation

In order to generate a more varied dataset, different momentum observer gain values are used. These values are chosen to be 10% higher and lower than the nominal gain value. This results in a doubling of the dataset, as two copies are produced. The momentum observer values are calculated after the torque sensor noise is added to the dataset.

In the case that the dataset is doubled without augmentation, the nominal value of the momentum observer gain is used to generate both samples.

5.3.6 Generated Dataset Details

Table 5.3 provides the final details of the simulated datasets.

Table 5.3: Details of the simulated datasets

Dataset	Samples	Poses	Actions
Simulated Static	1233	8 Poses 135-160 samples each	304-311 each of Bump/Adjust/Shove/None
Simulated Dynamic	1380	7 Movements 174-212 samples each	332-356 each of Bump/Adjust/Shove/None

Chapter 6

Results

This chapter presents the results of the tests performed. Figure 6.1 shows the overall results. The SVM demonstrates improvement when the best-performing features are chosen, which includes a combination of raw sensor-based and model-based features, but no improvement when using per-joint methods. The CNN and LSTM both demonstrate improvements when adding simulated data, and the CNN demonstrates this even when using less real-world data. Section 6.1 and Section 6.2 discuss these results further.

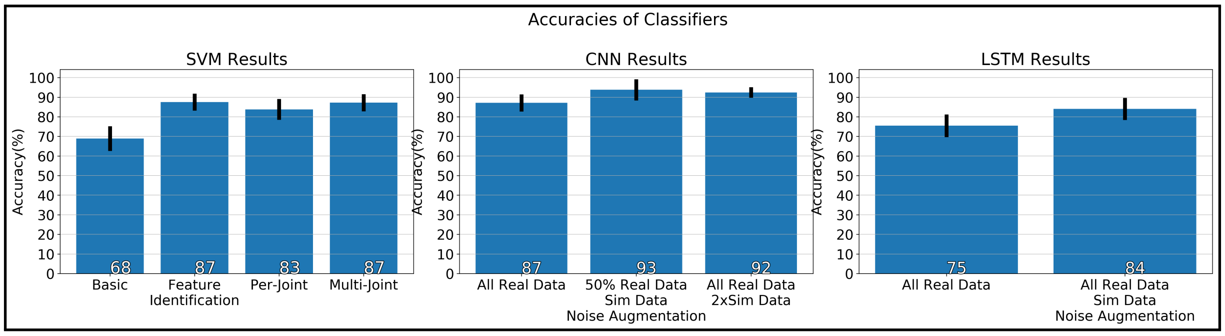

Figure 6.1: Overall results of classifiers

6.1 Detailed SVM Results

This section provides detailed results on the SVM tests performed.

6.1.1 Feature Identification

Figure 6.2 shows the classification performance when using different time series features.

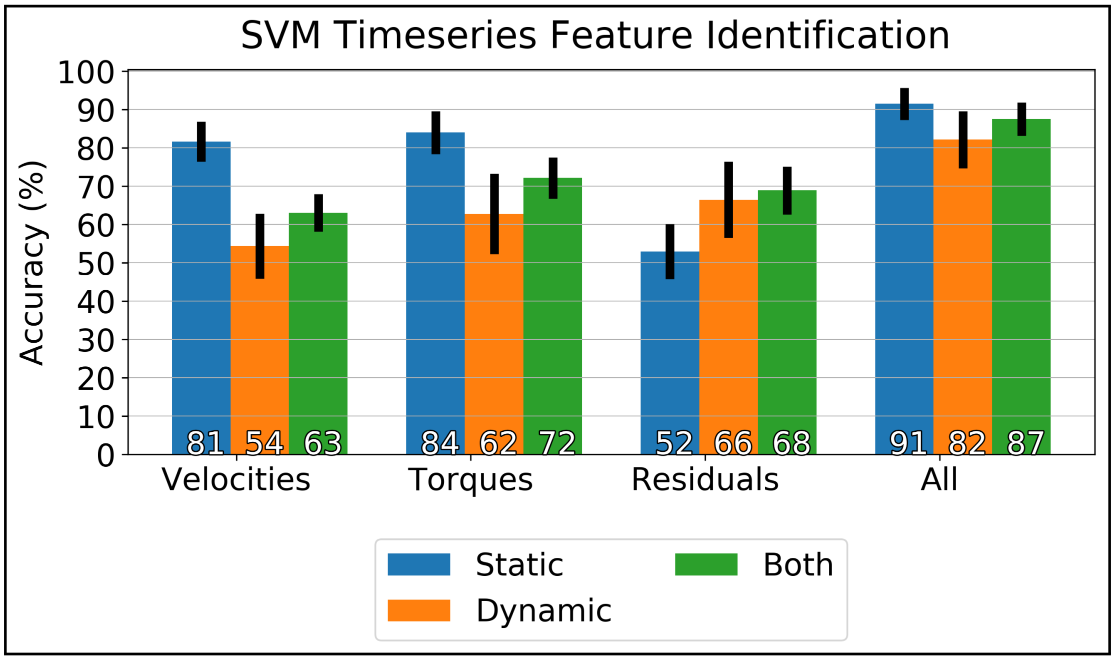

Figure 6.2: Classifier accuracies using various timeseries features

It can be seen that a classifier trained on both sensor- and model-based features outperforms a classifier trained on any single feature. This improvement is seen regardless of dataset used. Appendix B describes additional feature identification tests performed.

6.1.2 Per-Joint Methods

Figure 6.3 shows the results of per-joint classification, recreating the approach proposed in [39]. Note that while the per-joint method of [39] is re-created, [39] classified only interaction/collision or colliding object type, while these results show the use of the four classes presented in Section 3.1.

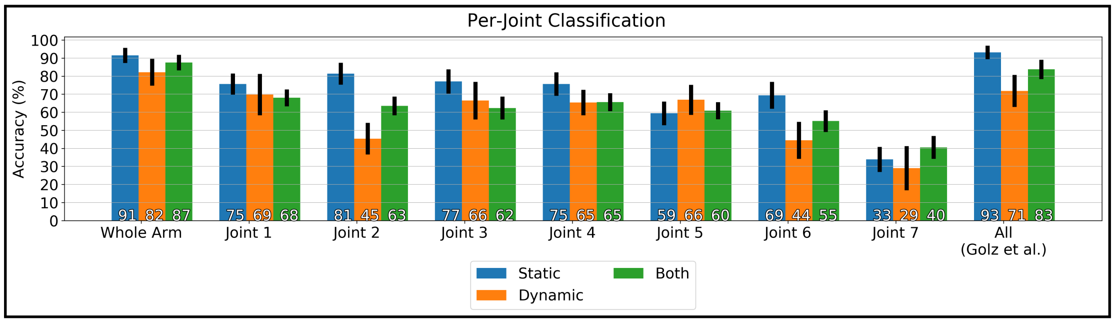

Figure 6.3: Classification accuracy of per-joint ensemble classifier

It can be seen that the classification results from SVMs trained on outputs from single joints are often more than 50% accurate. This is a promising result for building an ensemble classifier. However, the ensemble performs similarly to, or less well than, a classifier trained on the entire arm.

Figure 6.3 shows the results of per-joint classification, using the newly-proposed multi-joint strategy.

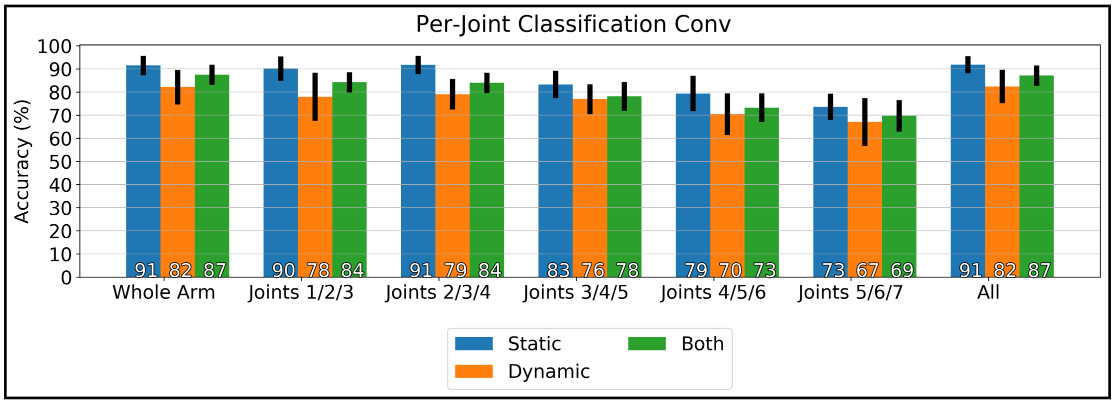

Figure 6.4: Classification accuracy of modified per-joint ensemble classifiers

The accuracy results from classifiers trained on subsets of joints are all above 50%, once again showing promise. When combining these results, the final classification accuracy is approximately equal to the whole-arm classifier. While this achieves higher performance than the previous method, it does not out-perform the simpler whole-arm classification.

It is likely that the individual classifiers trained provide similar predictions for the data, and in this way do not complement each other in a way that leads to greater overall classification. However, Figure 6.4 shows that accuracy near the performance of the whole-arm classifier can be achieved using only subsets of joints. This may prove useful in scenarios where fast performance is needed, and occasional inaccurate results can be tolerated, or the classification problem is simplified.

6.1.3 Discussion

The highest accuracy achieved by a SVM when classifying the combination of static and dynamic data has given an accuracy of approximately 87%. In comparison, [39] reported an accuracy of 97%, however when classifying between collision and interaction only. Popovic et al. [60] reported accuracies of 99% when classifying collisions vs nominal operation and 94% when determining the link in contact. This thesis investigates a classification problem with a larger set of classes than these other works. Thus, some reduction in accuracy is expected.

Factors limiting performance may include the additional number of classes used, the problem domain of classifying between interaction types, or due to aspects of the data used.

Speed Influence

A likely confounding factor is the speed of the arm during dynamic experiments. Motion or acceleration will exacerbate errors in the dynamic model due to friction. Figure 6.5 shows the results of classification with various speeds of arm movement included in the data. Three series are shown: arm motion at a given speed, arm motion at or below a given speed, and arm motion at or below a given speed, including static data.

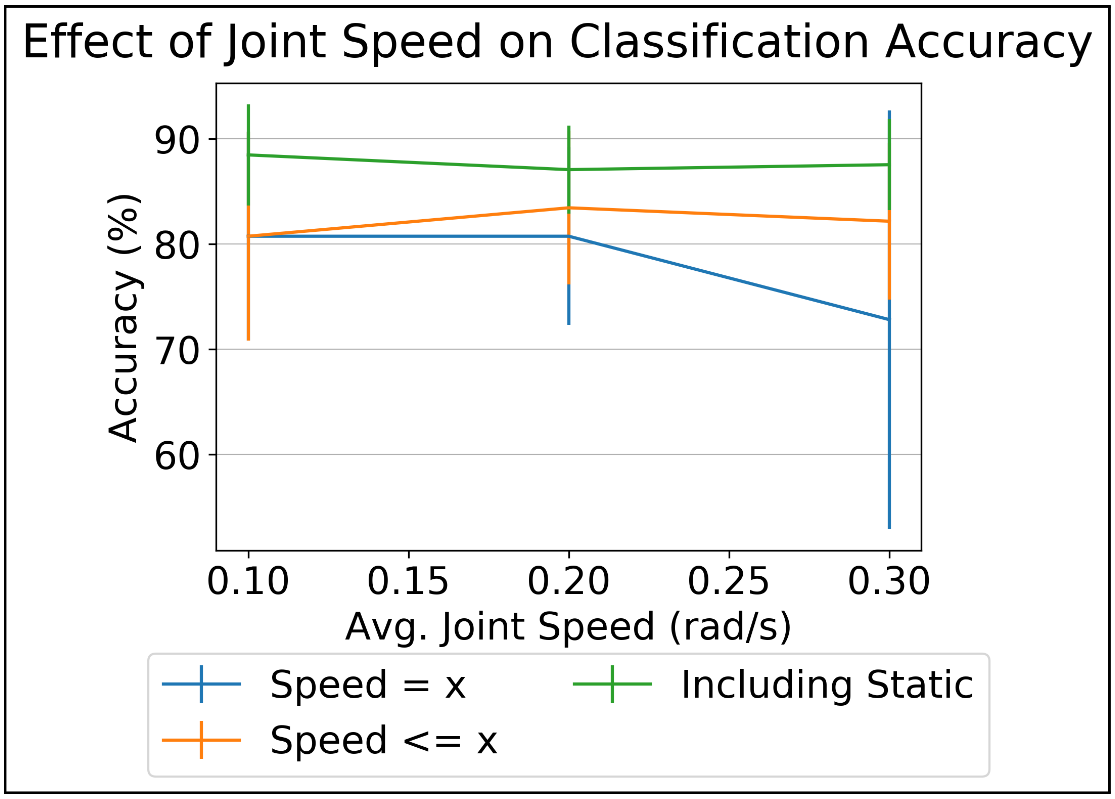

Figure 6.5: Effects of joint speed on SVM accuracy

While it can be seen that using a dataset consisting of only higher-speed data results in some loss in accuracy, this effect is not seen when slower-speed data is introduced.

This is likely due to the features used in classification. While joint velocity and torques will be different under different speeds, the range of their values will likely stay relatively the same across the timespan of a data window. The momentum observer residuals will ideally be equivalent until an interaction occurs, aside from modelling errors being excited at different speeds.

Classes and Context

Several of the classes used in this thesis would fall under a single class of 'intentional interaction' in other works. This use of more classes would be expected to lead to a reduction in accuracy if they are harder to separate by the classifier.

Figure 6.6 shows the confusion matrices generated from tests. These matrices are normalized by row (ie: by each true label), and summed over the 25 SVMs tested in each case. The classes are labelled as (N)one, (A)djust, (B)ump, and (S)hove.

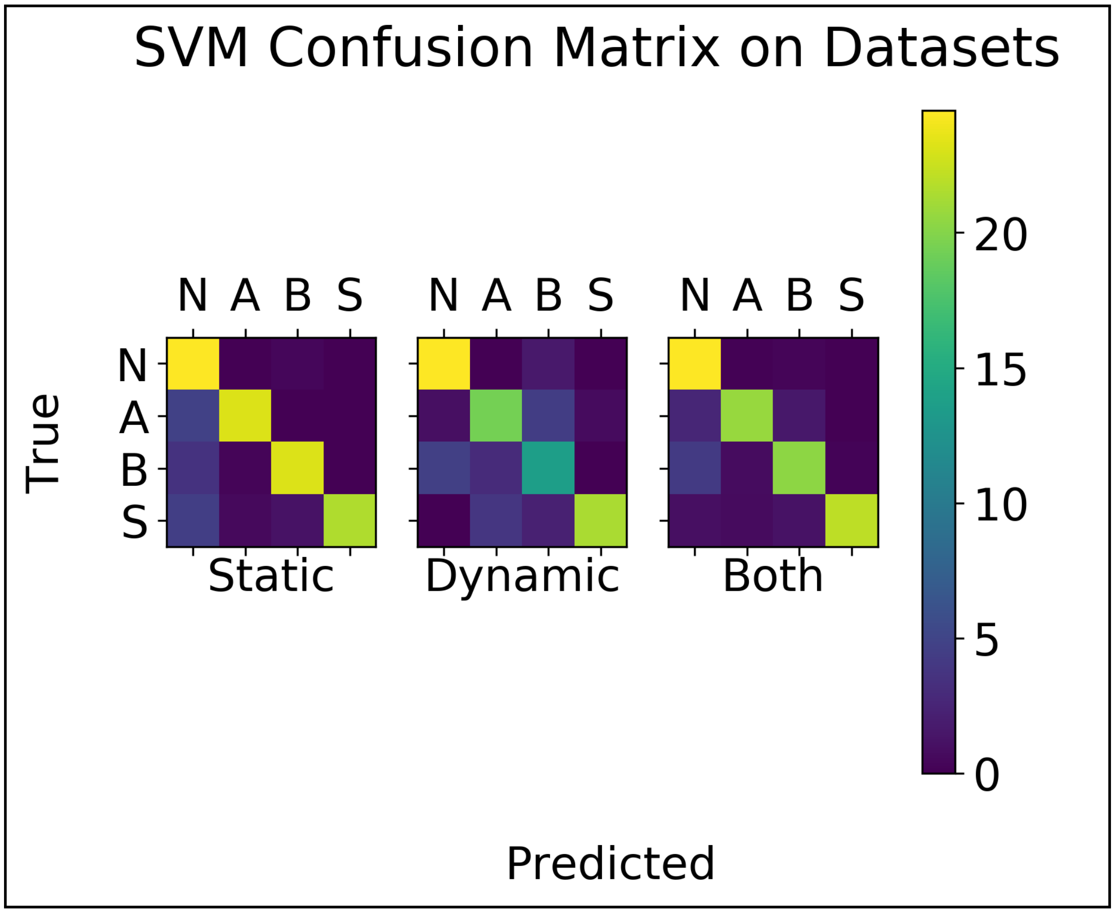

Figure 6.6: Confusion matrix of SVM results

Within the static dataset, there is a reduction in accuracy among the 'Shove' class. Reduced accuracy is seen for the 'Adjust' and 'Bump' class in the dynamic dataset, which correspond to actions with generally lower magnitudes of applied force. Overall, actions appear to be more often mis-classified as actions which are less vigorous, or have a lower magnitude of applied force. Figure 6.7 shows a plot of the magnitude of the residual vector for both correctly and incorrectly classified samples of the dynamic dataset, of the window of data used in classification. It can be seen that the incorrectly-labelled samples appear to have plots with a lower magnitude than most of those which were correctly labeled. It is likely variance in user actions produce these less-vigorous motions, which are classified incorrectly.

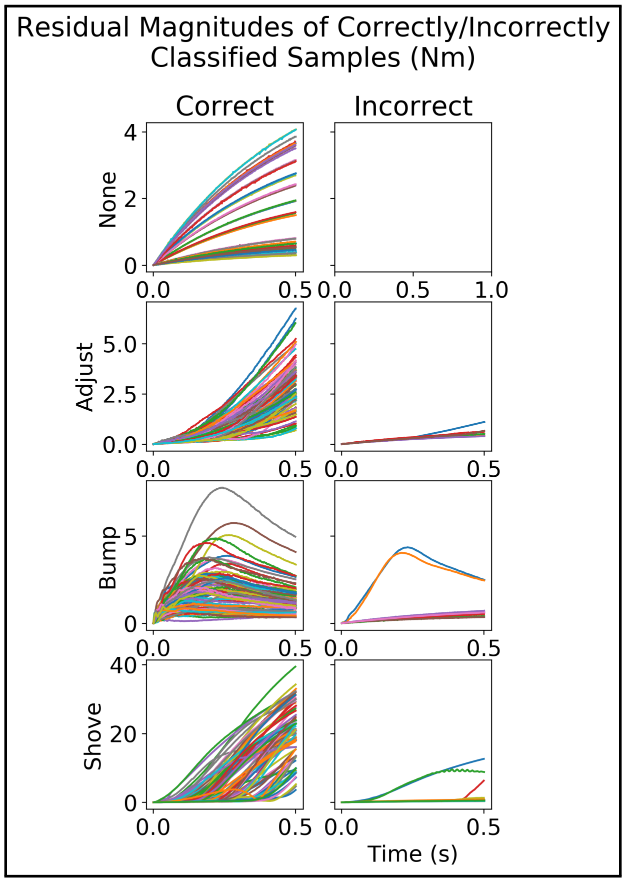

Figure 6.7: Magnitude of residual vector for correctly- and incorrectly-classified samples

Figure 6.8 shows the results of classification with fewer classes. The 'Reaction' case considers the 'Bump' action equivalent to no action; each class may correspond to the need to respond in a different way. The 'Contact' case considered the 'Adjust', 'Shove', and 'Bump' classes to be equivalent; this is a closer approximation of the classes in [39]. Figure 6.8 displays the new distribution of actions into separate classes under each case.

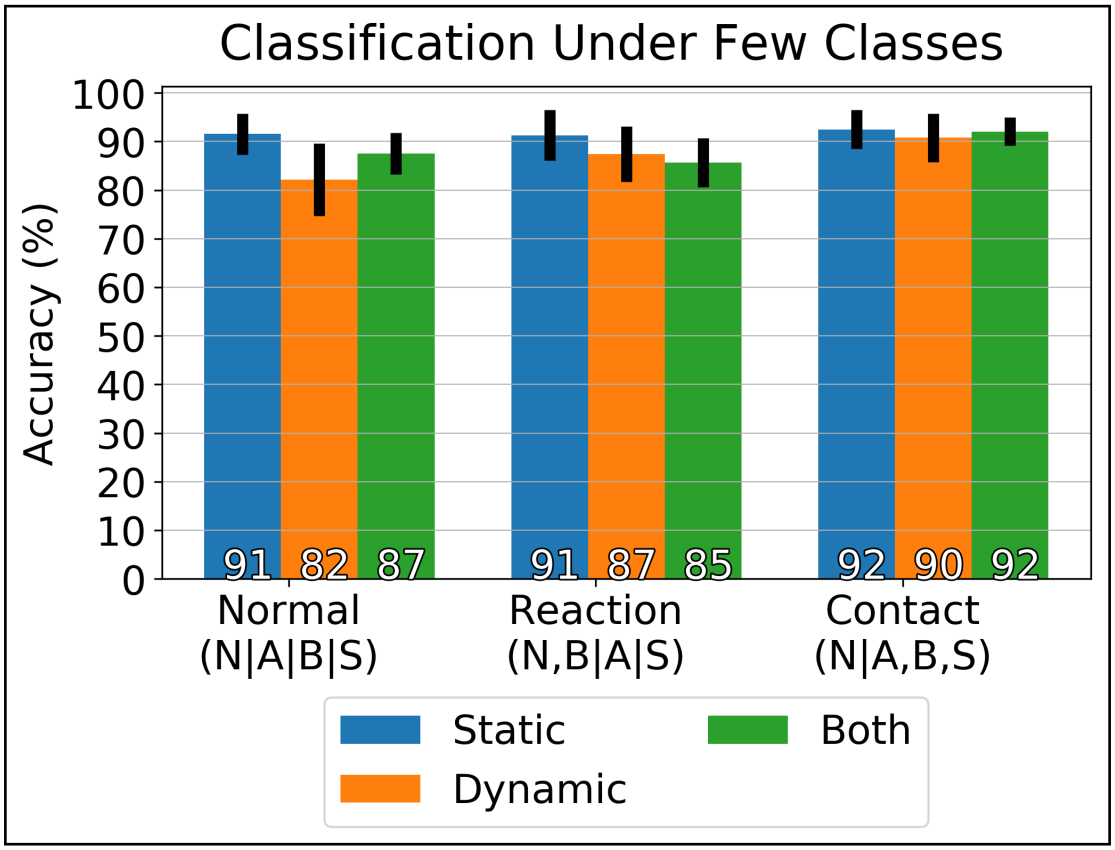

Figure 6.8: Accuracies of SVMs with reduced numbers of classes

Both of these demonstrate some improvement when used with the dynamic dataset, and the 'Contact' case shows improvement with the combined dataset.

Figure 6.9 shows a repetition of these tests where certain low-accuracy actions are removed from each dataset. In the 'Limited' cases, the 'Shove' action is removed from the static dataset and the 'Adjust' action is removed from the dynamic dataset. Note that this refers to samples being excluded from the dataset, and not simply reclassified. the 'Contact' case introduces the reduction in classes discussed above, and the 'Intentional' case considers the 'Bump' action to be equivalent to no action.

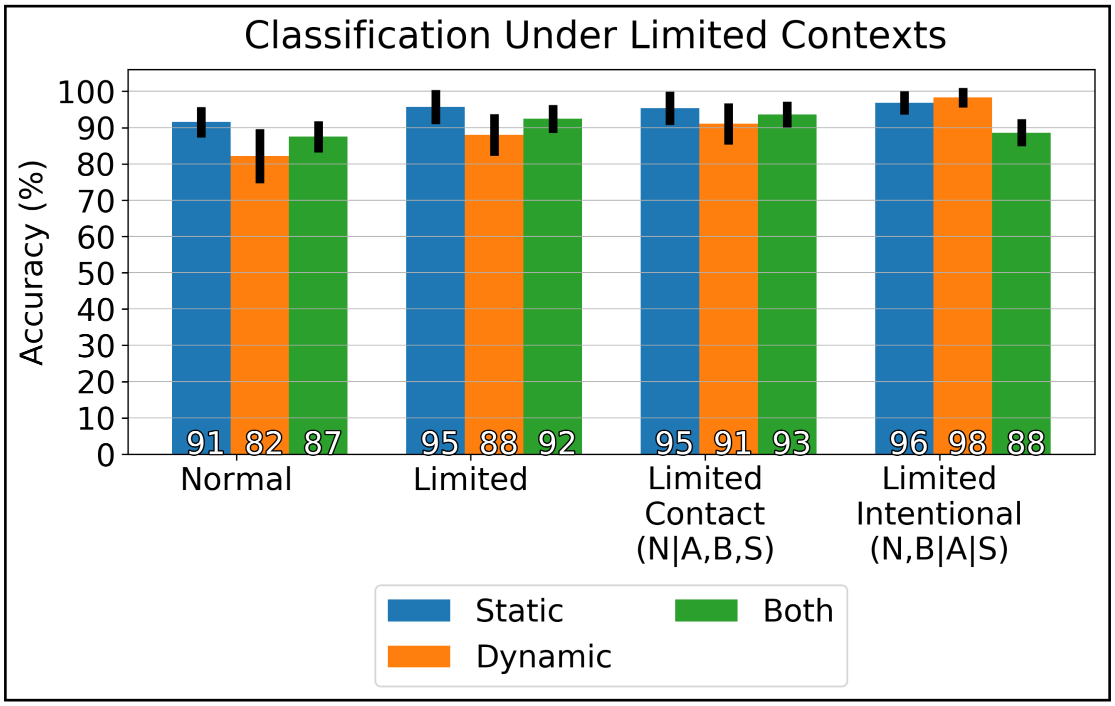

Figure 6.9: Classification accuracy when limiting contexts and classes

Each case shows improvement over the normal classification scheme. The greatest improvement comes from the combination of limiting the number of classes, removing certain actions, and ignoring the 'Bump' action.

It can be seen that limiting either the classes used in the ML problem or the actions performed in different situations leads to better results in accuracy. While these classifier designs may not be as applicable in a pHRI scenario, where each action should be expected at any time, this comparison indicates which cases are most challenging for the classifier.

6.2 Neural Network Tests

This subsection reviews the results of introducing simulated data and data augmentation, as well as reducing the amount of real-world data, for the CNN and LSTM networks, respectively. Significant trends are discussed in each subsection, while Appendix C discuss the trend for each series of data, as well as results of this test preformed on only the static or dynamic datasets.

6.2.1 CNN Results

Figure 6.10 presents the CNN classifier results. Note that the left and right subplots show the performance of the classifier using the same set of data under multiple conditions. The left subplot shows the addition of simulated data without noise augmentation and the right subplot shows the effect of data augmentation with noise. Also note that the results for '2xSim' and '0' of the respective subplots are duplicates, meant to show continuity between the addition of simulated data, and augmentation of the data by noise.

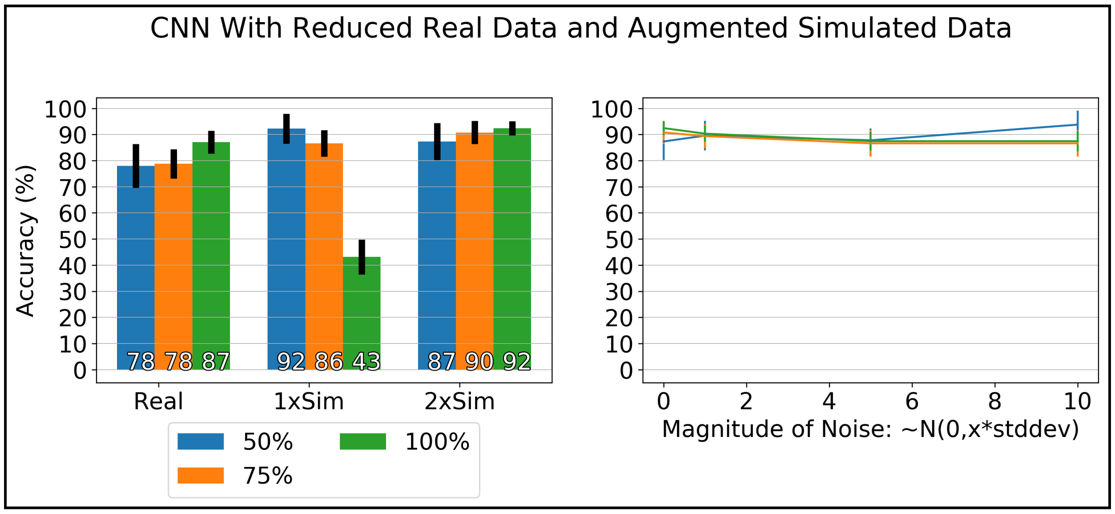

Figure 6.10: Results of reducing real-world data and adding simulated data with varying levels of noise augmentation with CNN

In two cases shown in Figure 6.10, the CNN outperforms the SVM classifier accuracy with an accuracy of 92%. This is seen in the case of using 50% of real-world data with one copy of simulated data, and using 100% of real-world data with two copies of simulated data. The CNN also outperforms the SVM classifier accuracy with an accuracy of 93% when including all real data and two copies of simulated data.

Observing Figure 6.10, there does not appear to be a statistically significant trend when using 75% or 100% of the real-world data. It is unknown why the use of 100% real-world data and 1xSim data produces a one-time drop in performance; this is likely due to random initialization or data selection during the test-train split, but will require additional testing such as k-fold cross-validation to determine.

The 50% real-world data series, however, presents a trend. By adding simulation data, the performance of the classifier increases to 88% at the 1xSim point, nearly the same performance as the network with 100% real-world data and no added simulation data.

The effect of noise augmentation of the simulated dataset appears to lead to slight increases in performance in the 50% and 100% real-world data cases.

6.2.2 LSTM Results

Figure 6.11 presents the LSTM classifier resutls.

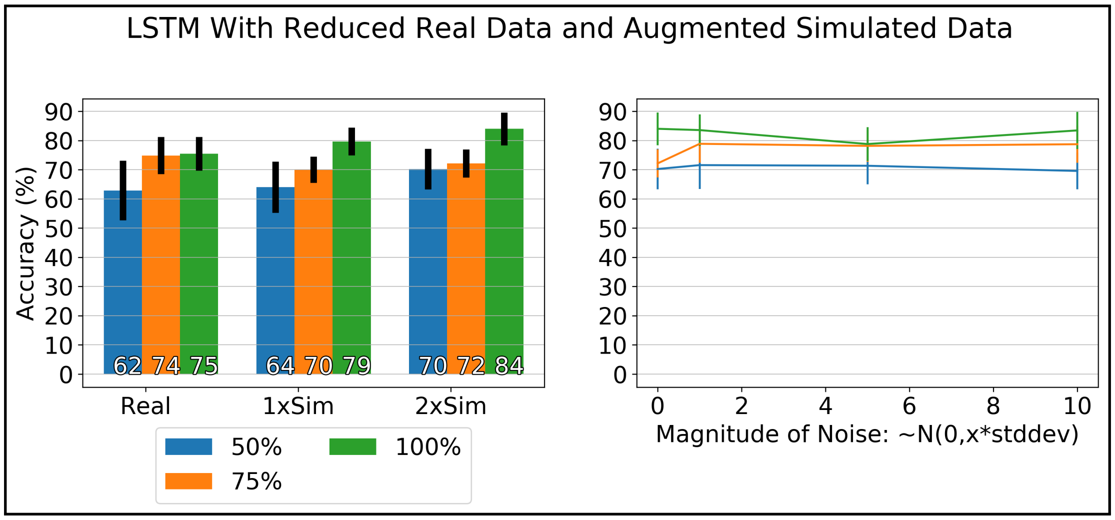

Figure 6.11: Results of reducing real-world data and adding simulated data with varying levels of augmentation with LSTM

Observing Figure 6.11, similar trends are seen when adding simulated data to the 50% real-world dataset. However, in this case, these trends are echoed in the 100% real-world dataset, with an increase being seen with each amount of real-world data. The performance level of the 50% real-world data series does not appear to reach the point of the 100% real-world data with no simulated data added. These results suggest that additional training data will improve results further.

It is not known why the improvements are not seen in the 75% real-world data series, which shows a drop and slight recover in accuracy as simulated data is added. This may simply be due to random initialization of the network or selection/splitting of test/train data.

The effect of augmentation of the simulated dataset does not appear to have a consistent effect.

6.2.3 Discussion

The CNN classifier can outperform the SVM when sufficient training data is available. However, most cases of training the CNN, and all cases of training the LSTM, have not achieved the same performance as the SVM on the combined dataset (87%). This may be due to a few factors.

Dataset Size

The simulated datasets were created to be approximately 3-4 times the size of the corresponding real datasets. With momentum observer augmentation, the doubling of the dataset corresponds to multiples of 6-8. The trends shown often show increased accuracy with additional simulated data, suggesting that even larger datasets may provide improved results.

Randomness in NN Training

NNs are trained with random initialization. This may lead to variations in performance, especially with the potential to encounter local minima during training. While additional time and computational resources may allow for proper k-fold cross-validation and estimation of network variance, current results should be interpreted with this limitation in mind.

Simulated Data Interactions

The current datasets used contain only actions performed on the end-effector link. This contrasts to the real datasets, which consist of actions performed on many links of the robot. The simulated data may have provided increased accuracy when classifying interactions with the end-effector, but biased predictions of samples not involving end-effector interactions.

Chapter 7

Conclusions

This section presents the final conclusions of this thesis, limitations, and proposals for future work.

7.1 Conclusions

In this thesis, the classification of different types of intentional interactions in pHRI was investigated. This was done through the generation of a dataset, and the development of various classifiers. Methods of improving the accuracy of each classifier were investigated.

7.1.1 Support Vector Machines

The first classifiers developed were support vector machines. Feature identification showed that there were accuracy increases seen from combining sensor-based and model-based timeseries features, and that simply using the Range of each timeseries feature as scalar inputs to the classifier gave the best performance. This is surprising given the many past works which used frequency-based data, but allows for a simple implementation of a classification system.

An investigation into using a single SVM for each joint of the robot, and combining their results, showed little improvement over using a single SVM with inputs from the entire arm. A proposed modification, which included SVMs trained on subsets of joints, also showed no significant improvement. This suggests that simple classification systems, with only a single classifier, can perform well.

A final investigation into limiting factors suggests that the major limiting factor in achieving higher classification accuracy lies with identifying certain actions in certain contexts. In particular, more subtle interactions appear to be harder to identify. While ignoring these situations is not appropriate for pHRI, it shows some areas of further development.

7.1.2 Neural Networks

Two neural networks, a Convolutional Neural Network and a Long Short-Term Memory network, were developed. A dataset of simulated data was developed, to investigate the improvement of the networks' performance with augmented datasets.

Both networks showed improvement when adding the simulated data with the CNN achieving higher accuracy than the SVM. In particular, higher accuracy with less real-world data was demonstrated, especially by the CNN. However, this effect was not entirely consistent across networks, or the amount of real-world data used. Drawing further conclusions should be reserved until a more rigorous k-fold cross-validation can be performed.

7.2 Limitations

The primary limitation of this work is the limited amount of real-world data gathered. In particular, all real-world data was gathered by the author. This has the potential to bias the data towards pre-conceived notions of each interaction type, and thus be more easily-separable. Larger studies should be performed before generalizing these results.

A limitation of the neural network development was the lack of k-fold cross-validation, due to limited time and computational resources. While an attempt was made to estimate network performance uncertainty, it cannot fully capture the effects of multiple trained networks.

7.3 Future Work

There are several areas where the investigations and results presented in this thesis can be improved upon.

7.3.1 Varied Participants

An attempt should be made to gather data from a greater number of participants interacting with the robot arm. This will allow for a larger dataset to be generated, and will capture any variations in how different participants interact with a robot. This can help determine the robustness of an approach similar to that this thesis presented, and if it is fit for use in real-world applications.

Such a study can also be implemented in a way that gathers data where a proper task is being performed, as opposed to the arbitrary poses used in this thesis.

7.3.2 Recording of Applied Forces

Future experiments should be performed while recording the forces applied by the human participants (ie: by using a force sensor). This will be beneficial in identifying the starting time of an interaction. It will also allow simulated forces to be generated in a way that better matches the real-world applied forces, as it will allow forces to be generated to match the applied forces, as opposed to applying forces which generate similar calculated values (ie: momentum observer residuals).

7.3.3 Additional Robots and Sensors

The methods and experiments in this thesis should also be tested on other robot platforms. This will help to show robustness, as well as identify features which work well on multiple platforms.

These experiments can make use of any additional sensors installed on the platforms. While these results will be limited to only platforms with each particular set of sensors, they may provide significant benefits. The extension of the per-joint methods to per-joint and per-sensor methods may lead to the development of a modular classification system.

7.3.4 Modifications to Data Classes

Additional classes could also be considered in the future. For example, it may be more useful to classify interactions on a sliding scale, to classify based on required reactions, etc.

7.3.5 Formalized Feature Extraction

Future work should consider performing feature identification using more formal methods. This may include using algorithms such as the Recursive Feature Elimination feature within the sklearn library [1], or additional methods applicable to the machine learning model being considered.

7.3.6 Neural Network Testing

The following improvements should be investigated for training neural networks for this application.

Simulated Data Improvements

The simulated data created for this thesis was limited in the simulated interactions. The interactions were only simulated with the end link of the robot arm, and only with forces in the horizontal plane. Future works should simulate forces interacting with different links, and in more varied directions of applied forces. Both the CNN and LSTM will likely benefit from the generation of larger datasets.

K-Fold Cross-Validation

K-fold cross-validation should be performed. This will allow the actual variance between training different networks to be captured.

Network Architecture Design

A larger-scale grid-search, or other method of optimizing network architecture, should be investigated. This process will also benefit from proper k-fold cross-validation.

Standardizing Split Data

The methods used to reduce data size, and to split data between train/validation/test sets, where entirely random. This leaves the possibility that some tests involved datasets with mismatched amounts of data in each class, or from static/dynamic contexts.

Further work should investigate properly splitting the available data to preserve the portions of classes and motion contexts in each set.

7.3.7 Developing a Real-Time System

Future efforts should be made to develop a classification system operating in real-time on a robot. This can improve the ability to perform real-world experiments, as well as lead to testing of different reaction strategies. It will also motivate the selection of quickly-calculated features, and finding optimized implementations of the ML tools and features discussed in this thesis.

Similarly, the real-time system should also investigate classification with shorter windows of data. A 0.5s window of data was used in this thesis. As this is a preliminary work, and the proposed classification system is not responsible for responding directly to collisions, this is considered acceptable at the moment. However, future works should attempt to increase the response time of any system where safety is a concern.

References

[1]

[2] Sajeeva Abeywardena, Qiaodi Yuan, Antonia Tzemanaki, Efi Psomopoulou, Leonidas Droukas, Chris Melhuish, and Sanja Dogramadzi. Estimation of tool-tissue forces in robot-assisted minimally invasive surgery using neural networks. *Frontiers in Robotics and AI*, 6:56, 2019.

[3] Arash Ajoudani, Andrea Maria Zanchettin, Serena Ivaldi, Alin Albu-Schäffer, Kazuhiro Kosuge, and Oussama Khatib. Progress and prospects of the human–robot collaboration. *Autonomous Robots*, 42(5):957–975, 2018.

[4] Rachid Alami, Alin Albu-Schäffer, Antonio Bicchi, Rainer Bischoff, Raja Chatila, Alessandro De Luca, Agostino De Santis, Georges Giralt, Jérémie Guiochet, Gerd Hirzinger, et al. Safe and dependable physical human-robot interaction in anthropic domains: State of the art and challenges. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, page 1–16. IEEE, 2006.

[5] Alin Albu-Schäffer, Sami Haddadin, Ch Ott, Andreas Stemmer, Thomas Wimböck, and Gerhard Hirzinger. The dlr lightweight robot: design and control concepts for robots in human environments. *Industrial Robot: an international journal*, 34(5):376–385, 2007.

[6] Brenna D Argall and Aude G Billard. A survey of tactile human–robot interactions. *Robotics and autonomous systems*, 58(10):1159–1176, 2010.

[7] S. S. Baishya and B. Bäuml. Robust material classification with a tactile skin using deep learning. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8–15, 2016.

[8] Forrest Bao. pyeeg. https://github.com/forrestbao/pyeeg, 2018.

[9] Forrest Sheng Bao, Xin Liu, and Christina Zhang. Pyeeg: an open source python module for eeg/meg feature extraction. *Computational intelligence and neuroscience*, 2011, 2011.

[10] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[11] Anastasia Bolotnikova, Sébastien Courtois, and Abderrahmane Kheddar. Contact observer for humanoid robot pepper based on tracking joint position discrepancies. In *2018 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, page 29–34. IEEE, 2018.

[12] Nolwenn Briquet-Kerestedjian, Arne Wahrburg, Mathieu Grossard, Maria Makarov, and Pedro Rodriguez-Ayerbe. Using neural networks for classifying human-robot contact situations. In *2019 18th European Control Conference (ECC)*, page 3279–3285. IEEE, 2019.

[13] Gabriele Buondonno and Alessandro De Luca. Combining real and virtual sensors for measuring interaction forces and moments acting on a robot. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, page 794–800. IEEE, 2016.

[14] Christopher JC Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

[15] Justin Carpentier, Guilhem Saurel, Gabriele Buondonno, Joseph Mirabel, Florent Lamiraux, Olivier Stasse, and Nicolas Mansard. The pinocchio c++ library – a fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives. In *IEEE International Symposium on System Integrations (SII)*, 2019.

[16] Justin Carpentier, Florian Valenza, Nicolas Mansard, et al. Pinocchio: fast forward and inverse dynamics for poly-articulated systems. https://stack-of-tasks.github.io/pinocchio, 2015–2019.

[17] Mingzhe Chen, Ursula Challita, Walid Saad, Changchuan Yin, and Mérouane Debbah. Machine learning for wireless networks with artificial intelligence: A tutorial on neural networks. *arXiv preprint arXiv:1710.02913*, 9, 2017.

[18] Chang-Nho Cho, Joon-Hong Kim, Young-Loul Kim, Jae-Bok Song, and Jin-Ho Kyung. Collision detection algorithm to distinguish between intended contact and unexpected collision. *Advanced Robotics*, 26(16):1825–1840, 2012.

[19] Chang-Nho Cho, Joon-Hong Kim, Sang-Duck Lee, and Jae-Bok Song. Collision detection and reaction on 7 dof service robot arm using residual observer. *Journal of mechanical science and technology*, 26(4):1197–1203, 2012.

[20] François Chollet et al. Keras. https://keras.io, 2015.

[21] Andrea Cirillo, Fanny Ficuciello, Ciro Natale, Salvatore Pirozzi, and Luigi Villani. A conformable force/tactile skin for physical human–robot interaction. *IEEE Robotics and Automation Letters*, 1(1):41–48, 2015.

[22] David Coleman, Ioan Sucan, Sachin Chitta, and Nikolaus Correll. Reducing the barrier to entry of complex robotic software: a moveit! case study. *arXiv preprint arXiv:1404.3785*, 2014.

[23] The SciPy community. numpy.var, 2006–. [Online; accessed Sept 02, 2020].

[24] Sven Cremer, Sumit Kumar Das, Indika B Wijayasinghe, Dan O Popa, and Frank L Lewis. Model-free online neuroadaptive controller with intent estimation for physical human–robot interaction. *IEEE Transactions on Robotics*, 2019.

[25] Alessandro De Luca, Alin Albu-Schaffer, Sami Haddadin, and Gerd Hirzinger. Collision detection and safe reaction with the dlr-iii lightweight manipulator arm. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, page 1623–1630. IEEE, 2006.

[26] Alessandro De Luca and Fabrizio Flacco. Integrated control for phri: Collision avoidance, detection, reaction and collaboration. In *2012 4th IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, page 288–295. IEEE, 2012.

[27] Alessandro De Luca and Raffaella Mattone. Actuator failure detection and isolation using generalized momenta. In *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, volume 1, page 634–639. IEEE, 2003.

[28] Alessandro De Luca and Raffaella Mattone. Sensorless robot collision detection and hybrid force/motion control. In *Proceedings of the 2005 IEEE international conference on robotics and automation*, page 999–1004. IEEE, 2005.

[29] Agostino De Santis, Bruno Siciliano, Alessandro De Luca, and Antonio Bicchi. An atlas of physical human–robot interaction. *Mechanism and Machine Theory*, 43(3):253–270, 2008.

[30] Andrea Del Prete, Nicolas Mansard, Oscar E Ramos, Olivier Stasse, and Francesco Nori. Implementing torque control with high-ratio gear boxes and without joint-torque sensors. *International Journal of Humanoid Robotics*, 13(01):1550044, 2016.

[31] Enrico Di Lello, Markus Klotzbücher, Tinne De Laet, and Herman Bruyninckx. Bayesian time-series models for continuous fault detection and recognition in industrial robotic tasks. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, page 5827–5833. IEEE, 2013.

[32] Fabrizio Flacco, Torsten Kröger, Alessandro De Luca, and Oussama Khatib. A depth space approach to human-robot collision avoidance. In *2012 IEEE International Conference on Robotics and Automation*, page 338–345. IEEE, 2012.

[33] Franka Emika. Panda powertool. https://www.franka.de/technology. Accessed: 2020-04-28.

[34] Franka Emika. Robot and interface specifications. https://frankaemika.github.io/docs/control_parameters.html. Accessed: 2020-08-04.

[35] S. Funabashi, S. Morikuni, A. Geier, A. Schmitz, S. Ogasa, T. P. Torno, S. Somlor, and S. Sugano. Object recognition through active sensing using a multi-fingered robot hand with 3d tactile sensors. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2589–2595, 2018.

[36] Claudio Gaz, Marco Cognetti, Alexander Oliva, Paolo Robuffo Giordano, and Alessandro De Luca. Dynamic identification of the franka emika panda robot with retrieval of feasible parameters using penalty-based optimization. *IEEE Robotics and Automation Letters*, 4(4):4147–4154, 2019.

[37] Claudio Gaz, Emanuele Magrini, and Alessandro De Luca. A model-based residual approach for human-robot collaboration during manual polishing operations. *Mechatronics*, 55:234–247, 2018.

[38] Milad Geravand, Fabrizio Flacco, and Alessandro De Luca. Human-robot physical interaction and collaboration using an industrial robot with a closed control architecture. In *2013 IEEE International Conference on Robotics and Automation*, page 4000–4007. IEEE, 2013.

[39] Saskia Golz, Christian Osendorfer, and Sami Haddadin. Using tactile sensation for learning contact knowledge: Discriminate collision from physical interaction. In *2015*

IEEE International Conference on Robotics and Automation (ICRA), page 3788–3794. IEEE, 2015.

[40] Sami Haddadin, Alin Albu-Schaffer, Alessandro De Luca, and Gerd Hirzinger. Collision detection and reaction: A contribution to safe physical human-robot interaction. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3356–3363. IEEE, 2008.

[41] Sami Haddadin, Alin Albu-Schäffer, and Gerd Hirzinger. Requirements for safe robots: Measurements, analysis and new insights. *The International Journal of Robotics Research*, 28(11-12):1507–1527, 2009.

[42] Sami Haddadin, Alessandro De Luca, and Alin Albu-Schäffer. Robot collisions: A survey on detection, isolation, and identification. *IEEE Transactions on Robotics*, 33(6):1292–1312, 2017.

[43] Bo Hjorth. Eeg analysis based on time domain properties. *Electroencephalography and clinical neurophysiology*, 29(3):306–310, 1970.

[44] Jan Hoffmann and Daniel Göhring. Sensor-actuator-comparison as a basis for collision detection for a quadruped robot. In *Robot Soccer World Cup*, page 150–159. Springer, 2004.

[45] Hwan-Wook Je, Jun-Young Baek, and M. Lee. A study of the collision detection of robot manipulator without torque sensor. In *2009 ICCAS-SICE*, pages 4468–4471, 2009.

[46] Alexandros Kouris, Fotios Dimeas, and Nikos Aspragathos. A frequency domain approach for contact type distinction in human–robot collaboration. *IEEE robotics and automation letters*, 3(2):720–727, 2018.

[47] Tin Lun Lam, Hoi Wut Yip, Huihuan Qian, and Yangsheng Xu. Collision avoidance of industrial robot arms using an invisible sensitive skin. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, page 4542–4543. IEEE, 2012.

[48] Alexander Lenail. Nn-svg. `http://alexlenail.me/NN-SVG/AlexNet.html`. Accessed: 2020-07-23.

[49] Nan Liu, Liangyu Li, Bing Hao, Liusong Yang, Tonghai Hu, Tao Xue, and Shoujun Wang. Modeling and simulation of robot inverse dynamics using lstm-based deep

learning algorithm for smart cities and factories. *IEEE Access*, 7:173989–173998, 2019.

[50] Dylan P Losey, Craig G McDonald, Edoardo Battaglia, and Marcia K O'Malley. A review of intent detection, arbitration, and communication aspects of shared control for physical human–robot interaction. *Applied Mechanics Reviews*, 70(1), 2018.

[51] Shujun Lu, Jae Heon Chung, and Steven A Velinsky. Human-robot collision detection and identification based on wrist and base force/torque sensors. In *Proceedings of the 2005 IEEE international Conference on Robotics and Automation*, page 3796–3801. IEEE, 2005.

[52] Stanislav Mikhel, Dmitry Popov, Shamil Mamedov, and Alexandr Klimchik. Development of typical collision reactions in combination with algorithms for external impacts identification. *IFAC-PapersOnLine*, 52(13):253–258, 2019.

[53] Seung-Hyeon Oh, Yu-Ri Lee, and Hyoung-Nam Kim. A novel eeg feature extraction method using hjorth parameter. *International Journal of Electronics and Electrical Engineering*, 2(2):106–110, 2014.

[54] Christopher Olah. Understanding lstm networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015.

[55] Travis Oliphant. NumPy: A guide to NumPy. USA: Trelgol Publishing, 2006–. [Online; accessed Sept 02, 2020].

[56] PAL Robotics. Talos. http://pal-robotics.com/robots/talos/. Accessed: 2020-04-28.

[57] Jia Pan, Ioan A Şucan, Sachin Chitta, and Dinesh Manocha. Real-time collision detection and distance computation on point cloud sensor data. In *2013 IEEE International Conference on Robotics and Automation*, page 3593–3599. IEEE, 2013.

[58] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[59] María Pérez-Ortiz, Silvia Jiménez-Fernández, Pedro A Gutiérrez, Enrique Alexandre, César Hervás-Martínez, and Sancho Salcedo-Sanz. A review of classification problems and algorithms in renewable energy applications. *Energies*, 9(8):607, 2016.

[60] Dmitry Popov, Alexandr Klimchik, and Nikolaos Mavridis. Collision detection, localization & classification for industrial robots with joint torque sensors. In *2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, page 838–843. IEEE, 2017.

[61] Sandra Robla-Gómez, Victor M Becerra, José Ramón Llata, Esther Gonzalez-Sarabia, Carlos Torre-Ferrero, and Juan Perez-Oria. Working together: A review on safe human-robot collaboration in industrial environments. *IEEE Access*, 5:26754–26773, 2017.

[62] ROS.org. rosbag. `http://wiki.ros.org/rosbag`. Accessed: 2020-03-23.

[63] scikit-learn developers. sklearn.svm.svc. `https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`. Accessed: 2020-08-09.

[64] Abdel-Nasser Sharkawy and Nikos Aspragathos. Human-robot collision detection based on neural networks. *International Journal of Mechanical Engineering and Robotics Research*, 7(2):150–157, 2018.

[65] Bruno Siciliano and Oussama Khatib. *Springer handbook of robotics*. Springer, 2016.

[66] Okkee Sim, Jaesung Oh, Kang Kyu Lee, and Jun-Ho Oh. Collision detection and safe reaction algorithm for non-backdrivable manipulator with single force/torque sensor. *Journal of Intelligent & Robotic Systems*, 91(3-4):403–412, 2018.

[67] Ralf C Staudemeyer and Eric Rothstein Morris. Understanding lstm–a tutorial into long short-term memory recurrent neural networks. *arXiv preprint arXiv:1909.09586*, 2019.

[68] MW Strohmayr, H Wörn, and Gerd Hirzinger. The dlr artificial skin step i: Uniting sensitivity and collision tolerance. In *2013 IEEE International Conference on Robotics and Automation*, page 1012–1018. IEEE, 2013.

[69] Ioan A. Sucan and Sachin Chitta. Moveit. `http://moveit.ros.org`. Accessed: 2020-03-23.

[70] Shinji Takakura, Toshiyuki Murakami, and Kouhei Ohnishi. An approach to collision detection and recovery motion in industrial robot. In *15th Annual Conference of IEEE Industrial Electronics Society*, page 421–426. IEEE, 1989.

[71] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[72] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. scipy.fftpack.fft. Accessed: 2020-09-01.

[73] Jonathan Vorndamme, Moritz Schappler, and Sami Haddadin. Collision detection, isolation and identification for humanoids. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4754–4761. IEEE, 2017.

[74] Waterloo Robohub. Panda powertool. https://uwaterloo.ca/robohub/people-profiles/panda-powertool. Accessed: 2020-08-09.

[75] Hongmin Wu, Shuangqi Luo, Hongbin Lin, Shuangda Duan, Yisheng Guan, and Juan Rojas. Recovering from external disturbances in online manipulation through state-dependent revertive recovery policies. In *2018 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, page 166–173. IEEE, 2018.

[76] Yoji Yamada, Yasuhiro Hirasawa, Shengyang Huang, Yoji Umetani, and Kazutsugu Suita. Human-robot contact in the safeguarding space. *IEEE/ASME transactions on mechatronics*, 2(4):230–236, 1997.

[77] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into imaging*, 9(4):611–629, 2018.

Appendices

Appendix A

Windows

A.1 Selecting Window Size

Before performing classification, each sample of data must be reduced to a reasonable timespan including measurements from when the interaction to be classified occurred. This is only a requirement for this offline processing of data, as it will not be required in a classifier running online with live data. However, it is an important aspect to consider, as it controls the quality of data used to train and test the classifiers.

Multiple methods of identifying the time the interaction occurred have been implemented and tested. Each is used to isolate a window of data T seconds long, given an initial estimate of the start of the interaction time, T_e. After any method is applied, the full sample of data is cropped to this window for further classification.

A.1.1 Hard-Coded Window

The easiest method makes use of an estimated time when each interaction occurs. This estimated time corresponds to the time the author was signaled to begin the interaction while gathering the datasets, and is encoded into the recorded data. Using a hard-coded method, a timeframe of T seconds an be selected, and the two endpoints of the interaction window can be set to the times $T/2$ seconds before and after the estimated interaction time, potentially with a hard-coded shift in time. While this is the simplest method, it is not robust to human error in performing the interaction at the correct time.

A.1.2 Programatically-Found Window

The final two strategies used involve identifying where an interaction appears to be occurring. At a high level, these methods take a window of time before the interaction occurs, and calibrate the expected noise in some signal. Then, the time when the signal begins to vary by more than some value is used as the start time.

The process for selecting the window of data is described in Algorithm 4. The process relies on a method of identifying when a signal changed sufficiently to consider the interaction as starting. Two methods of comparing the difference in the signal are described in Table A.1, each providing different times t_a and t_b used in Algorithm 4.

Algorithm 4: Data Window Extraction

```
Input  : Full set of recorded data of current sample
         containing times, joint positions $q(t)$, velocities $v(t)$, and torques $\tau(t)$
         Estimated interaction time $T_e$
         Desired window length $T$
Output: Recorded data of one $T$-second segment of the overall sample
1  forall Sample $s_i \in$ dataset do
2      Define signal $s(t) = \sum_{i=1}^{7} \tau_i(t)^2$;
3      Define start of calibration window $t_{cs} = T_e - 2$;
4      Define end of calibration window $t_{ce} = t_{cs} + T$;
5      Calculate $s(t)$ from $t = t_{cs}$ to $t = t_{ce}$;
6      Calculate standard deviation of signal $\sigma(s(t)|_{tcs}^{t_{ce}})$
7      Begin at end of calibration window $t_b = t_{ce}$;
8      Initialize $\Delta_t = 0$;
9      Set $c$ from empirical testing: $c = 5.5$;
10     while Window not found and $t_ce + \Delta_t < T_{end}$ do
11         Increment $\Delta_t$ by one timestep;
12         Determine $t_a$ and $t_b$ using Difference or Change method;
13         if $|s(t_a) - s(t_b)| > c * \sigma(s(t)|_{tcs}^{t_{ce}})$ then
14             Indicate window found;
15             Exit Loop;
16         end
17     end
18     if Window not found then
19         Set $t_b = T_e$;
20     end
21     Extract window from $t_b$ to $t_b + T$
22 end
```

Table A.1: Methods for calculating change in signal

Method	Description	t_a	t_b
Difference	One fixed time	$t_a = T_e - T$	$t_b = t_{ce} + \Delta_t$
Change	Moving comparison	$t_a = t_ce + \Delta_t$	$t_b = t_a + 0.15s$

When this occurs, t_b is taken as the start time of the interaction. If no such time is found, each method falls back to the estimated start time.

This process is illustrated in Figure A.1. The first plot shows the joint torque measurements of all seven of the Panda's joints. The bottom plot shows a plot of the signal $s(t)$. The vertical black line at 3.0s shows the estimated interaction time. The calibration window is shown highlighted in yellow, from 1.0s to 2.0s. The interaction start time was detected at approximately 3.5s, and the final selection of the 0.5s window is highlighted in pink. The plot was made using the Difference-Based method, using $c = 5.5$

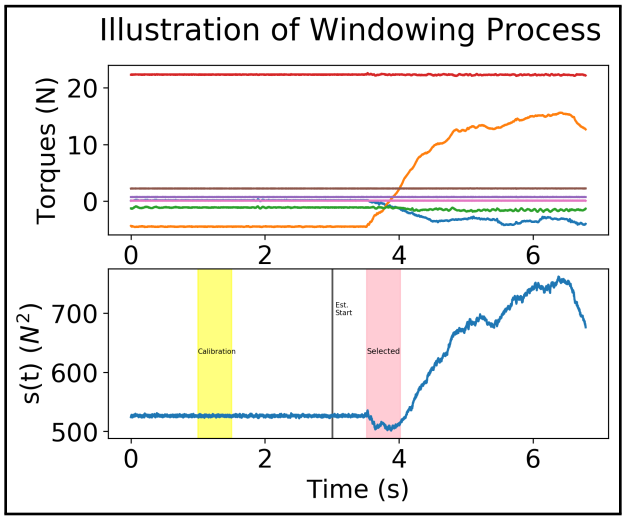

Figure A.1: Illustration of the windowing process

A.1.3 Results of Initial Testing

An initial set of tests indicated the following choices of window-choosing strategies for each dataset. These tests were performed similar to the tests described in Section 4, with joint torque timeseries features, and mean, variance, and range scalar features used across all tests.

The results showed that the Difference strategy performed best for the static real dataset, while hard-coded windows performed best for the real dynamic, and simulated, datasets.

Appendix B

SVM Dataset Investigation

The appendix describes a detailed investigation into the performance of SVMs on the real datasets. This includes feature identification, as well as the effects of filtering input data before classification.

B.1 Description of Classification

With the appropriate window of data selected, classification can be performed. There are 3 major components to this process:

1. Calculation and Selection of Timeseries Features
2. Calculation and Selection of Scalar Features
3. Training and Evaluation of Classifiers

B.1.1 Calculation of Timeseries Features

The first step in this process is to calculate any additional timeseries features from the recorded sensor readings. These include (along with a shorthand label used in some figures):

- Joint acceleration (from finite differences)

- Momentum Observer residuals
- External Forces (estimated from residuals)
- Internal joint power (product of joint velocity and torque)
- External joint power (product of residual and torque)

and additional model-based features. The Pinocchio rigid body dynamics library [16], introduced by Carpentier et al. [15], was used to perform model-based calculations.

B.1.2 Selection of Timeseries Features

The second step in the process is to select which timeseries features will be used in each classification test. For example, one classifier may use the measured torques from every joint, while another may use the measured torque and velocity of a single joint.

B.1.3 Selection of Scalar Features

The third step in the process is the selection of which scalar features to use. A scalar feature is a single (or multiple finite) value calculated from a timeseries of data. For example, the mean is a scalar feature, whether calculated over a set of joint torque readings, or external force values. A list of the scalar features is given below, with many taken from [39]. Unless otherwise noted, these were implemented using the Python numpy [55] or scipy [71] libraries, or through manual implementation with Python

- Mean
- Variance
- Range
- Delta (final - initial value)
- Skew
- Energy
- Hjorth Mobility and Complexity [43] [9] [8], referenced as 'Hjorth'
- Fast Fourier Transform (FFT)

B.1.4 Testing of Classifiers

Classifiers are trained and tested as described in Section 4.3.3. 25 SVMs are trained on the provided data, each using a random selection of 80% of the data, with 20% held out for testing.

B.2 Testing of Timeseries Features

The first tests were conducted to determine the most useful timeseries features for classification. In these tests, the mean, variance, and range scalar features are used.

B.2.1 Sensor-Based Timeseries Features

This test compares the various timeseries features which are recorded directly from sensors on the Panda, or can be simply calculated (acceleration and internal power). The results are shown in Figure B.1.

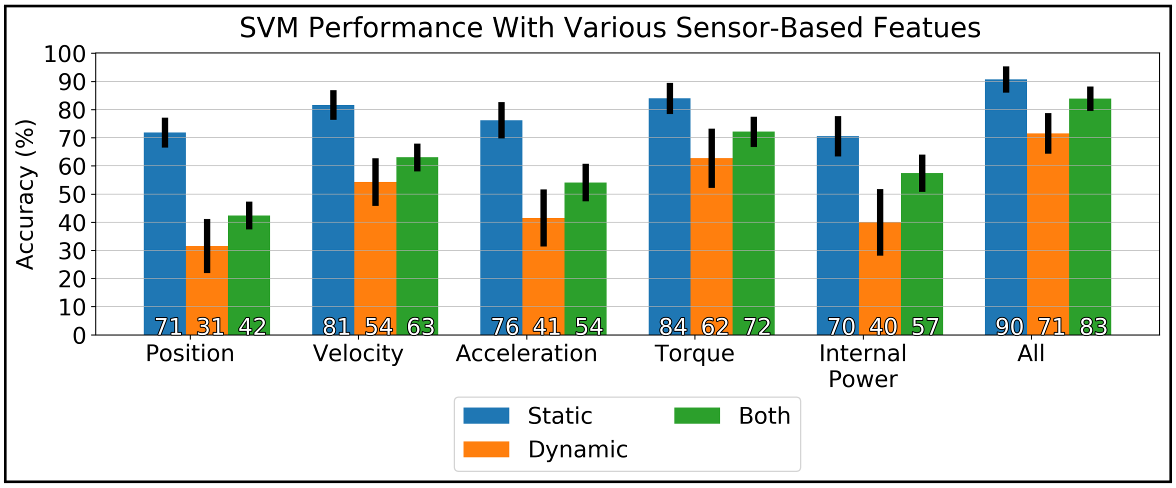

Figure B.1: Comparison of Sensor-Based timeseries features

Figure B.1 shows that the best performance is seen when using either joint velocity or joint torque as timeseries features. There is an increase in accuracy seen when using all features. However, a smaller selection of features will generally be preferred due to computational efficiency.

B.2.2 Model-Based Timeseries Features

This test compares the various timeseries features which must be calculated using a dynamic model of the robot. In this test, only those features which map to joint-space are shown. The results are shown in Figure B.2.

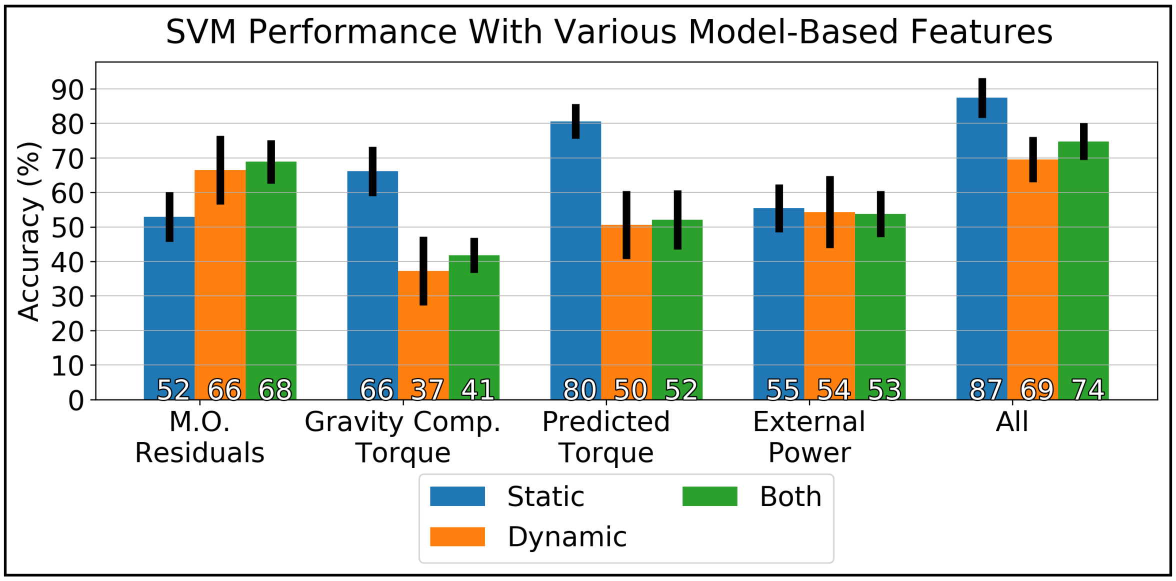

Figure B.2: Comparison of Model-Based timeseries features

Figure B.2 a more varied set of results. Both gravitational force and inverse-dynamics torque perform will on the static dataset. However, the momentum observer residuals perform the best over all datasets. There is a similar performance gain in combining all features.

B.2.3 Workspace-Based Timeseries Features

The final timeseries features to be tested are those which exist in the work-space, or cartesian space. On the Panda, these are the estimated external forces and torques acting on the arm, and are estimated using the momentum observer residuals. Figure B.3 shows the results of testing three versions of these features:

- 3-element vector of external forces

- 3-element vector of external moments
- 2-element vector of the magnitudes of the above

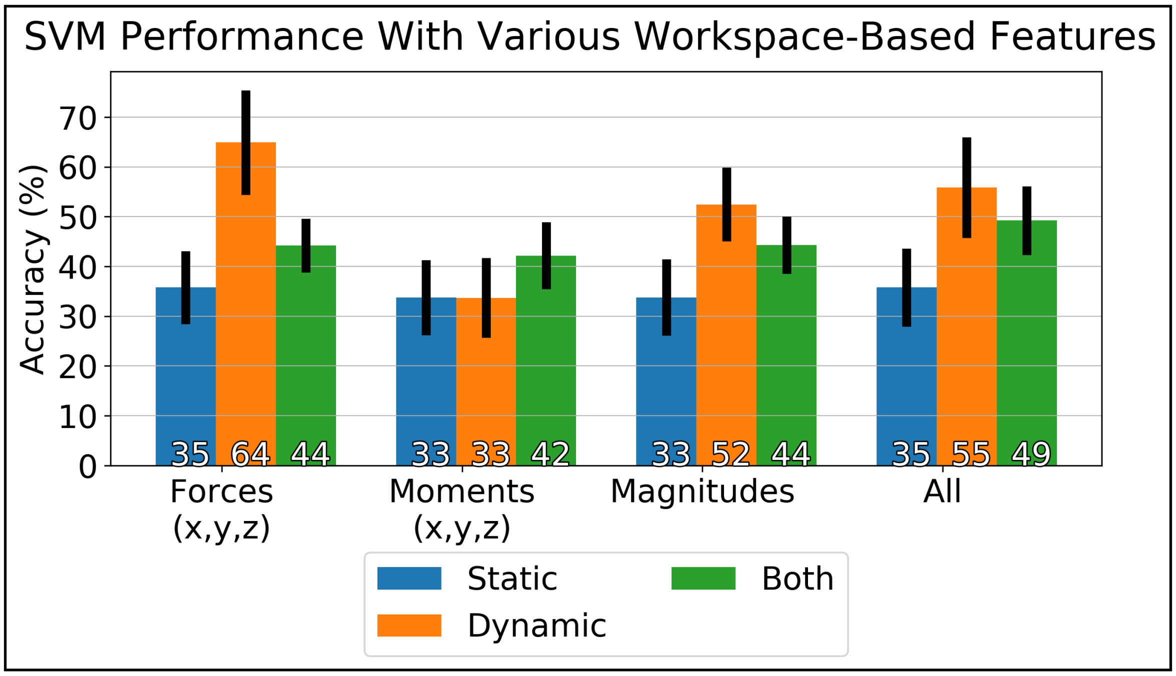

Figure B.3: Comparison of Workspace-Based timeseries features

Figure B.3 shows that very few of these features provide classification results above 50%.

B.2.4 Combinations of Timeseries Features

As explored in Section 4 and presented in Section 6, the combination of joint velocity, torque, and momentum observer residuals was show to perform will over all datasets. Figure B.4 shows a test of combinations of these features.

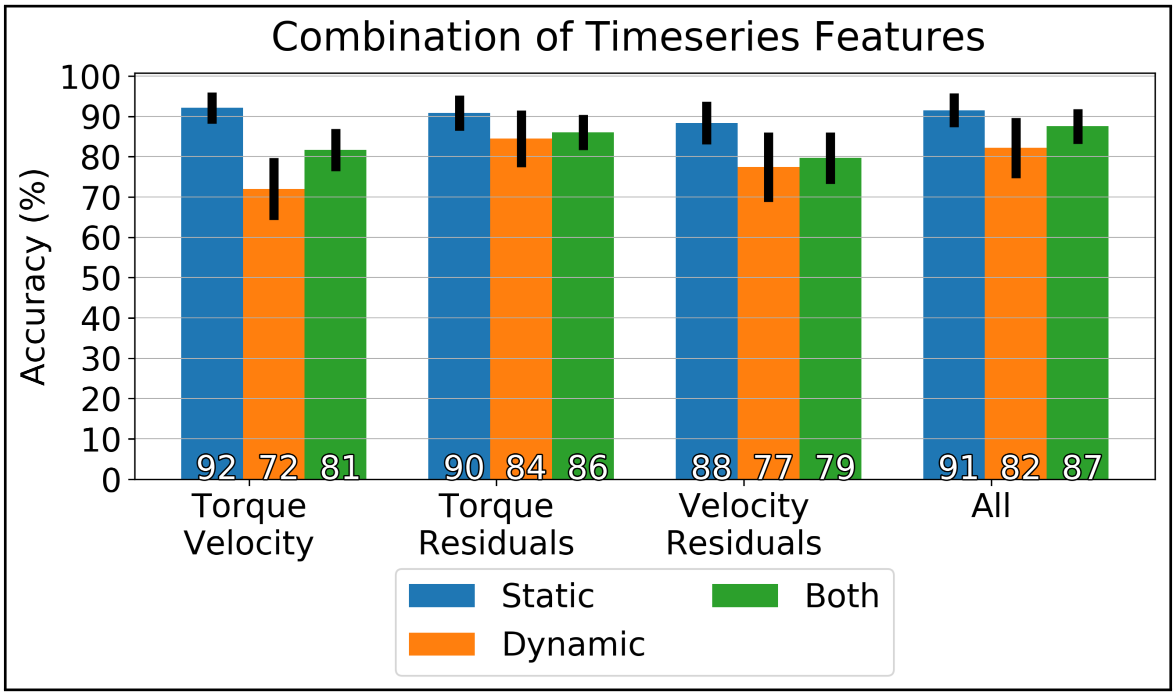

Figure B.4: Comparison of combinations of timeseries features

It can be seen from Figure B.4 that the combination of all three timeseries features does perform the best. The next-best performing combination across all datasets is the combination of forces and momentum observer residuals, which is also a combination of sensor- and model-based features. It should be noted that the combination of these 3 features out-performs all of the previously presented 'All' features tested.

It remains a possibility that the workspace-based features may provide some additional information when combined with joint-space-based features. Figure B.5 shows the addition of linear forces or force/moment magnitudes to the torque/residual and velocity/torque/residual features.

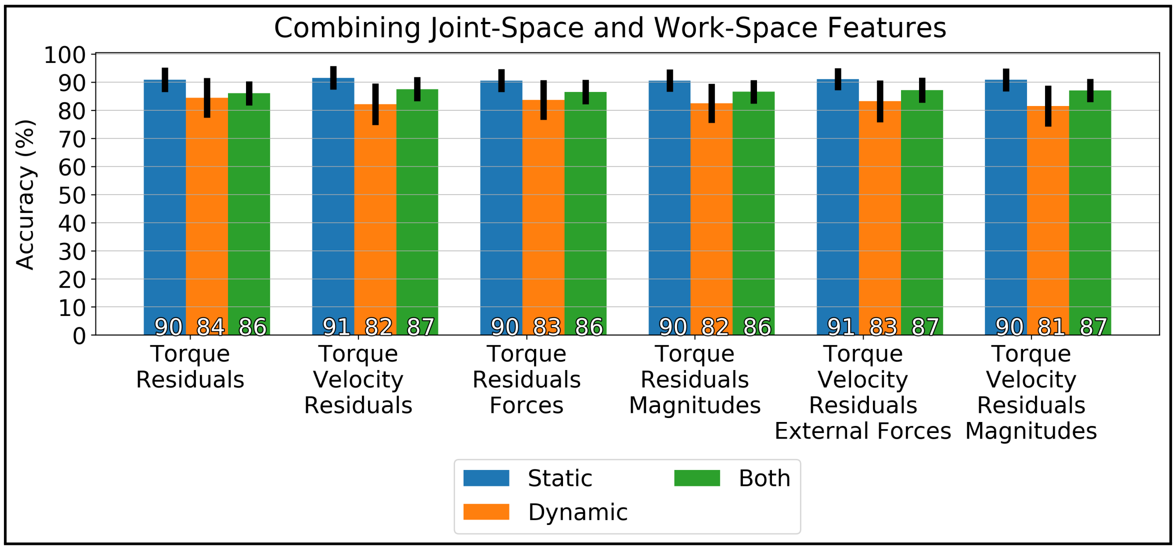

Figure B.5: Addition of workspace-based timeseries features

It can be seen that this addition does not further improve the classification results. This is surprising, given previous works such as [75] and [31] which make use of end-effector wrench measurements. A complicating factor may be that the location of application of a force is needed to accurately reconstruct the force. Performance might be improved in other methods such as those used in [60], or simply isolation of the contact link [42] are used in order to determine the correct jacobian for the calculation.

B.3 Testing of Scalar Features

These tests were conducted to determine the most useful scalar features for classification. These tests were run parallel to the timeseries features tests, and the joint torque timeseries features were used for all tests. Figure B.6 shows the accuracies produced by using each feature.

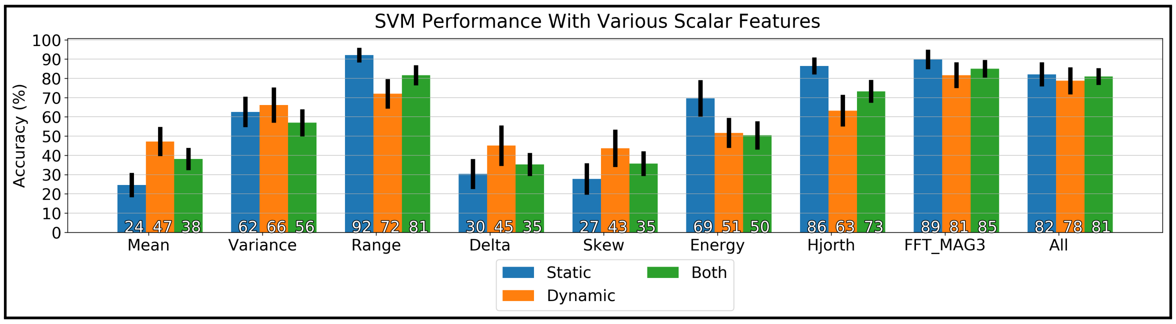

Figure B.6: Comparison of scalar features

The best results come from the use of the range, hjorth mobility and complexity, and Fast Fourier Transform features. Note that the label 'FFT_MAGX' refers to the use of the first X magnitudes returned by the FFT. Figure B.7 shows a test where varying numbers of magnitudes returned by the FFT were used for classification. 3 was chosen as the optimal number across all datasets.

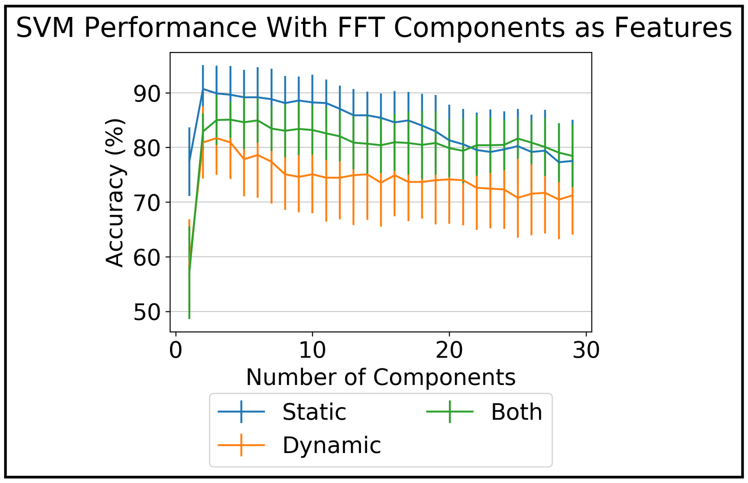

Figure B.7: SVM performance with differing numbers of FFT components

It is notable that both the Hjorth and FFT features include information on the frequency of a signal. It is also interesting that a simple feature such as the range of a the timeseries features provides such relatively high accuracies.

In the interests of future development of a live classifier, the time taken to calculate each feature was also recorded. It should be noted that no development into optimal implementations has been done, and thus, only the relative times taken for each feature are currently of interest. Figure B.8 shows the average time taken in microseconds to calculate each feature across the entire dataset. For simplicity, this was performed only on the static dataset.

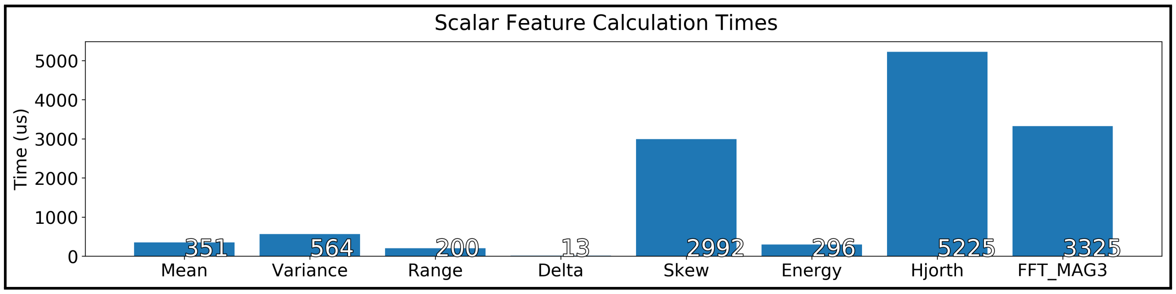

Figure B.8: Time taken to calculate each feature

From Figure B.8, the Hjorth and FFT features take a significant amount of time to compute, while the Range feature does not take a significant amount of time. In future tests, while the Hjorth and FFT features will continue to be investigated, quickly-calculated features and their combinations will be investigated as well.

B.3.1 Investigating Combinations of Scalar Features

Range feature is the highest-performing feature, with the Hjorth and FFT features still performing among the best. Figure B.9 shows a test of combinations of these features. In addition, the combination of Mean, Variance, and Range ('MVR') is tested, in hopes that these quickly-calculated features will complement each other.

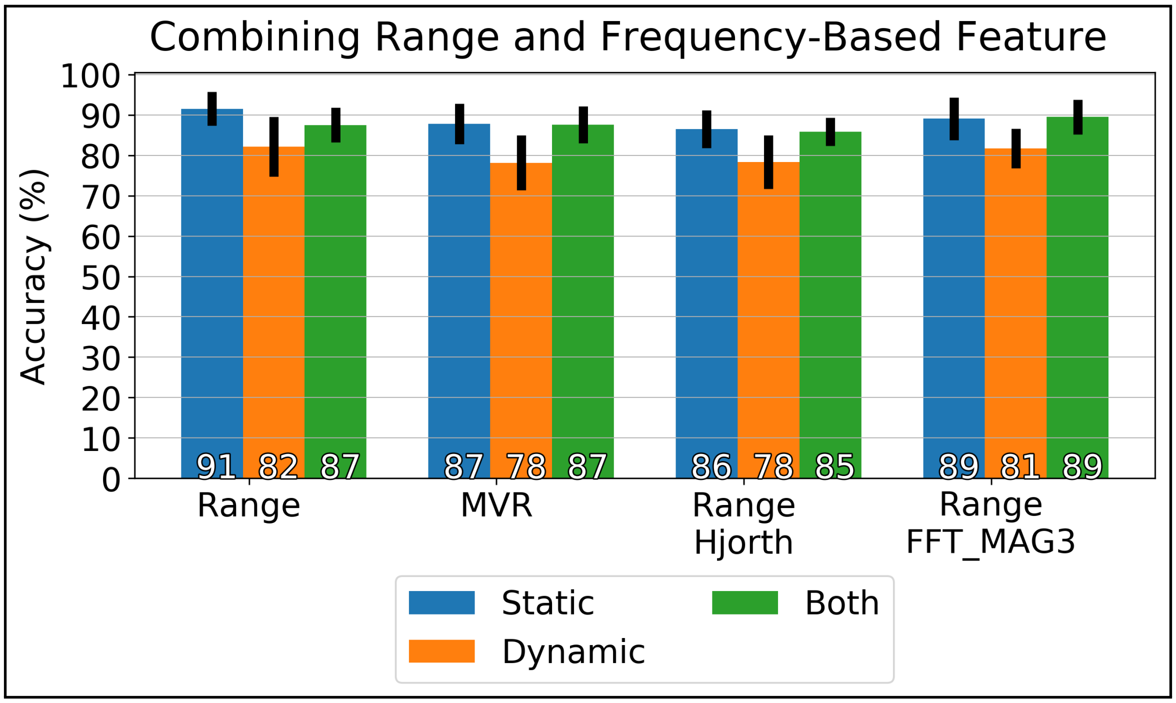

Figure B.9: Testing scalar feature combinations

It can be seen that the use of the range feature alone still provides the optimal results.

B.4 Additional Approaches

This section lists additional approaches taken to investigate ways to improve classification performance.

B.4.1 Filtering of Data

Low-pass filtering can be used to remove noise present in sensor measurements. High-pass filtering has been suggested to remove modelling errors by [40].

A test was created to determine the effect of filtering sensor data. Low- and High-pass filters were implemented using the scipy [71] *signal* library. Filters were applied to joint position, velocity, and torque measurements. Momentum observer values were calculated after this process. A combination of static and dynamic datasets was used.

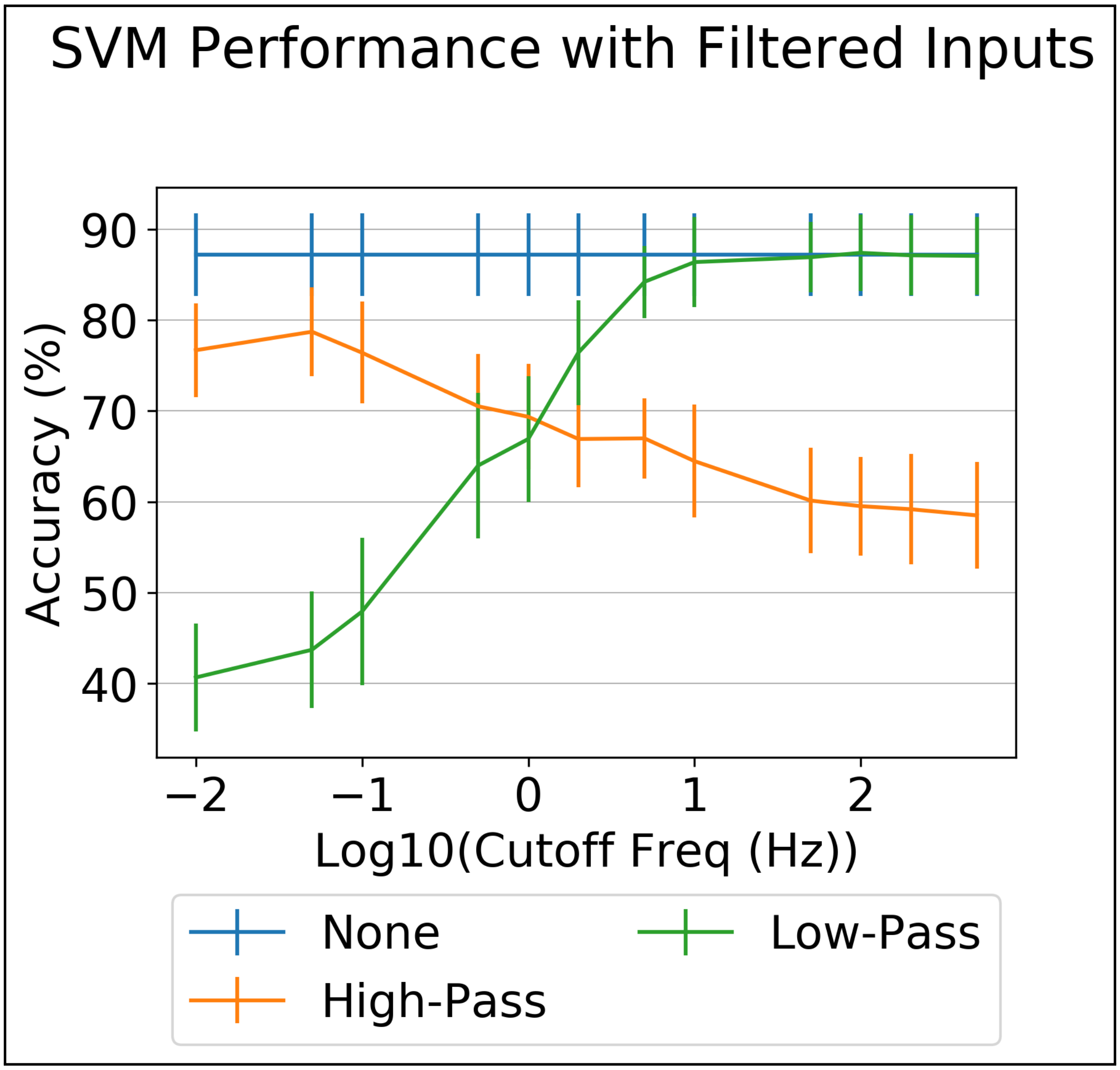

Figure B.10: Testing effect of filtering sensor data

No improvement over the unfiltered case can be seen.

Appendix C

Detailed Neural Network Results

This appendix discusses the results of neural networks in additional depth. This includes an estimation of the variance between multiple trained networks and attempts to estimate this variance, and more detailed results of the tests described in Chapter 5.

C.1 Network Variance

There are many sources of variance in training neural networks. Each time a NN is initialized, it is given an initial set of random weights. During training, different selections/divisions of data into train/test sets may be done, especially when the datasets themselves vary. During training, the training data is also shuffled between epochs. All of these can lead to different results with different networks.

An attempt was made to estimate this variance, and evaluate methods of estimating it. A CNN was trained five times on the real dataset. The results were compared with a single trained network, which used subsets of the training data to estimate the variance in training, as described in Section 5.2.1. These results are shown in Figure C.1.

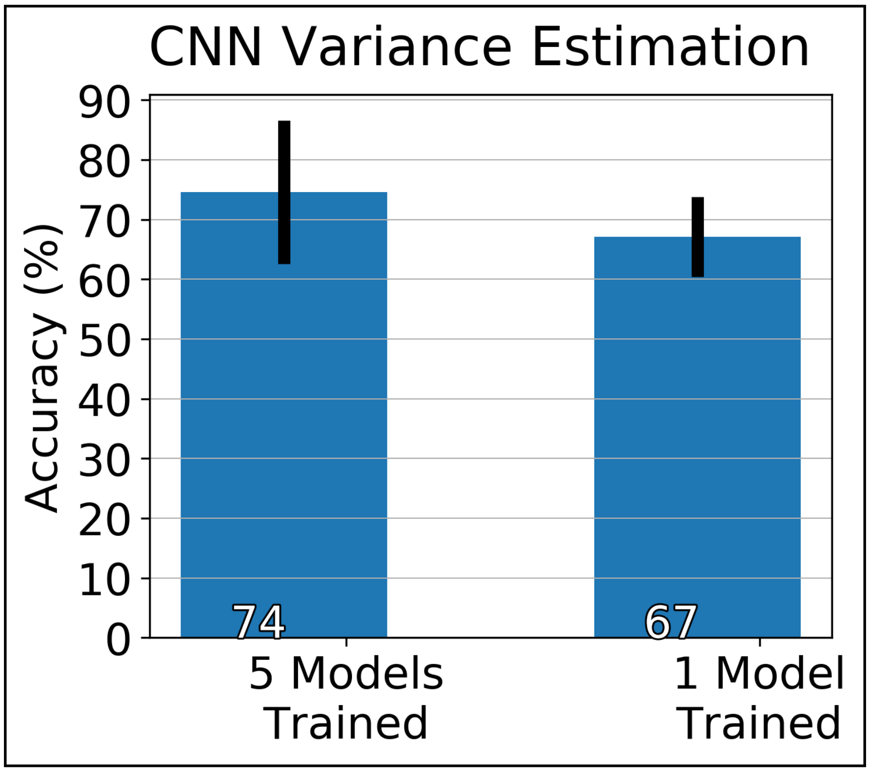

Figure C.1: Estimation of variance of trained networks

It can be seen that there is a significant variance when training multiple neural networks. In addition, the method of estimating this variance underestimates the magnitude of the variance. While current limitations still prevent a proper k-fold cross-validation, interpretations of these results should keep this in mind.

C.2 CNN Augmentation Results

This section reviews the results of augmenting the training of a CNN with simulation data, as originally shown in Figure 6.10. This section presents these results, as well as similar tests performed with only static and dynamic datasets. This comparison may suggest additional trends, or strengthen those already seen through demonstrating the trends repeatedly. Figure C.2 shows the results of tests with both datasets, Figure C.3 the results with the static dataset only, and Figure C.4 ith the dynamic dataset only.

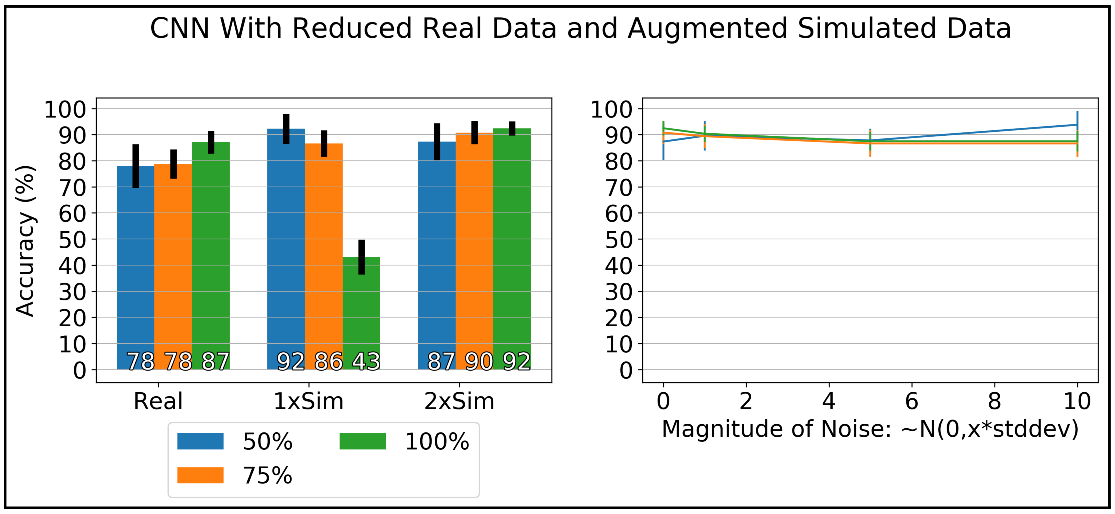

Figure C.2: CNN performance with training augmentation on all datasets

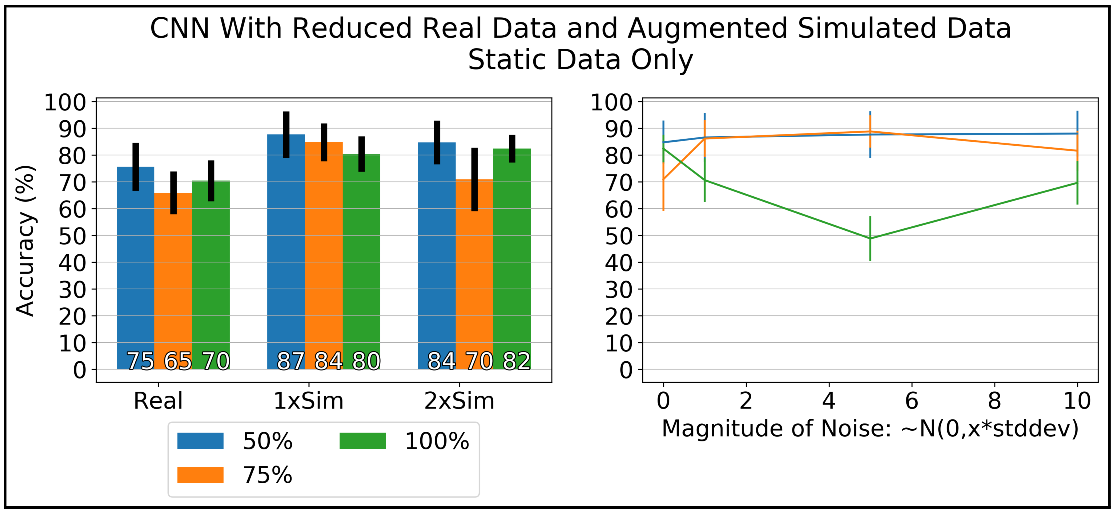

Figure C.3: CNN performance with training augmentation on static dataset only

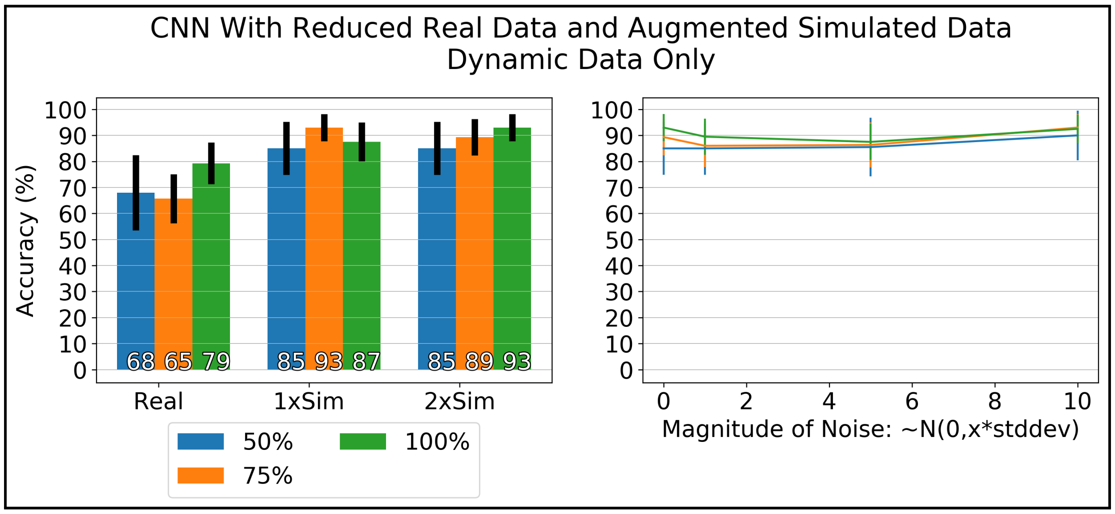

Figure C.4: CNN performance with training augmentation on dynamic dataset only

The trends noted in Section 6.2 was the increase in accuracy under the 50% case. This increase is seen with both the static and dynamic datasets as well, and some further increases are seen with added noise augmentation. There appears to be an overall increase in the performance of the 75% series as well. Finally, the 100% series stays constant and then degrades in the static case, while it increases and levels off in the dynamic case. Table C.1 outlines any trends for each series of data, during both simulated data addition and augmentation.

Table C.1: Detailed trends from CNN test

Dataset	Real Data %	Adding Simulated Data	Augmenting Simulated Data
Both	50%	Increase then decrease	Increase at higher levels
Both	75%	Steady increase	No significant trend
Both	100%	Unexplained drop	No significant trend
Static	50%	Increase then decrease	Slight increase
Static	75%	Increase then decrease	Increase then decrease
Static	100%	Steady increase	Decrease then increase
Dynamic	50%	Increase then hold	Slight increase
Dynamic	75%	Increase then decrease	Decrease then increase
Dynamic	100%	Steady increase	Decrease then increase

C.3 LSTM Augmentation Results

This section reviews the results of augmenting the training of a LSTM with simulation data, as originally shown in Figure 6.11. This section presents these results, as well as similar tests performed with only static and dynamic datasets. This comparison may suggest additional trends, or strengthen those already seen through demonstrating the trends repeatedly. Figure C.5 shows the results of tests with both datasets, Figure C.6 the results with the static dataset only, and Figure C.7 ith the dynamic dataset only.

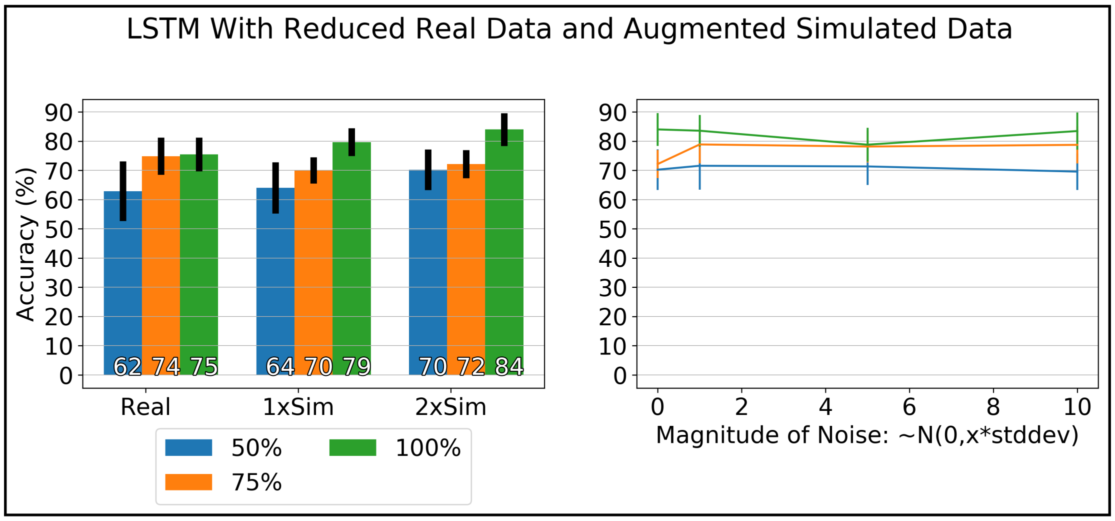

Figure C.5: LSTM performance with training augmentation on all datasets

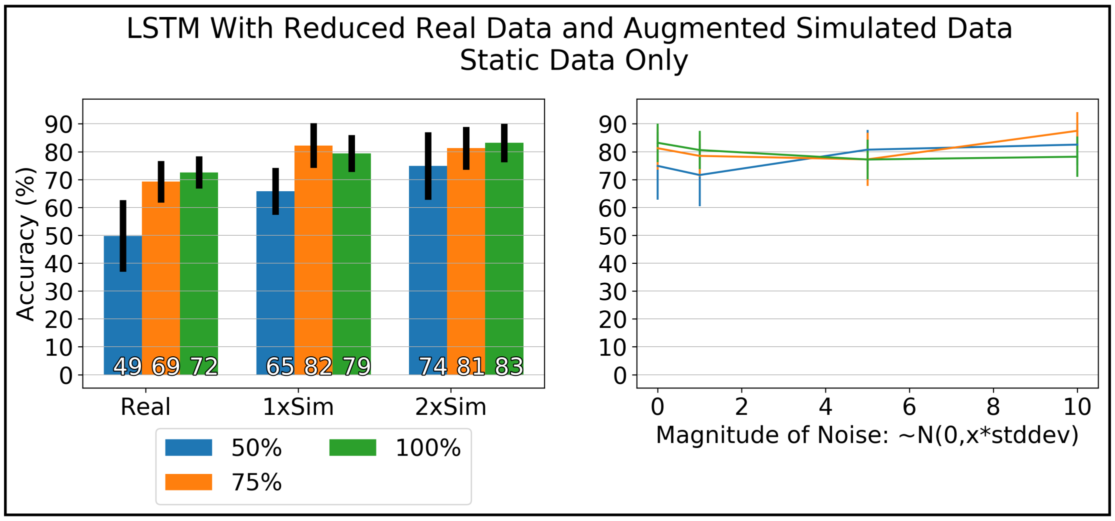

Figure C.6: LSTM performance with training augmentation on static dataset only

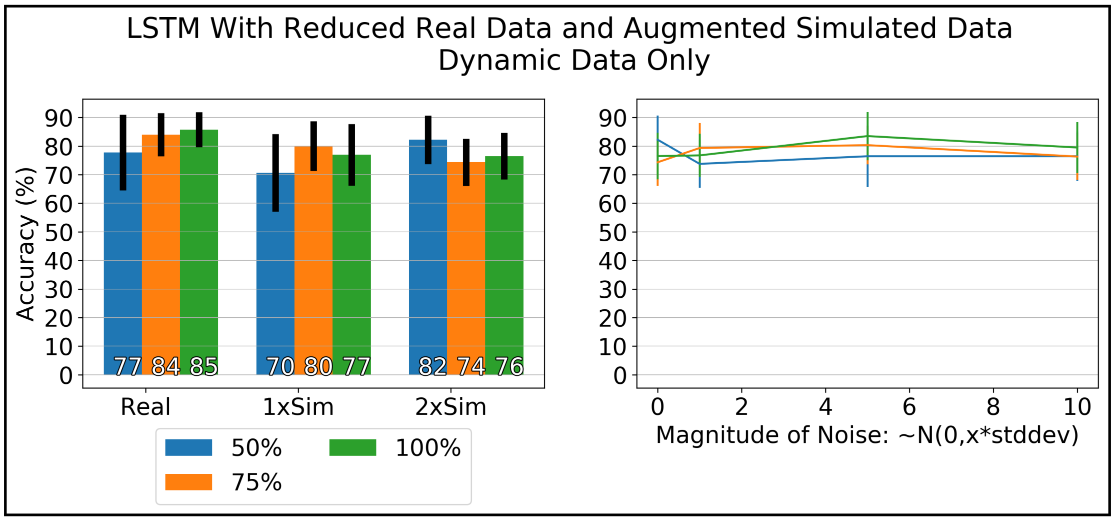

Figure C.7: LSTM performance with training augmentation on dynamic dataset only

The trends noted in Section 6.2 were an increase in the 50% series and 100% series accuracy as simulation data was added. Using only the static dataset, there is an increase seen across all series. The effect of noise augmentation is different for each series, with the 50% and 75% series seeing overall improvements, and the 100% series seeing a slight decrease. Using the dynamic dataset, a drop in accuracy is seen for the 75% and 100% series, while the 50% series experiences a drop and then increase. Augmenting the simulated data with noise appears to generally increase the accuracy. Table C.2 outlines any trends for each series of data, during both simulated data addition and augmentation.

Table C.2: Detailed trends from LSTM test

Dataset	Real Data %	Adding Simulated Data	Augmenting Simulated Data
Both	50%	Slight increase	Little change
Both	75%	Decrease	Some increase
Both	100%	Steady increase	Decrease and increase
Static	50%	Steady increase	Overall increase
Static	75%	Overall increase	Decrease then increase
Static	100%	Steady increase	Decrease
Dynamic	50%	Decrease then increase	Overall decrease
Dynamic	75%	Steady decrease	Increase then decrease
Dynamic	100%	Steady decrease	Increase then decrease