UNIVERSITY OF BEDFORDSHIRE

SCHOOL OF COMPUTER SCIENCE AND TECHNOLOGY

Bachelor's Thesis in Artificial Intelligence and Robotics

Automated Optimisation of Collision-aware Contact Classification in Robotics

Hannah M. Claus

a collaboration between

UNIVERSITY OF BEDFORDSHIRE

SCHOOL OF COMPUTER SCIENCE AND TECHNOLOGY

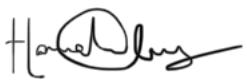
Bachelor's Thesis in Artificial Intelligence and Robotics

Automated Optimisation of Collision-aware Contact Classification in Robotics

Author:	Hannah M. Claus
Supervisor:	Dr Vitaly Schetinin
Advisor:	Dr Daniel Leidner
Submission Date:	13th May 2022

a collaboration between

I confirm that this bachelor's thesis in artificial intelligence and robotics is my own work and I have documented all sources and material used.



Luton, 13th May 2022 Hannah M. Claus

Acknowledgments

First of all, I want to thank my supervisor Dr Vitaly Schetinin at the University of Bedfordshire for the continuous support and motivation during my Bachelor's Thesis, and for always believing in my research.

Specifically, I would also like to thank my advisor Dr.-Ing. Daniel Leidner for his patience, tolerance, and overall wisdom. He and his group of young researchers, FUTURO, made it possible for me to put my thoughts and ideas into viable solutions. They invested a lot of time and patience into helping me implement the methods, create a comprehensible structure and proofread the thesis.

Furthermore, I would like to express my sincere gratitude to the German Aerospace Center (DLR) for giving me the opportunity to work with the humanoid robot Rollin' Justin and use its data for this thesis. Working at the DLR has truly made a dream come true for me, and I am extremely grateful for all the new insights, the gained skills and experiences.

Abstract

Contact classification is of great importance when it comes to robotics and aeronautics. Robotic manipulators should be able to quickly register contact and act accordingly in order to prevent or limit damages through collisions. Hence, the robot needs to be able to discriminate unintended collisions from desired contacts. For this determination, a collision-aware contact classifier is needed. In this thesis, a Fourier-based approach is introduced, which implements a convolutional neural network. Moreover, a framework for collision-aware contact classification is optimised in its labeling and training process. Usually, these processes are time-consuming, hence, automation techniques will be proposed and integrated into the learning pipeline of the humanoid robot Rollin' Justin from the German Aerospace Center (DLR). The aim is to improve the ease of use of such classifiers in future applications and increase the autonomy of humanoid robots.

Contents

List of Figures

List of Tables

1. Introduction

When Karel Capek first introduced the word *robot* from the Czech word for forced labour in 1920 [27], no one would have imagined the impact robots would have on human's lives, society, and science one day. Since then, industrial robots have revolutionised the economy, space robots have explored distant planets and extreme environments, and science fiction has managed to both humanise and dehumanise intelligent robots at the same time, by introducing examples like Terminator and C-3PO. Robots have become part of most people's everyday life, whether they interact with them actively, for instance in care homes or hospitals [14], or passively, i.e. through a vacuuming robot. These robots are used because of their enhanced efficiency and precision, but also because they are in fact not human, hence, they are able work in extreme environments that are unavailable to humans, for example the depths of the oceans, space, or nuclear places. However, robots are also integrated into settings that are usually designed for humans, so that they can collaborate and improve productivity. One example of such a collaborative robot, is a humanoid robot, which resembles the human shape. It can have arms, legs, a head, a torso, and other human attributes. This design makes it possible for it to work efficiently in human environments using human tools. These humanoid robots are programmed to take over simple, repetitive tasks that have previously only been done by humans, such as cleaning a surface or moving objects from one place to another. Especially in environments that are still inaccessible to humans, for example Mars, humanoid robots are trained to assist astronauts.

These robots need to be able to adapt to a certain degree of change and react autonomously in case there is no human around to fix the situation for them. One way for humanoid robots to handle these situations autonomously, is by interpreting occurring interactions correctly. It makes a difference, whether a contact is intended and desired, for example when wiping a surface, or whether a contact is unintended and has the potential to harm the robot, for instance when colliding with a human being or object. Hence, the robot has to discriminate potentially harmful collisions from expected contacts. As a result, successful collision handling methods need to

be developed. To simplify the development of the approaches, collision handling is divided into five phases that describe certain stages of the process: detection, isolation, estimation, classification, and reaction [60]. However, here the focus lies mainly on collision detection. There are many approaches that incorporate collision detection, but there are two that stand out the most. The first method uses tactile sensors that act like human skin, so that collisions are located and identified quickly [58], however, this leaves room for too much noise and uncertainty in the data. Adding to this, the development of artificial skin is very expensive. In contrast, the second method uses a model-based approach that compares joint signals of the robot with predefined thresholds. If the signals are above a certain threshold, they are classified as collisions [46]. This method might be more cost effective and easier to apply, however, it is dependent on the constant thresholds, which do not perform well in changing environments or over different tasks.

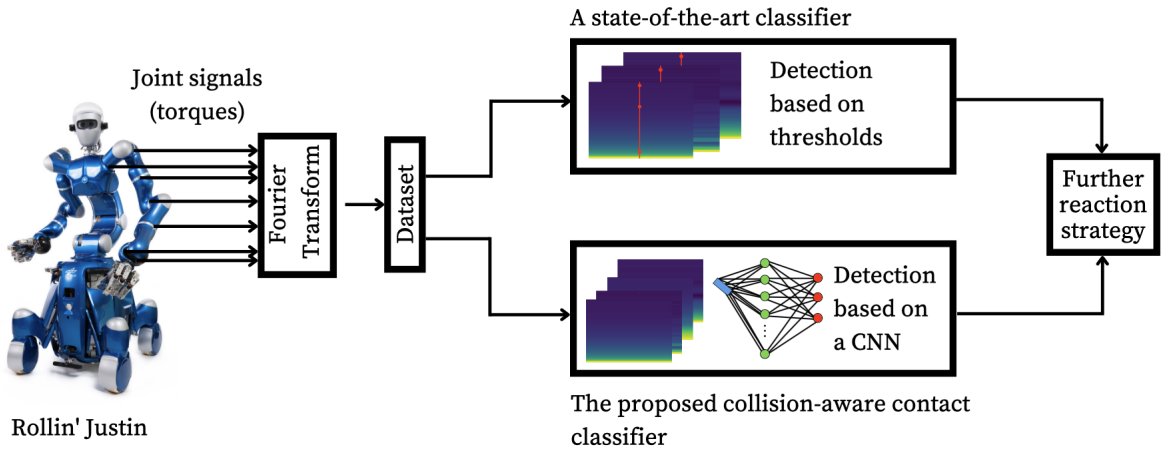


Figure 1.1.: Overview of the classifiers.

As a result, this thesis introduces a novel collision-aware contact classifier, which implements a convolutional neural network that reads torques of a robot, applies the mathematical Fourier Transform to separate the frequency components of the signals, and returns an evaluation of whether the robot is currently in one of three possible contact states, namely No Contact, Contact, or Collision. This way, the recognition accuracy of a state-of-the-art classifier, which only uses thresholds, is enhanced through a learning process to achieve better results. Figure 1.1 shows a visual overview of the two classifiers that will be the main focus of this thesis.

1.1. Motivation

Humans have thought about coexisting with robots for a long time. Even though intelligent systems might be faster than humans when it comes to calculations and and predictions, all these skills need to be trained first. Adding to this, moving from one place to another might be easy for an average human being, but it becomes a complex problem for a humanoid robot. However, the reality is not always the optimal working environment for a robot. Humans learn to adapt to their changing environments, hence, robots have to be able to do the same. This only works by implementing neural networks that support the learning process of the robot. The robot might fail the first time, or the tenth time, but after several training iterations, the robot has learned from past mistakes and uses that knowledge.

For instance, if a robot is supposed to perform the task of wiping a solar panel clean, and before it is finished, the moving arm collides with a surrounding obstacle and changes the trajectory, the robot will just continue its task if nothing was damaged or stop immediately due to an emergency protocol. However, now the calculated trajectory might be wrong because of the stone's impact, which will lead to a failure of the task. This failure mainly happened because the robot was not able to acknowledge the undesired contact and correct the trajectory.

Now, there are many scenarios in which a collision with a robot are possible, whether it is through humans, the environment, or other robots. But in order to use these humanoid robots in normal environments that might even be designed for humans only, they need to be able to learn and adjust so that they are efficient and complete their tasks.

Thus, the first step to making humanoid robots more aware of their environment and handle possible interactions efficiently, is to teach them how to detect contacts and recognise whether they are desired or not.

1.2. Research Question

This thesis focuses mainly on the development of a collision-aware contact classifier as a framework. The purpose of this classifier is to assist a humanoid robot during interaction with the environment and detect collisions as quickly as possible to react with suiting damage control. Nevertheless, there are many ways to implement a collision detection tool with the robot's system.

This work will find solutions for the following research questions:

1. How does the humanoid robot learn to recognise different contact states and classify them correctly?
2. How can the classifier be optimised automatically to achieve optimal results?
3. How can the classifier be implemented as a flexible tool that works on any robot?

This is achieved by following five steps as depicted in Figure 1.2. These steps create a clear structure for the successful development of the needed classifier.

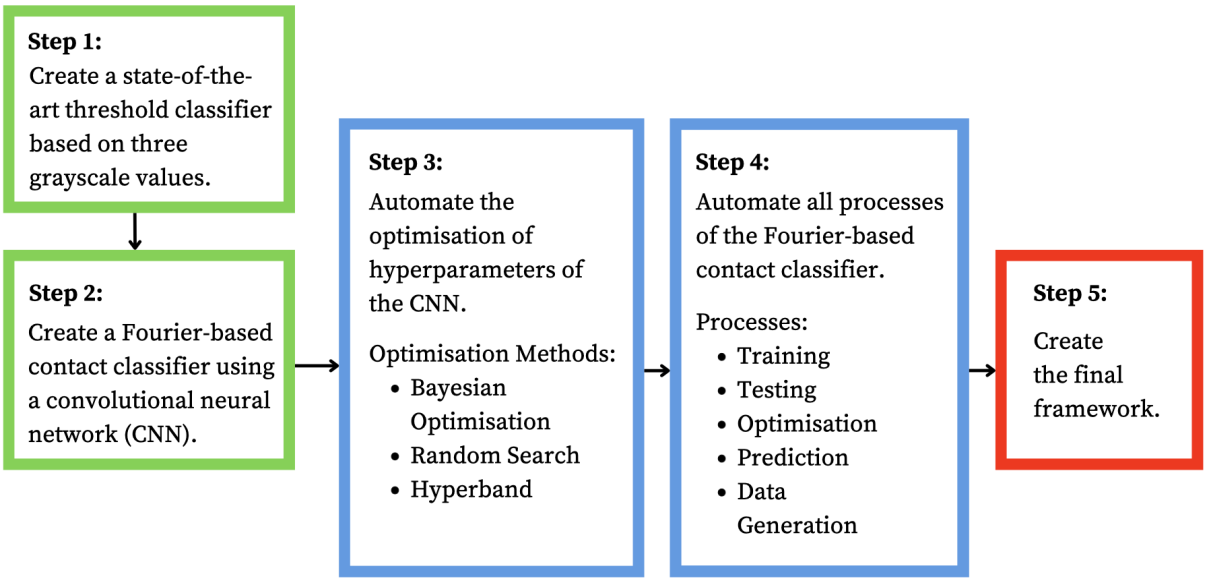


Figure 1.2.: Overview of the main steps for this thesis.

The first step is to create a state-of-the-art collision-aware contact classifier that is based on specific thresholds. Using these thresholds, it should be possible to recognise a contact state. Once the accuracy of the state-of-the-art classifier is established, a new approach is developed using the Fourier Transform as a base. This new classifier utilises a convolutional neural network to enforce the learning process of the robot. In order to achieve the best possible results and exceed the performance of the state-of-the-art classifier, the Fourier-bases classifier needs to be optimised using specific optimisation methods that work automatically. Once the best performance of the novel classifier is identified, the goal is to automate all processes of the classifier.

After the last step is finished, these automated processes are built into a framework that allows the classifier to be used as a tool but also as a pipeline for any future adjustments. Moreover, this pipeline makes the classifier work on any robot, granted that it is provided with the necessary data to retrain the neural network. Evidently, this concludes the steps that are taken during this work.

2. Related Work

Contact classification is essential for humanoid robots. They can only be successful in their tasks if they can interact with their environment. Depending on their specific functions and purposes, some humanoid robots need to be able to distinguish between the three contact states: No Contact, Contact, or Collision. In the following, important technological advances in contact classification are discussed and an overview of the main terminology is provided.

2.1. Theoretical Framework

In order to be able to follow the upcoming experiments fully, the theoretical framework needs to be established. Here, the most important concepts are explained and discussed.

Artificial Neural Network

An artificial neural network (ANNs) has a clear structure with a foundation of an input layer of neurons, a number of hidden layers of neurons and a final layer of output neurons [62]. Furthermore, there are four main elements of an ANN model: The first element is described by the connections between the layers created by synapses, each link is characterised with a weight that will be multiplied by the incoming signal. As a second element, the adder sums up all the products of the inputs and the synapses' weights. Furthermore, there are several activation functions that can be used to limit the outputs. Lastly, the external bias can decrease or increase the input of the activation function [5]. Putting this information into a context, it can be said that the inputs of each layer are the outputs of the preceding layer only. Thus, the activation function is applied to the weighted sum of the layer in order to obtain a new input for the next layer. This kind of network is called a feed-forward ANN because the inputs are fed into one direction only, namely forward.

Convolutional Neural Network

A convolutional neural network (CNN) is a specific kind of Artificial Neural Network (ANN) which is commonly used for image classifications [59]. CNNs regularise fully connected networks in order to prevent overfitting. Usually Multilayer Perceptrons (MLP) connect each neuron of one layer to each neuron of the next layer in a feed-forward ANN. However, CNNs add convolutional layers to the network by using kernels. The kernels stride along the input matrix and a feature map is generated. Following such a convolution, the pooling layer reduces the resolution of the feature map in order to achieve spatial invariance [38]. This is why CNNs are commonly used for image classification tasks because the spatial dependencies will be included as well.

Fourier Transform

The Fourier Transform (FT) is a mathematical transform that turns a function of time into a function of frequency [4]. This way, the same data can be mapped in a different domain and convey the information needed in a simpler manner to make it easier to work with.

The data received from the robot is set in the time domain, however, when classifying the contact states, the time domain might not be enough to create a clear distinction. By mapping the data onto a function that depends on frequency, the torque data can be decomposed into terms of the intensity of its constituent contacts. As a result, an unintended contact is easier to identify and classify as a collision instead.

Spectrogram

A spectrogram is a visual representation of frequencies of a given signal over time. It can also be defined as an intensity plot of the Short-Time Fourier Transform (STFT) magnitude [43]. This way, a sequence of fast Fourier Transforms (FFT) applied to smaller windowed data segments can be shown in a plot and allow an overlap of maximum 50% in time to encapsulate the different magnitudes. In this case, the magnitude of the value represents the amplitude.

Simply applying FFT to the robot's data will result in showing the different frequencies only and the time domain will be completely absent. However, the time information is vital because it adds the duration of the occurring contact to the final

analysis. Henceforth, the spectrogram solves this issue by depicting the FFT over time as can be seen in Figure 2.1.

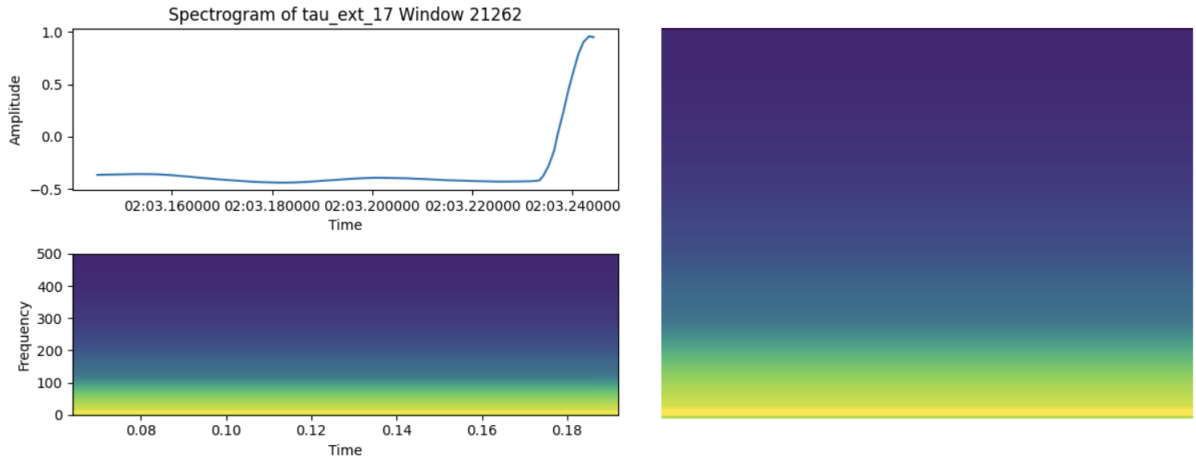


Figure 2.1.: Example of a raw spectrogram of a collision (right) and the time domain plot (left).

Usually, spectrograms are created over a certain time period that is long enough to create smaller windows so that FFT can be applied to each window separately. Nonetheless, Figure 2.1 shows only 100 ms, which is too small to divide into even smaller windows, hence, the figure depicts one full segment of the FFT. This way, the classifier learns to identify contact states based one one FFT segment, to react as quickly as possible. Thus, the left plot in Figure 2.1 displays a sudden increase in the amplitude which is mapped into the frequency domain through the spectrogram. As the window spans over 100 ms, the entire spectrogram shows a collision. Thus, the CNN will be fed with the raw spectrograms as image inputs. The spectrograms are created using Matplotlib's function *specgram*() [47].

2.1.1. Functionalities of the Optimisation Methods

The individual optimisation methods that are used to optimise the CNN, work in different ways, which is why they are chosen in the first place. Optimising a neural network is essential, however, it is only beneficial if the optimal method is chosen for the corresponding CNN. In the following, the broad functionalities of each implemented optimisation method will be described.

Random Search Optimisation

Random Search is a simple approach that randomly chooses possible combinations of parameter values within a defined range. Each dimension has a variety of possible values. This makes it possible to use more values for each variable, because the chance of choosing the same value twice is lower compared to a similar method called Grid Search. As Grid Search tries every possible combination in a defined grid of values, it can only work with predefined values, which makes it less efficient [1]. The advantages of using Random Search instead of Grid Search have been shown in several experiments [3], which is why only Random Search is chosen.

Bayesian Optimisation

Random Search, however, does not take previous measurements into account. In comparison, Bayesian Optimisation offers a sequential model-based approach. This way, a prior probabilistic distribution is established that is refined throughout the optimisation process. The posterior represents an updated distribution over the optimisation function and uses exploration and exploitation to achieve better results based on historic data [40].

Hyperband Optimisation

Estimating the prior distribution is expensive, which is why there are other cost effective approaches, for instance Hyperband Optimisation. Random Search and Bayesian Optimisation are both part of black box optimisations, but there are also multi-fidelity optimisations, such as Hyperband. Hyperband is an extension of the Successive Halving method [17]. Using Successive Halving, the algorithm has a set budget and a set number of configurations. During each iteration, the best performing half of the configurations is kept and the process is repeated until only one configuration is left [17]. Successive Halving has the disadvantage of an existing trade-off between the number of configurations and the number of cuts to get to the limit of the budget. For this reason, Hyperband proposes an alternative method which performs Successive Halving with different budgets to find the best configurations [24].

These three optimisation methods are all part of the KerasTuner library and are used on the CNN built with TensorFlow [32]. However, there are also three inner

optimisation methods that are utilised during the compilation of the neural network. Only the *learning rate* value is be passed to these functions.

SGD stands for stochastic gradient descent. It uses a momentum value that can accelerate the gradient descent in the right direction and dampen oscillations [45].

Adam is an optimisation method that implements stochastic gradient descent. It is based on adaptive estimation of first-order and second-order moments [18].

RMSprop, short for Root Mean Square Propagation, aims to maintain a moving average of the square of gradients [21].

2.1.2. The Programming Environment

Developing a CNN to classify the three states is a very complex task and implements important packages, the preferred way to run the entire process is by creating a virtual environment first. This can be easily done and will not interfere with other already installed packages on the device. Nonetheless, there are 7 main packages to install:

- **OpenCV:** a library for algorithms in computer vision [34],
- **TensorFlow:** a library for machine learning purposes [56],
- **Keras Tuner:** a library for optimising TensorFlow models [32],
- **NumPy:** a library for programming with Python [29],
- **Scikit-Learn:** a library for machine learning in Python [35],
- **Scikit-Image:** an image processing library for Python [61],
- **Imutils:** a library for better image processing [63],
- **Matplotlib:** a library for interactive visualisations [47].

These packages are installed and imported into the neural network environment. OpenCV is used during the prediction process to predict the correct label of a collision-aware contact state and draw it onto the image [34]. TensorFlow offers the functions to create the architecture of the convolutional neural network [56]. Moreover, NumPy and Scikit-Learn are able to create the reports for the testing process of the CNN [29] [35]. Adding to this, Scikit-Image and Imutils improve the quality of image processing during both the testing and prediction cycles [61] [63]. To create plots to show the

performance of the neural network, Matplot is used [47]. TensorFlow also offers a visualisation toolkit called TensorBoard that is used to depict the progress of the training cycles [56] [55]. After the neural network is built using the aforementioned libraries, the optimising functions from the Keras Tuner library are implemented [32].

The entire programme is written in Python and divided into several classes to implement object-oriented programming.

2.2. Literature Review

The number of robots working in collaboration with humans has increased over time due to the growing flexibility of their actions [8]. This means that a good communication infrastructure has to be established between all participants, so that working in the same environment is safe but still efficient [11]. However, working with humans makes most encounters unpredictable, especially in natural environments when regular intersections are inevitable. Even implementing collision prevention systems can lead to unexpected collisions, as humans constantly change their paths and cannot be fully predicted in their future behaviour. Grushko et al. [11] have proposed to notify humans before the robot changes its trajectory, so that unnecessary disruptions can be avoided. However, this method does not help the robot to better distinguish between intended and unintended contacts. A humanoid robot that supports care homes for instance, must know the difference between whether an elderly person is accidentally falling onto them or just leaning against certain parts for assistance. Depending on the situation, different reaction strategies follow, and it is essential for the robot to know the distinction. Nevertheless, utilising robots in industrial or domestic settings has the potential of enhancing the abilities and efficiency of all participants [8]. Furthermore, humanoid robots can be used in extreme environments that are not (yet) fully accessible to humans, for instance, in space, underwater, or in nuclear environments [26].

As a result, specific methods for collision detection must be developed. There are two main ways to implement collision detection within a robot's system. The first method is to use tactile sensors by adding artificial skin onto the robot's surface, so that the impact of an occurring intersection can be located and identified [58]. However, this leads to a complex design that needs to be developed for the specific robot in the first place, which makes it more expensive. The second approach does not depend on tactile or visual sensors, but other signal sources. The first basic model-based method was introduced in 1989 by Takakura et al. by detecting collisions

through signals of the disturbance observer [46]. They suggested to set up a threshold level in the estimated disturbance torque which splits the received data into collision signals and ordinary disturbance signals. As soon as a disturbance signal surpasses the threshold, it is classified as a collision. Since then, this approach of using specific thresholds has been used in many cases, for example on the DLR-III Lightweight Manipulator Arm [6][12]. This arm is the same arm that is now part of the humanoid robot Rollin' Justin, whose torque data is used for this work, as further described in section 3.1.1. After many collision detection applications that implemented the signal thresholds, a new approach was introduced by Sotoudehnejad et al. [44] that used time-variant thresholds instead. This way, the timing of the intersections can be taken into the final evaluation, so that the classification is not fully dependent on the predefined constant thresholds. Throughout the years, it has been established that model-based approaches towards collision detection are more cost effective compared to using tactile sensors. However, model uncertainty and disturbances are likely to decrease the recognition accuracy of the collision detection system because signals are dynamically variable and constant thresholds do not adapt well to changes.

Thus, a new method is needed, that detects collisions without the use of either tactile sensors or thresholds. This can be achieved by implementing a deep neural network (DNN), as proposed by Heo et al. [13]. Their DNN uses joint signals from the robot as input and returns an evaluation of whether a collision occurred or not. Specifically, they used a CNN which achieved the best accuracy. CNNs get their origin from the way the brain processes data, especially images. Hubel and Wiesel found two types of visual cells in the brain that could be imitated in pattern recognition. While the *simple cell* resembles a convolutional layer which is supposed to detect important features, i.e. edges or colour gradients, the *complex cell* has the same behaviour as the pooling layer which puts the detected features into a context and minimises the input size [15]. Fukushima created the connection between the introduced cell types and the layer by calling it *neocognitron* [10]. But one of the first real implementations of a CNN was done in 1989 by Yann LeCun et al. working on recognising hand-written digits [22]. Nowadays, CNNs have several application areas from natural language processing [41] to time series forecasting [64]. Thus, CNNs play an even greater role in image recognition now. Based on the observations that intended contacts return different frequency characteristics than collisions, Kouris et al. propose a new method to discriminate collisions from contacts. Using the FT, the frequency components of a signal are separated which minimises the error and leads to easier distinctions between the two intersections [20]. Extending this method

because of an assumption by Cho et al., which suggests that external joint torques show a faster rate of change during collisions [2], Kouris et al. propose another method which uses the FT to monitor the rate of change instead, which is faster and more accurate [19]. They argue that using the FT to focus on the sudden change of separate frequencies, it becomes faster and simpler to determine whether the robot collides or interacts with another object.

Given these points, this work proposes a novel approach for efficient collision detection by incorporating a recommended CNN with torque data that is depicted in a frequency domain by applying FT. Using the best performing methods in a combination to improve collision detection, the following sections will describe and analyse the new collision-aware contact classifier.

3. Discriminating Collision from Contact

As already established in section 2.2, there are many different ways to detect collisions on a robot. In the following, it is further described, how this thesis introduces a new approach to discriminate Collision from Contact by implementing the FT and a CNN. In the end, the new classifier has to exceed the performance of the developed threshold classifier in section 3.1.3 in order to show its efficiency.

3.1. Experiment Set Up

In order to train the CNN, a dataset is needed. As the classifier should recognise three different contact states (No Contact, Contact, and Collision), enough data must be generated that shows each state separately. In the following, the set up of the experiments is further described.

3.1.1. The Robot

All data that is used originates from the German Aerospace Center's humanoid robot Rollin' Justin. The robot consists of 43 joints and 9 additional links that provide flexibility and several degrees of freedom. Generally, Rollin' Justin has 51 position sensors, 51 degrees of freedom, and 41 torque sensors. Rollin' Justin parts entail a head with cameras as eyes, a torso located on top of a moving platform with four wheels, and two arms with four-fingered hands. In its default upright position, the robot is as tall as a human, 190 cm tall and 90 cm wide [23]. However, once the robot is in action, the height and width can be adjusted. The fact that the robot has a body similar to humans makes it easier to use in environments that are initially designed for humans only. There are seven joints per arm, and each arm weighs 14 kg. Hence, the important data for this task is focused on the torque values of the arms. The

Figure 3.1 below shows the position of the 14 joints located within the arms, and the name of their associated torque value.

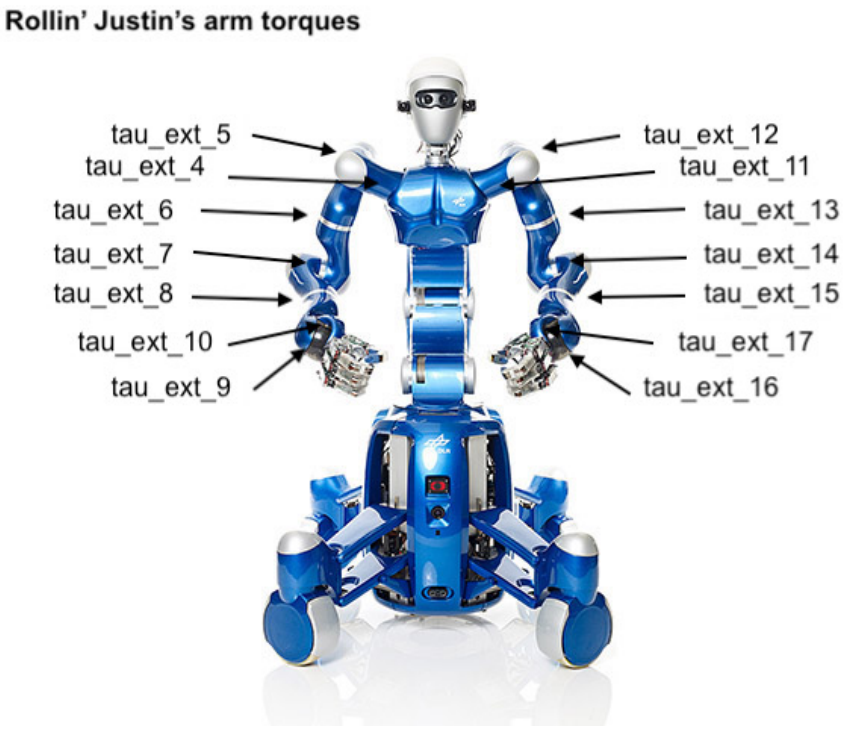


Figure 3.1.: Overview of the robot's joints.

As can be seen, the robot is shown from the front, hence, the left arm, which actively wipes the solar panel, is on the right hand side with the torque sensor values from 11 to 17, whereas the values from the right arm are shown on the left hand side from 4 to 10.

The robot is able to do several different tasks that involve movements of specific joints. This thesis uses the logged data that has been generated during the action of wiping a solar panel. In general, this action usually takes up to 30 seconds and involves several smaller steps which are pictured in Figure 3.2.

Once the robot is commanded to clean the panel, its first task is to move towards the solar panel if it is not already positioned in front of it. Then, the robot needs to grab the wiper, which is attached to the left side of the moving platform, with the left hand. The third step is to place the wiper at the top right corner of the panel. As soon as the wiper is positioned, the robot starts to wipe from right to left. This marks the fist contact period within the action. It starts with placing the wiper onto

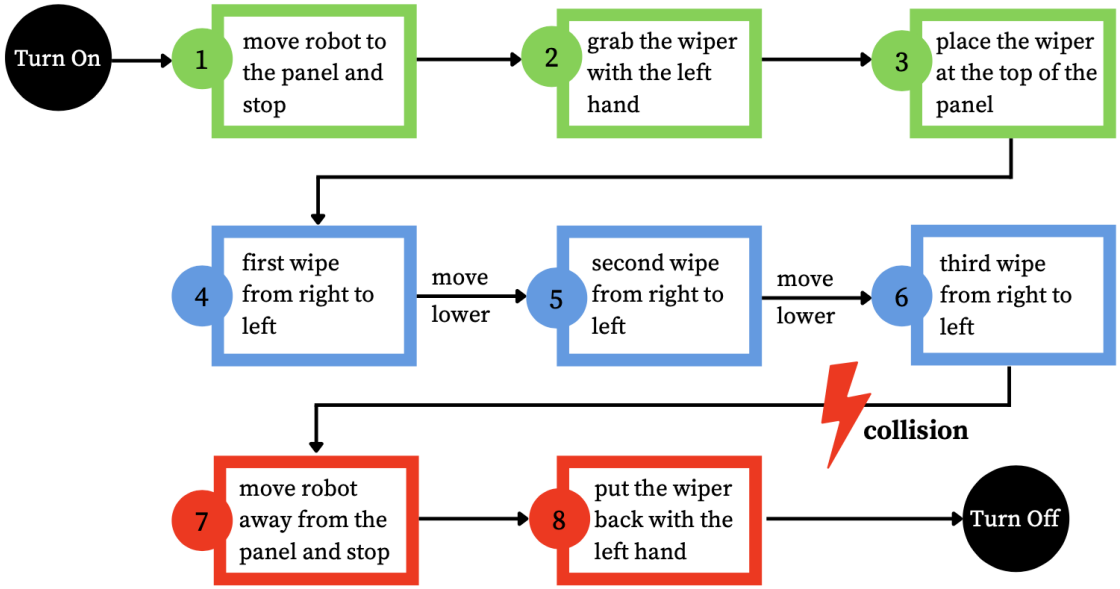


Figure 3.2.: Overview of all steps involved in the action of wiping a panel.

the panel and ends, once the wiper is lifted again after arriving at the end of the panel. In this case, a contact is not only the point when the robot first connects with the panel through the wiper, but it is a constant contact because the wiper is pressed and moved along the entire width of the panel. After the first wipe, the arm of the robot moves lower and places the wiper again on the right side of the panel to do the second wipe. Hence, the second contact period starts. The panel leaves room to wipe three times, hence, the arm moves lower again and wipes a third and last time. Usually, the action would end with the robot moving away from the panel and putting the wiper back. However, somewhere between the end of the third wipe and the moving away, a collision with the left arm is initiated. This is why all log data include three contact periods from the three wipes, and a collision towards the end of the action. In reality, a collision can happen at any time, during any action.

Figure 3.3 shows Rollin' Justin during the action. At the top, the beginning of the contact period can be seen. Once the wiper is fully across the panel, the wipe is finished and the wiper is lifted to be moved one wiper width lower and repeat the wipe.

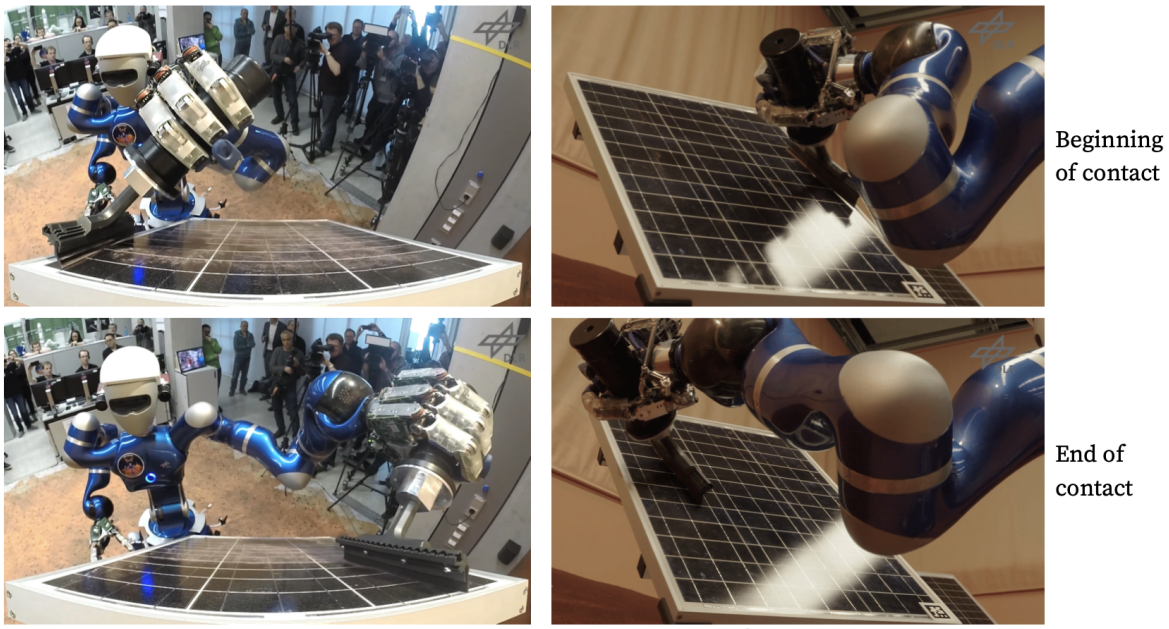


Figure 3.3.: Rollin' Justin during the action of wiping a panel.

3.1.2. The Dataset

The dataset consists of several log files from the humanoid robot Rollin' Justin. Specifically, the data which is received from the torque sensors will be used as they provide information on the three states.

As all tasks are done utilising only the left arm, and the collisions also only happen to the left arm, the torque values from the right arm will be ignored. Figure 3.4 shows a comparison of the torque data from the right arm and the data from the left arm. The graphs are positioned to mirror the view of the robot in Figure 3.1. The figure show the change in the amplitude in the upper plot and the spectrogram in the lower plot of each joint. It can be seen that the spectrograms of the right arm values only show low amplitudes compared to the left arm values. The intensity of the contacts and collision are interpreted through the brighter colours. The purple highlights on the graphs of the left arm show the three contact periods. Their spectrograms also show several bright columns with high amplitudes.

Looking more closely at the two joints which are directly connected to the hands, joint 10 on the right arm and joint 17 on the left arm, the differences are depicted in Figure 3.5. It can be seen that the spectrogram of the right arm does not show any great spikes where a contact or a collision could have happened. Comparing this to

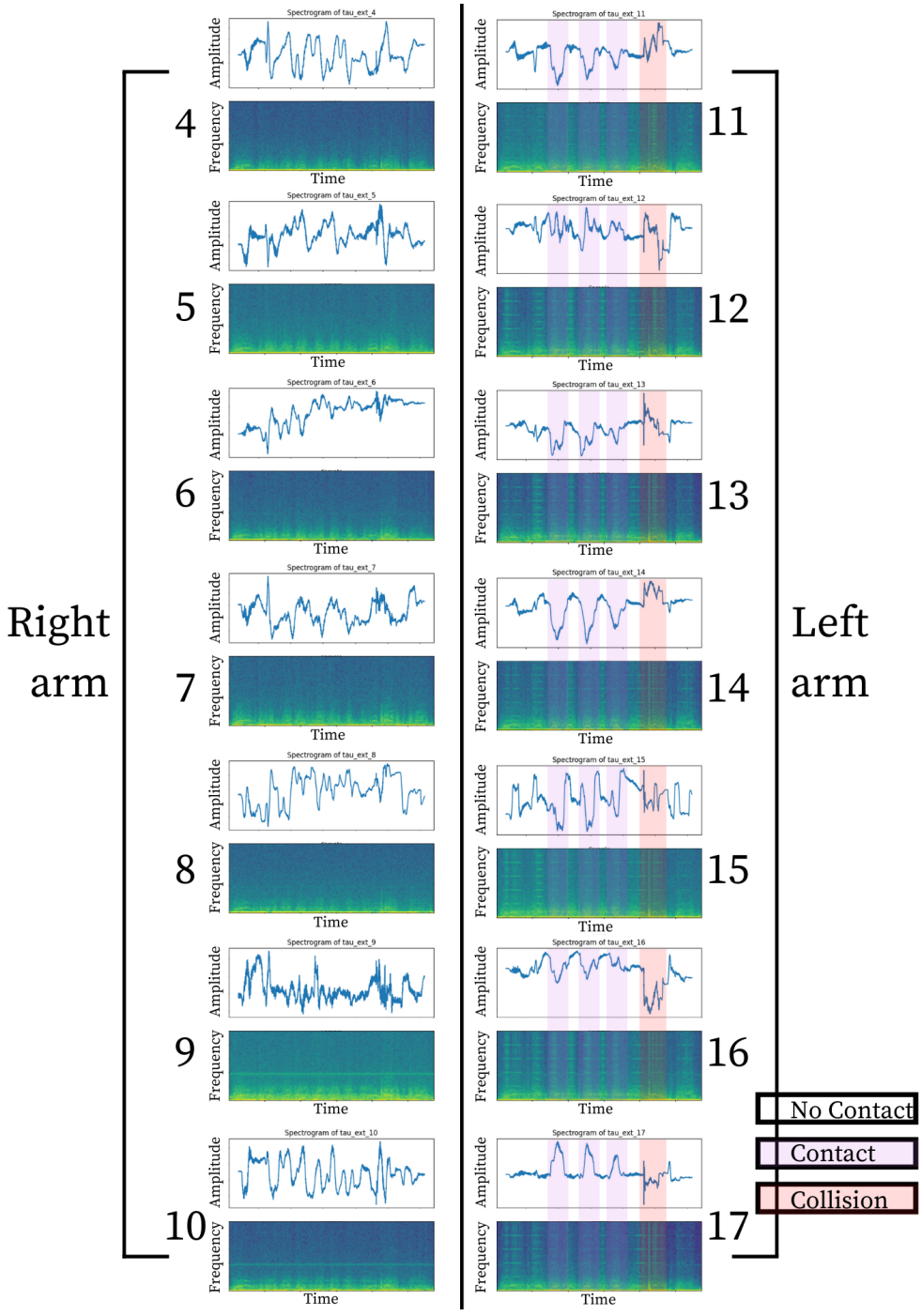


Figure 3.4.: Comparison between right and left arm torque values.

the spectrogram of the left arm joints, the plot shows a clean bright yellow line after the 20 second mark has been passed. This clearly shows a collision, which supports

the fact that the collision happened with only the left arm.

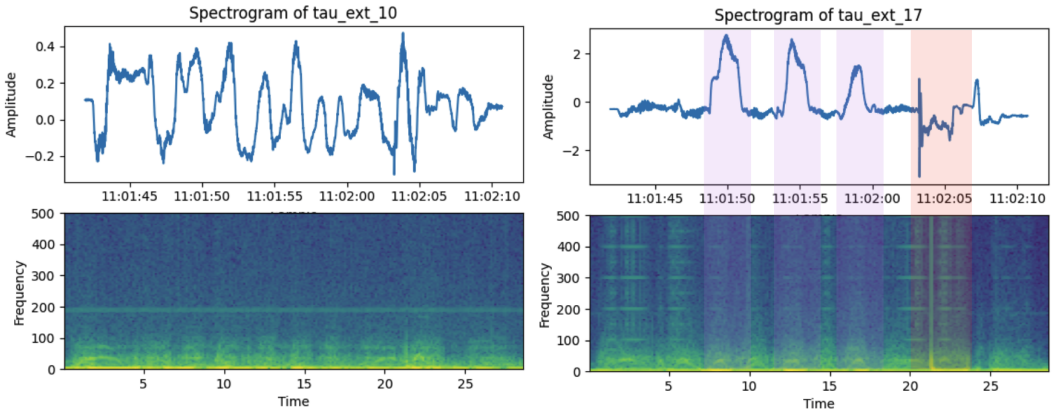


Figure 3.5.: Closer comparison of joint 10 (right arm) and joint 17 (left arm).

The upper plots also show the different ranges of the amplitude. It becomes evident, that the range for joint 10 is between -0.3 and 0.5, whereas the range for joint 17 is between -3 and 3, which explains why the amplitudes are more visible in the spectrogram of joint 17. Thus, only the torque data from the left arm is preferred for creating a dataset on collisions and contacts.

Partitioning

The dataset is used for the training, testing, optimising, and prediction processes. This dataset entails over 4,000 images of three different contact classes from all seven joints of the left arm across six different log files. The images are generated using the *specgram*() function, which is part of the Matplotlib library [47], and labelled in specific annotation files. The contact classes are as follows: No Contact, Contact, Collision. Each image should be classified as either of those labels.

The dataset is divided into three equally important parts:

- **Meta:** This stores a clean image of each contact class.
- **Train:** Here, the images are grouped by their contact class and given a separate directory. Each class shows 1,134 images of one single state from different joints and logs.
- **Test:** This directory consists of 690 images (230 per class) to test and evaluate the accuracy of the model. Some of these images include states that might be

easily misclassified because of their resemblance with other states to see if the network is trained well enough.

An example of images pre-labelled as Collision from all three directories are shown in Figure 3.6.

Figure 3.6.: Input image examples labelled as Collision from each directory.

Each directory is accompanied by their own annotation file that saves the labels of each image. The annotations consist of a table with 4 consistent columns:

- **Path:** The path and filename of the corresponding image,
- **ClassId:** there are 3 contact classes, each belongs to one particular state only,
- **Width:** width of the image (in this case 573 px),
- **Height:** height of the image (in this case 420 px).

Image Format

Each image contains a spectrogram that spans over 100 ms. This allows enough time to avoid further delay of the reaction after a collision. The images all have the same property values with a size of 573x420 pixels. However, in order to save computational power and time, the images are resized to 32x32 pixels. As 100 ms are too small for the function to create several spectrogram instances, resizing the image has no impact on the image itself.

Classification

When recognising and classifying the contact state, the format is predefined. There must be only one entry per test set which consists of the image filename (i.e. 1.png) and the assigned class ID (i.e. 2). This way the best probabilities are calculated and return a prediction accuracy.

Dependencies

As already mentioned before, it is important to include the spatial dependencies of the images within the network. This is done by implementing the CNN. However, this does not guarantee optimal results. Hence, the Train directory includes transitional images that show the state right before it changes to another one, so that the network is trained better.

Window Information

The purpose of the collision-aware contact classifier is to recognise contact states of the robot, including collisions. This classifier is then implemented to help the robot look out for undesired contacts and react on time without too much damage. In order to be as quick as possible, the window size must be optimal, so that collisions are found immediately.

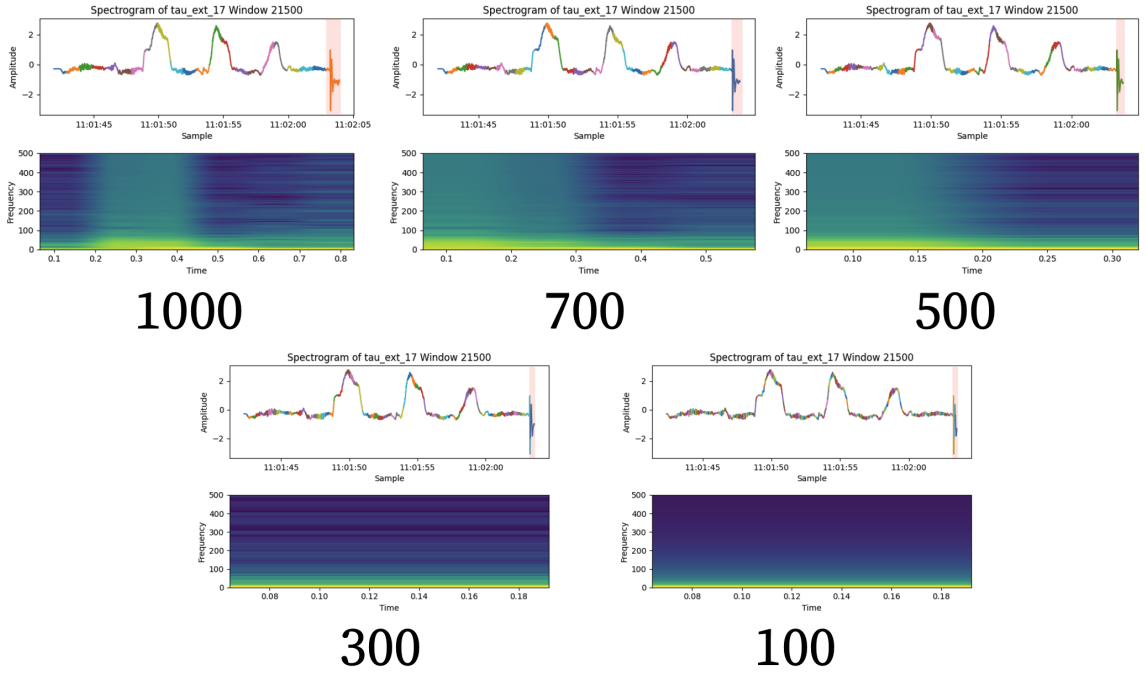


Figure 3.7.: Collisions portrayed in different window sizes.

After experimenting with different window sizes, it could be established, that 100 ms is the best option. Figure 3.7 depicts spectrograms and their respective time domain plots in different window sizes. In order to create these spectrograms, certain values are needed. The sampling frequency of the robot is 1 kHz and because the

entire action takes less than 30 seconds, everything is calculated in milliseconds, which provides more data to work with. Additionally, the windows stride along the x-axis with a step size of 1 ms and 50% overlap, so that no data is accidentally missed. Regarding window sizes, five different values have been trialed with at the beginning: 1000 ms, 700 ms, 500 ms, 300 ms, and 100 ms. The spectrogram at 21.5 seconds showing a window of 1000 ms has a very bright part at the beginning, which is the collision. However, until this image is generated, the collision has already passed and it will take the robot time to recognise it in the first place. Using the same time stamp but 700 ms as a window size, the collision is happening at this exact moment, which applies to the image with window size of 500 ms, too. However, once the window size is below 500 ms, it can be seen that the image shows the state after the collision has happened. Hence, when arriving at the time stamp of 21.5 seconds, the initial collision is already over, all the data that is shown at this point is only the subsidence. The 100 ms window shows specifically, that the Collision state has already passed. Hence, when using a window size as big as 1000 ms, 700 ms, 500 ms, or 300 ms the classifier risks the robot reacting too late. If the window size is small enough, for example 100 ms, it is easier to detect a contact or a collision as soon as possible and pass the information on for possible reactions that can mitigate the damage.

In order to create a collision-aware contact classifier that recognises the different states of the humanoid robot, first, a comparable state-of-the-art classifier needs to be created.

3.1.3. State-of-the-art Classifier

The state of the art suggests, that a collision can be detected solely by looking at the differences within the forces. Once a sudden spike appears in the data, it is classified as a contact or a collision depending on the amplitude of the force.

This thesis hypothesises that this assumption is not enough to classify collision-aware contact states. There are many factors that count towards creating a collision. The following questions should be considered when trying to classify a collision or a contact:

- When does it happen? (i.e. during an action or in a resting state?)
- Where does it happen? (i.e. on which surface? Is there resistance?)
- Which joints or parts of the robot are affected? (a collision on the torso might show differing data compared to a collision on an arm)

- How much force is involved?
- How long does the contact/collision take?

Generally speaking, each collision is also a contact, but not each contact is also a collision. Here, it needs to be considered whether the contact is desired (i.e. during an action in which the robot interacts with certain objects) or undesired (hence, a collision).

Before the novel approach is implemented, the state-of-the-art performance needs to be established. By looking intensely at the dataset, thresholds are identified. As the CNN classifier uses the spectrograms as image inputs, the threshold classifier must use them, too, to compare their performance. For this purpose, three pixels are chosen to receive the RBG values and calculate the grayscale value. The pixels are as follows:

1. 280, 32
2. 280, 280
3. 280, 400

The pixels are chosen carefully, in order to find locations that show different RGB values depending on the state, as can be seen in Figure 3.8.

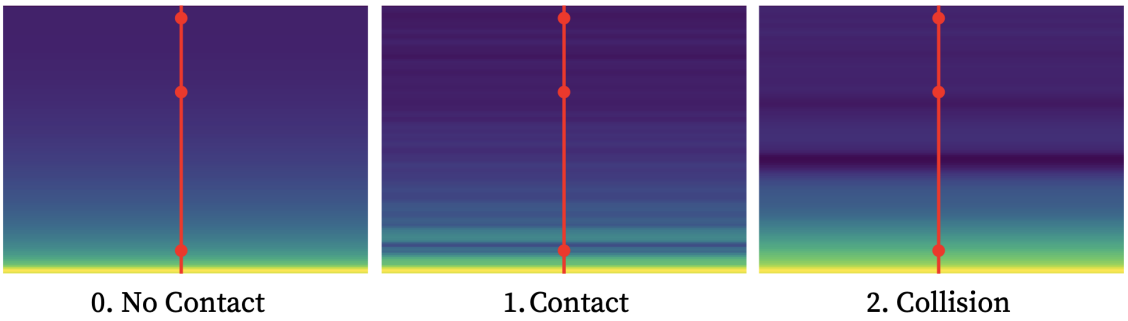


Figure 3.8.: Location of the three RGB pixels.

The spectrograms show the intensity of the contact by applying brighter colours, hence, this is how the classifier is supposed to recognise the correct states. However, comparing three separate values for R, G, and B respectively, already creates three dimensions. Finding thresholds for each of these dimensions and differentiating the classes as well, the classifier becomes too complex. Thus, processing the images

by calculating the grayscale value of a specific pixel allows more complexity within the thresholds. Nevertheless, the RGB values are not represented equally in the spectrograms. This is improved by using the weighted grayscale method, which weighs RGB according to their wavelengths. The formula is as follows:

$$Grayscale = 0.299R + 0.587G + 0.114B.$$

By calculating the grayscale of each spectrogram entailed in the training dataset of the classifier, certain thresholds are identified, as can be seen in Table 3.1.

	Thresholds		
Class	Grayscale 1	Grayscale 2	Grayscale 3
No Contact	44 - 56	74 - 81	132 - 149
Contact	<39 or >81	<69 or 82 - 102	<131
Collision	40 - 43 or 57 - 80	70 - 73 or >103	>150

Table 3.1.: Thresholds for each grayscale value and each class.

As shown in the table, if a certain grayscale value lies within the space of a certain class, it is classified as such. In order to get these thresholds, all grayscale values of the dataset need to be evaluated in the first place. Some thresholds overlap, which is not acceptable for a classifier. This leads to several thresholds for one class, as shown for contacts and collisions in Table 3.1.

Forthwith, the thresholds are used to build the state-of-the-art classifier and test the performance on the dataset. The following Table 3.2 shows the final results. There are three separate datasets for each class with exactly 1,134 images, which are later used during the training process of the CNN classifier. However, as the threshold classifier does not learn, these images are treated as new. Using the final thresholds, the classifier can classify an image as: No Contact, Contact, Collision, or Unknown. The Unknown class is added in order to see how well the thresholds have been chosen. As there are three grayscale values, the classifier is implemented three times, each time with a different grayscale value. Thus, for the first iteration, only the thresholds of the first grayscale value are used.

The results show three final average accuracies, one for each grayscale value. The highlighted classes and their respective accuracies show how well the classifier performs for the correct class. For instance, all three classifiers achieve at least 99.9% accuracy for the class No Contact on the database that only has images of said class.

Classification		Accuracy		
Dataset	Class	Grayscale 1	Grayscale 2	Grayscale 3
No Contact	**No Contact**	**100%**	**100%**	**99.9%**
	Contact	0%	0%	0%
	Collision	0%	0%	0%
	Unknown	0%	0%	0.1%
Contact	No Contact	65.6%	51.1%	40.7%
	Contact	**6.4%**	**20.2%**	**28.3%**
	Collision	22.3%	13.9%	24.9%
	Unknown	5.6%	14.8%	6.1%
Collision	No Contact	37.2%	16.0%	9.1%
	Contact	16.1%	62.4%	11.4%
	Collision	**41.1%**	**14.1%**	**77.9%**
	Unknown	5.6%	7.5%	1.7%
	Average Accuracy	**49.2%**	**44.8%**	**68.7%**

Table 3.2.: Performance of the threshold classifier with all grayscale values.

Thus, the thresholds are efficient for recognising the resting state of the humanoid robot. However, when looking at the other two important classes, the results show mixed accuracies. Using the first grayscale thresholds on the Contact dataset, only 6.4% of the images are actually classified as contacts. Comparatively, the classifier recognises 65.6% of the images as No Contact and 22.3% as Collision, which is evidently a wrong classification. Although the classifier has the highest accuracy for Collision on the corresponding dataset, 41.1% accuracy is not fit for use.

Moreover, using the second grayscale values for the classifier, 51.1% of Contact images are classified as No Contact, only 20.2% of images are correctly classified. The Contact dataset also has the highest number of unknown classifications. However, 62.4% of Collision images are classified as Contact, compared to the 14.1% of correct classifications.

Lastly, the third grayscale value performs better on average. Even though only 28.3% of Contact images are recognised as such, it is still higher than the accuracies achieved by the other two classifiers. Especially, looking at the 77.9% Collision accuracy on the Collision dataset, it can be said that the last classifier achieves the

best results.

When calculating the average accuracies, it is clear to see that the third grayscale classifier performs better with an average of 68.7% compared to 49.2% using the first classifier, and 44.8% through the second classifier.

Additionally, a pattern is identified regarding the misclassifications. Even though the No Contact dataset is classified correctly using all three classifiers, the thresholds also overlapped into the Contact and Collision space. During all three trials do the No Contact classifications on the Contact dataset exceed the Contact classifications by 12%-59%, which is a lot, if the average accuracy is between 45%-69%. This pattern is also continued inside the Collision dataset, though not as dominant. However, the distinction between Contact and Collision recognition in their respective datasets is very close.

Drawing conclusions from these results, it can be said, that the difference between the three states is hard to detect when focusing only on thresholds. Contact images are classified as No Contact or Collision, the same applies to Collision images. Only No Contact images are well recognised. However, the state-of-the-art classifier achieves a maximum average accuracy of 68.7%. This is the threshold that the CNN classifier needs to exceed. Additionally, as soon as the state-of-the-art classifier is implemented to recognise contact states during a new action of the robot, entirely new thresholds have to be established beforehand, which leaves little room for adjusting to changes. This is why the new approach is introduced.

4. Methodology

There are several steps involved when developing a collision-aware contact classifier. Even after the full development stage is completed, the classifier must be constantly optimised and adjusted to achieve the best possible outcome at all times. The methodology consists of all stages that lead to the final framework of the classifier. The first stage introduces the concept of the classifier in section 4.1, which is followed by a detailed description of the development stage in section 4.2. Furthermore, once the classifier is fully developed, it has to be optimised using specific optimisation methods, which is documented in section 4.3. Finally, section 4.4 goes further into the correct implementation of the framework and its usage.

4.1. Concept

As there is no universal rule that describes the difference between a contact or a collision when looking at torque data only, the classifier should learn from examples and find connections that might be harder to identify for the human eye. Depending on the torque and time involved in the contact, thresholds can be established that show limitations. However, these thresholds are not fixed numbers but variables that change with each action, joint, or collision. There are soft and slow collisions that might be classified as desired contacts, and there are strong and abrupt contacts which might be classified as collisions. Hence, the entire context of the situation must be gathered before it can be labelled.

The concept of the collision-aware classifier is to identify the current contact state of the robot through torque sensor data. This way, the robot is able to differentiate between desired contacts and external collisions and react accordingly in time to limit potential damages. There are three different states to be detected: No Contact(0), Contact(1), or Collision(2). Through these three states, the classifier is bound to recognise whether there is an interaction happening at the time and whether the interaction is desired or not. The input data are generated spectrograms, which are visual representations of frequencies (in this case, torque values from the torque

sensors) over a specific time period. A window length of 100 ms is chosen deliberately to support real time collision detection. Using the torque values of the robot as training input of the classifier, makes it possible to discriminate Collision from Contact successfully.

4.2. Development of the Classifier

The classifier is developed using mainly the TensorFlow infrastructure. Combining the generated training and testing data with the convolutional neural network, a framework for training, testing, and optimising the classifier is built.

This section focuses on all parts connected to developing this collision-aware contact classifier.

4.2.1. The Neural Network

The deep learning frameworks from TensorFlow are used to build a convolutional neural network (CNN) that could classify the contact-aware states. A CNN is chosen in this case because CNNs have become a standard for image classifications in machine learning [28]. TensorFlow and Keras offer a sequential model to implement the neural network [57]. The sequential model makes it possible to build the model layer by layer which is described step by step within the framework.

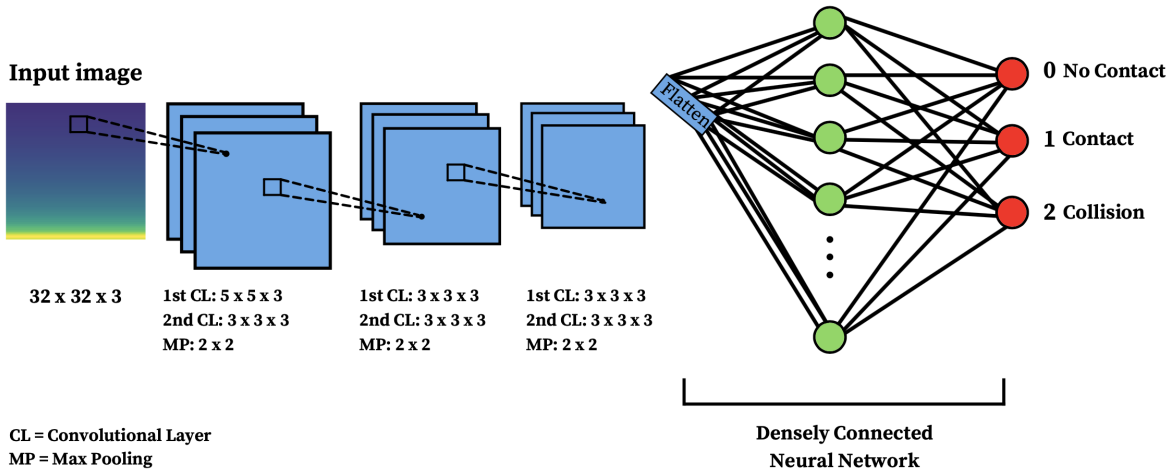


Figure 4.1.: Architecture of the CNN.

The CNN is divided into several different layers, as shown in Figure 4.1. The first

layer is a convolutional layer with four main parameters: $filters$, $kernel\_size$, $padding$, and $input\_shape$ [48]. The $input\_shape$ value is only passed in the first convolutional layer to set the dimensions of the image with $height$, $width$, and $depth$. When uploading the input images, they are resized to 32x32 pixels. The $depth$ value describes whether the image shows colours or is in grayscale. In this case, the spectrograms are in RGB, hence, they are passed as coloured images. The $filters$ determine the dimensionality of the output space. The *kernel_size* specifies the width and height of the two-dimensional convolution window. In order to capture the spatial and temporal dependencies properly, the first kernel is of size 5x5 to learn larger features. Adding to this, same padding is applied through the *padding* parameter to keep the image dimensions the same after each convolution. After defining these parameters, an activation function must be declared. For this purpose, *relu* is chosen, which is a rectified linear unit that removes negative values and turns their values into zeros. This also adds to a better run time when training the network [37]. The convolutional layer ends with specifying the channel order, in this case, *channels last*. This means that the three-dimensional array that represents the image data has the information about the colour channel (RGB) as the last parameter. Going through all these steps creates the first convolutional layer. The following convolutional layers have the same parameters and only change values in $filters$ and *kernel_size*.

After each set of two convolutional layers follows a pooling layer which decreases the computational power to train the network. Here, *max pooling* is implemented with dimensions 2x2. *Max pooling* returns the maximum value of each kernel of the image, so that the spatial dependencies can stay, although the computing costs are lowered. It also suppresses noise while reducing the dimensionalities [53].

The CNN consists of three such sets where two convolutional layers and one consecutive pooling layer are combined. The first set is described above and creates the first stage of breaking down the image into several smaller parts. After the first set, another set follows, which applies a new convolutional layer twice successively before applying *max pooling* again. Here, the convolutional layers only have a 3x3 kernel, which leads to a deeper neural network structure. The last set also implements two convolutional layers successively and a last *max pooling*. These convolutional layers deepen the network even further but the *kernel_size* stays the same.

After running through six convolutional layers, the images need to be flattened into a column vector. The flattened layer leads into the densely connected layers to create the head of the neural network [51]. This network can now be treated as a regular feed-forward ANN. The density and activation function (*relu*) are added

with a dropout of 0.5. The dropout is needed to work against overfitting during the training process of the ANN [49]. The value specifies the probability with which a node will be ignored to reduce the network. After the training stage, these nodes are reinserted with their original weights. Overfitting is decreased by avoiding training all nodes of a network with all training data. Hence, dropout is also added to each set of convolutional layers after *max pooling* is applied. The last stage of the complete convolutional neural network is the softmax layer. The softmax classification returns probabilities for each class label that the image could be a part of [54]. The resulting accuracy depicts the highest probability.

4.2.2. Generating Data

The dataset is generated from existing log files, which entail data from the robot that are documented during actions. This data is used to label sequences as one of the three states: No Contact, Contact, or Collision. However, there are several steps to include before deciding on the specific parameters of the dataset. As the classifier should be based on the Fourier Transform (FT), the raw data from the log files needs to be altered in the first place by applying the FT directly.

The FT can be used for different input data. Using an acoustic signal, for instance the Imperial March passed through a .wav file, the plots are shown in Figure 4.2. The upper graph depicts the signal plotted in the time domain, showing the amplitude on the y-axis. The lower graph is the spectrogram to visualise the frequencies over time. Here, the colours represent the amplitude; the brighter the colour, the higher the amplitude.

Nevertheless, FFT can also be applied to forces. Figure 4.3 shows the plot and spectrogram with input data from the log files of the robot. Specifically, because the contacts and collisions only affect the robot's left arm, the torque values from the joints 11 to 17 (hence, 7 joints) are used. Within the plot, the three consecutive maxima depict the three contacts of the robot with the wiping surface. The sudden spike around 21 seconds shows the collision. Moreover, wherever the amplitude is between 0.5 and -0.5, there is no contact or collision.

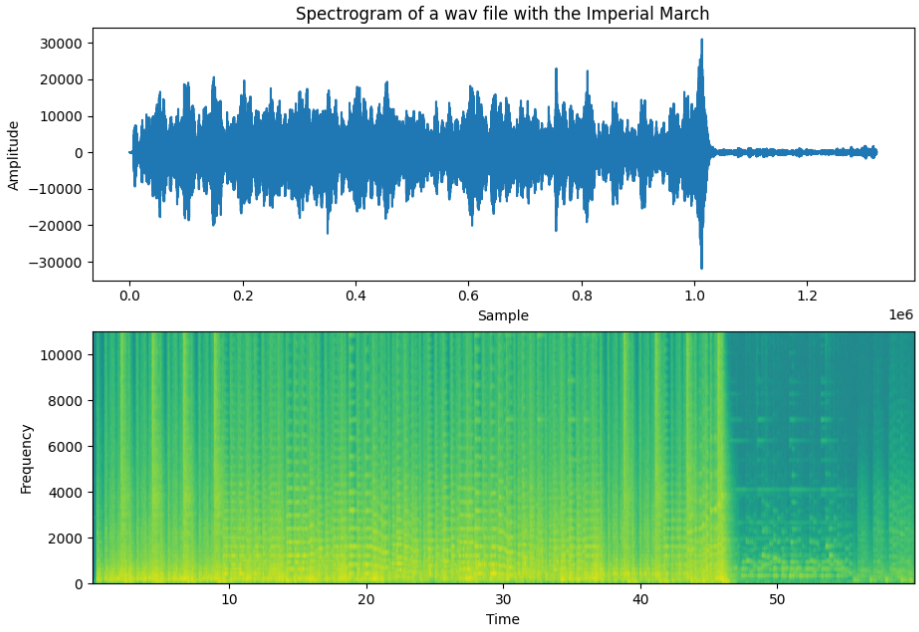


Figure 4.2.: Example of fast Fourier Transform using an audio file with a .wav extension.

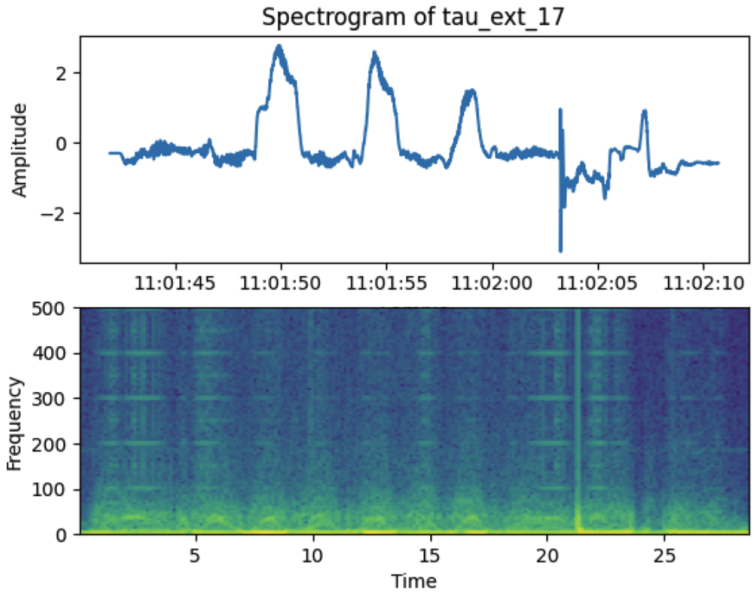


Figure 4.3.: Example of fast Fourier Transform using torque values.

4.2.3. Reducing Overfitting

Overfitting is a common complication when training and testing a neural network. There are too many possible parameter combinations to tune them all to an optimal standard. From the kernel and layer dimensions to the activation function, overfitting some of the values seems inevitable. However, there are techniques to prevent overfitting at certain stages.

Early Stopping

During the training process of the CNN, early stopping is implemented to stop the training of the model before the overfitting process starts. TensorFlow offers early stopping as a function, where it is specified that the loss of each epoch should be monitored. After 3 epochs with no improvement, the training is stopped. This way, the number of epochs varies for each cycle in order to prevent the model from fitting the noise [50].

Dropout

Dropout is another parameter that reduces overfitting [39]. By dropping units out, all their incoming and outgoing connections are ignored temporarily. Whether a unit is ignored or not depends on the fixed probability p, which ranges from 0.1 to 0.6 during the optimisation process. This way a reduced network is trained. TensorFlow also offers a function for Dropout, hence, it is easily implemented as part of the layers of the neural network [49].

Data Augmentation

Another way of reducing overfitting on image data is to augment the data for the training. This is done by enlarging the dataset artificially using label-preserving transformations [42]. TensorFlow offers the image data generator that generates batches of tensor image data with real-time data augmentation [52]. For this CNN the data augmentation is set with values for random shift, zoom, and shear. However, vertical and horizontal flips are set to *False* because the spectrograms will not change their axis during the implementation.

CLAHE

Contrast Limited Adaptive Histogram Equalization (CLAHE) is an image processing technique to improve low contrasts in images [36]. CLAHE is needed to deepen the contrasts shown in the spectrograms and get better accuracies. Some states look similar to one another and sometimes it is hard to tell whether a spectrogram shows a contact or a collision. Thus, CLAHE brings out the nuances and amplifies the transitions between the colours, so it becomes easier to identify the correct contact state.

K-Fold Cross Validation

K-Fold Cross Validation is used to maximise the prediction accuracies by splitting the dataset into two subsets which are specified by the value k. One subset is used for the training process of the neural network, which uses $\frac{k-1}{k}$ of the dataset, whereas the rest of the dataset ($\frac{1}{k}$) is part of the testing phase to evaluate the model [16]. This separation is important to make sure that the model is tested with image data that it is not trained with, to ensure the functionality on new and unknown data [7]. This way, the risk of overfitting or underfitting is decreased and the CNN will not start fitting the noise [25]. The final CNN uses $k = 5$ to ensure that 20% of the entire data is meant for the evaluation of the model.

4.3. Optimisation of the Classifier

After training and testing the neural network with manually chosen hyperparameter values, a base accuracy is achieved. However, this accuracy is improved by utilising optimisation methods. In the following, the different stages and methods of optimising the CNN are described and compared.

There are a number of hyperparameters that can be tuned within this CNN. Moreover, the hyperparameters are connected to each single layer of the neural network. The values can be tuned when creating the initial network or during its compilation. The hyperparameters and their ranges for this task are the following:

- $filters$: 32, 64, 128, 256, 512
- $units$: 32, 64, 128, 256, 512
- $dropout$: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6

- *learning rate*: 1e-1, 1e-2, 1e-3, 1e-4

As described in section 4.2.1, the neural network consists of three sets of layers that entail six convolutional layers in total. Each set of convolutional layers includes values for $filters$ and *dropout*. The $filters$ determine the dimensionality of the output space. The *units* value represents the same as the $filters$ value, however, it is passed to the layer that creates the densely connected neural network at the end. The *dropout* value is further described in section 4.2.3. Lastly, the *learning rate* value is used by the inner optimisation function during the compilation, which can be either *Adam*, *SGD*, or *RMSprop*.

Furthermore, in order to receive a variety of results to compare, the three optimisation methods are used from the KerasTuner library: Bayesian Optimisation [30], Hyperband [31], and Random Search [33].

4.3.1. Establishing the Base Accuracy

In order to start with the optimisation process of the CNN, the base accuracy must be known. This accuracy serves as a standard for the CNN and becomes the threshold that an optimisation method must surpass in order to actually achieve a better accuracy. Only then, the optimisation is successful.

Hyperparameters			
Batch Size	Learning Rate	Epochs	Accuracy
4	1	44	72%
4	0.1	36	72%
8	0.01	24	72%
16	0.1	59	67%
2	0.1	33	66%

Table 4.1.: The five best base accuracies.

Table 4.1 shows the five best base accuracies and the values for their hyperparameters. These hyperparameters seem different from the aforementioned ones, however, this is only because no optimisation method has been used yet. The more complex the optimisation method, the more complex the hyperparameter dimensions can become.

Table 4.1 clearly shows that 72% accuracy is the highest result that is achieved. This means, that by using the classifier with its manually tuned hyperparameter values, the performance is already better than using thresholds as described in section 3.1.3, where it only achieves an average accuracy of 68.7%.

	Hyperparameters		
Set of Layers	Filters	Units	Dropout
1st	32	—	0.2
2nd	64	—	0.3
3rd	128	—	0.4
Densely Connected NN	—	128	0.5

Table 4.2.: Fixed hyperparameter values for the neural network.

Instead of applying the hyperparameter optimisation methods, the neural network is built using a general architecture. As depicted in Figure 4.2, the values increase with each set of layers in order to deepen the network after each stage.

4.3.2. Optimising with the Inner Optimisers

As there are three optimisation methods and three inner optimisation methods that all work differently, the optimisation process is split into two parts. First, the best performing inner optimiser needs to be found, and then the greater optimisation methods can fully focus on optimising the entire CNN. Hence, in the first part, the inner optimiser becomes a hyperparameter itself, with the three algorithms as potential values.

At this stage, the focus lies entirely on the performances of the inner optimisers, hence, the training set and testing set are split using K-Fold Cross Validation with k=20, to achieve the best training performance of the inner optimisers possible.

Table 4.3 depicts the results received after optimising the hyperparameters of the CNN with the three different inner optimisers. Evidently, all methods achieve training accuracies that surpass 92%. However, when looking at the test accuracies, only *Adam* keeps close to the training accuracies, whereas the other two inner optimisers only achieve between one third and two-thirds of the corresponding training accuracy. Thus, it is concluded that utilising *Adam* as an inner optimiser achieves the best performance compared to the other two inner optimisers.

Optimisation		Accuracy			
Optimisation Method	Inner Optimiser	Training	Test	Loss	Duration
Random Search	Adam	97.9%	90.2%	6.0%	01h 15m 03s
	SGD	95.4%	69.9%	10.8%	00h 59m 49s
	RMSprop	96.6%	71.2%	6.0%	01h 09m 25s
Bayesian Optimisation	Adam	93.2%	89.6%	12.0%	00h 17m 10s
	SGD	92.7%	30.7%	16.2%	00h 14m 17s
	RMSprop	93.3%	30.7%	110.8%	00h 16m 14s
Hyperband	Adam	96.7%	87.1%	6.0%	01h 20m 05s
	SGD	95.2%	47.9%	5.8%	01h 23m 49s
	RMSprop	98.2%	38.7%	4.6%	01h 07m 37s

Table 4.3.: Comparison of Inner Optimisers and their performance.

4.3.3. Optimising the final CNN

After establishing the fact, that *Adam* is the best performing inner optimiser for this CNN, one hyperparameter is eliminated. Now, the focus lies on the performance of the neural network when faced with 80% training images and 20% testing images. It is expected that the test accuracies decrease compared to the accuracies in section 4.3.2 because there are fewer images to train with, but more testing images to check against overfitting.

At this stage, two different experiments are conducted. During the first experiment, the aforementioned hyperparameters are added to the range of the optimisation methods, hence, there are four different hyperparameters with four to six different values to choose from. The results are shown in Table 4.4. Each optimisation takes over an hour, with Random Search at almost two hours. However, Random Search achieves the best training accuracy with 98.1%. Hyperband comes close with 98.0% accuracy, but a better test accuracy than Random Search by 1%. Nevertheless, even though Hyperband and Random Search achieve similar results, Hyperband is faster by 34 minutes. Bayesian Optimisation still achieves 94.6% accuracy, but compared to the other two methods, it is not enough. Thus, regarding the duration and the test accuracy, Hyperband achieves the best results, however, Random Search excelled in the training accuracy and the lowest loss value. Thus, Hyperband and Random Search perform similarly well.

	Accuracy			
Optimisation Method	Training	Test	Loss	Duration
Random Search	98.1%	79.7%	9.3%	01h 52m 44s
Hyperband	98.0%	80.7%	10.0%	01h 18m 41s
Bayesian Optimisation	94.6%	68.1%	9.7%	01h 16m 26s

Table 4.4.: Performance of optimising the CNN with four hyperparameters.

Considering the fact, that even though the final training accuracy is already very high, compared to the 68.7% from the threshold classifier and the 72% base accuracy, the loss and duration make the optimisations expensive. Henceforth, a new experiment with alternative settings is conducted in an effort to shorten the duration and decrease the loss of the optimisation methods.

During the second stage, the hyperparameter values are set at the beginning, so that the values increase with each set of convolutional layers as shown in Table 4.2. Only the *learning rate* is integrated into the range of the optimisation methods. This way, it can be seen, whether the neural network performs better if the values of each layer stay the same or increase to deepen the network.

	Accuracy			
Optimisation Method	Training	Test	Loss	Duration
Random Search	96.8%	78.3%	4.4%	00h 17m 46s
Hyperband	85.8%	83.6%	7.0%	00h 02m 43s
Bayesian Optimisation	97.6%	78.3%	3.9%	00h 42m 15s

Table 4.5.: Performance of optimising the CNN with one hyperparameter.

When comparing the results from Table 4.5 with the results from Table 4.4, it can be seen that the training accuracies when optimising only one hyperparameter do not surpass the 98% mark. Surprisingly, Bayesian Optimisation, which achieves the lowest training accuracy in the first stage with 94.6%, achieves the highest accuracy in this experiment with 97.6%, which is only 0.5% less than the general maximum accuracy. Adding to this, it achieves this in 42 minutes, which is almost half the time Hyperband and Bayesian Optimisation need in the first stage. Moreover, Hyperband takes less than 3 minutes to optimise the CNN and achieve 85.8% training accuracy. Even

though this is the lowest optimisation accuracy so far, it is still better than the base accuracy by 13.8%. Comparatively, the loss decreases during all three optimisation processes. As a result, Bayesian Optimisation achieves the best training and testing accuracy but also has the smallest loss value with 3.9%. However, it is interesting to see that Hyperband has the highest loss value in both experiments and does not manage to receive better results than Random Search and Bayesian Optimisation, except for having the highest test accuracy. Thereupon, it is safe to say that Random Search performs better when using multiple hyperparameters and allowing a greater dimension. Then again, Bayesian Optimisation performs best when only optimising one hyperparameter.

	Accuracy	Hyperparameters			
Optimisation Method	Training	Learning Rate	Filters	Units	Dropout
Random Search	98.1%	0.001	64	64	0.2
	96.8%	0.001	—	—	—
Bayesian Optimisation	94.6%	0.01	16	32	0.1
	97.6%	0.001	—	—	—
Hyperband	98.0%	0.001	128	64	0.2
	85.8%	0.001	—	—	—

Table 4.6.: Comparison of the optimised values.

After comparing both experiments separately, Table 4.6 shows the final values that are used to achieve the optimised accuracy. As already stated, the neural network values for the second experiment are shown in Table 4.2 because they increase with each set of layers, whereas the values in the first experiment stay the same throughout each layer. Looking at Table 4.6, a pattern is identified. The best performing value for the *learning rate* is 0.001 in all cases, except for Bayesian Optimisation in the first experiment. However, when using 0.01 instead, Bayesian Optimisation achieves the lowest training accuracy during the first stage. Additionally, the *dropout* values are close to one another, between 0.1 and 0.2, even though the range spans from 0.1 to 0.6. After all, Hyperband and Random Search produce the exact same optimised values, with one exception when Hyperband applies 128 $filters$ instead of 64, which leads to Hyperband achieving 0.1% less accuracy than Random Search. In contrast, all optimisation methods find the exact same *learning rate* value in the second

experiment and use the same neural network structure, however, they all achieve different accuracies. This can be explained by looking at their different ways of optimising a neural network. Here, they are already bound to the fixed values of the neural network, which only leaves the optimisation range for the *learning rate* and the inner optimisation method, in this case, *Adam*. In this case, Bayesian Optimisation achieves the best results because it is able to take previous measurements into account and adjust the probabilistic distribution.

4.4. Implementation of the Framework

Provided that the convolutional neural network is built and optimised so that the classifier achieves the best possible accuracy, the framework is based on the conclusions from the previous sections. The framework provides a tool that can be used freely to train the humanoid robot to recognise the three contact states. The framework is built in a way that allows the extension of the dataset for training, testing, and predicting purposes. In the following, the complete structure and architecture of the final framework is described and analysed.

4.4.1. Class Structure

The framework is written in Python using several additional libraries in order to utilise the most important functions. Consequently, by implementing the framework in an object-oriented programming manner, a clear structure is established. By understanding this structure, the framework is used easily and extended for future applications. Figure 4.4 shows the class diagram of the framework. Below, each class is described in more detail.

Main

The Main class is where all the other classes come together in order to run the programme successfully. Thus, this is where objects are created from each class, to gain access to their functions. Depending on the arguments passed in the function call, the `condition` value indicates whether new images need to be generated instantly for the labelling process at the end, or whether there is a provided directory with images that is iterated through the final framework in order to receive a prediction.

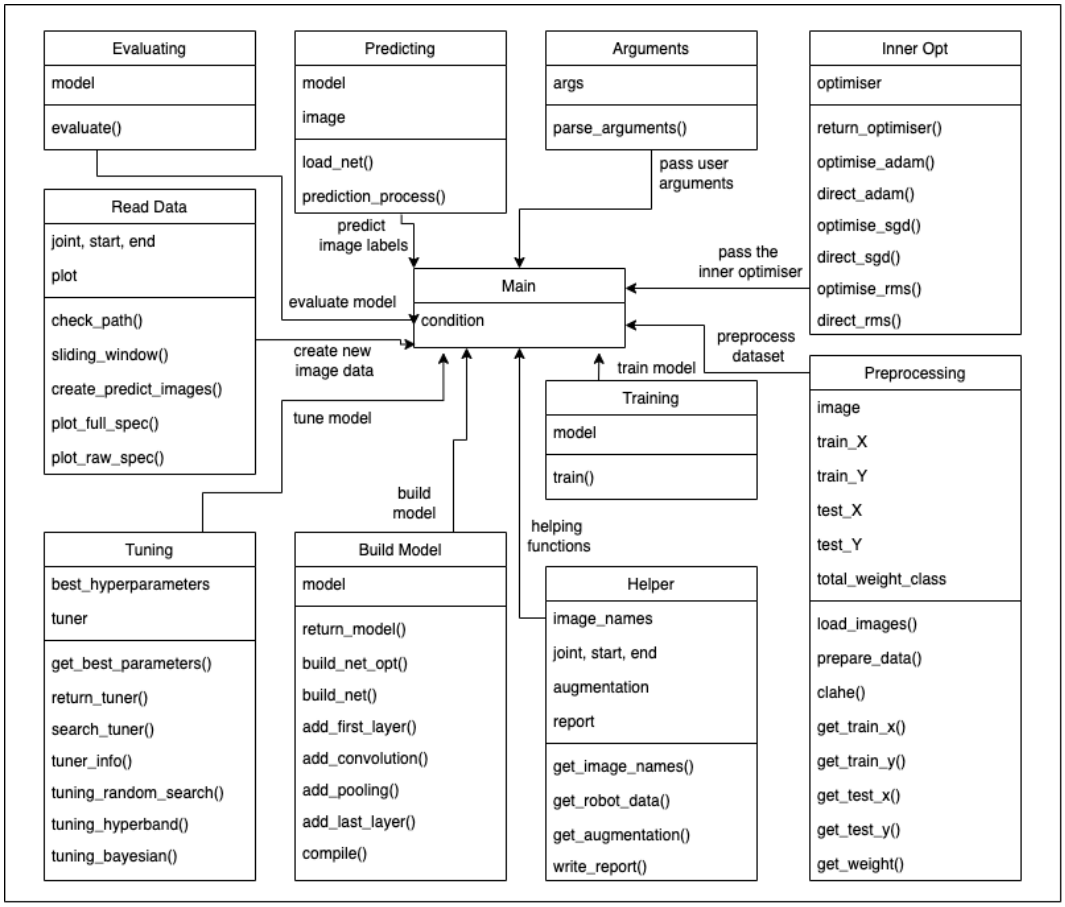


Figure 4.4.: Class Diagram of the Framework.

The programme is called through a command in the Terminal:

```
$ python3 main.py -m -op -inop -d -pr -tr
```

By adding specific arguments to the command, the Main class guides the framework in the right direction to get to the desired result.

Arguments

Granted that the desired result is known before using the framework, the methods are specified through the function call. There are six optional arguments:

- `-model` or `-m`: path to the saved model,
- `-optimiser` or `-op`: optimisation method, can be either `[bayesian, hyperband, random]`,

- `-inner_optimiser` or `-inop`: inner optimisation method, can be either `[adam, sgd, rms]`,
- `-dataset` or `-d`: path to the input data for the training and testing processes,
- `-predictions` or `-pr`: path to the directory with images to predict or a .h5 file to generate new images to predict
- `-train` or `-tr`: indicate the number of hyperparameters to optimise, or skip the training entirely and just predict labels, can be either `[all, one, none, pred]`.

If no argument is passed, the default setting is the instant prediction of the states within the images, as this is the main task of the framework. Each argument has its own default value. All classes use the information given through the arguments to determine the correct output.

Preprocessing

The Preprocessing class takes care of the proper preparation of the input images. In order to minimise overfitting, CLAHE is applied to the images. Moreover, they are resized to 32x32 pixels to decrease the computing power. Using the information from the annotation files that include the correct label of each image, the images are separated into training and testing datasets. These directories are part of the framework and include 1,134 training images and 230 testing images per state, hence 3,402 (80%) training and 690 testing images (20%). This way, K-Fold Cross Validation is applied to ensure the successful prediction of the labels.

Helper

The purpose of the Helper class is to provide the framework with sub-functions that cannot be assigned to any other class solely. These functions are used by any class independently and strengthen the communication between the classes.

Build Model

In this class, the convolutional neural network is built layer by layer. Two functions are available, one uses fixed values for each layer, and the other function implements the values from the hyperparameter optimisations. Depending on the argument value, it is determined whether the model values are part of the optimisation process or not. The fixed values are the same as shown in Table 4.2.

Inner Optimiser

After building the neural network, it must be compiled, using an inner optimisation method from this class. The exact method to use is indicated through the passed arguments. A combination of the arguments specifies whether the inner optimiser is implemented with the *learning rate* as a hyperparameter to optimise, or as a fixed value. The fixed value is 0.001, which achieved the best performance in all experiments, as highlighted in Table 4.6. The inner optimisation is then passed to the Build Model class to compile the network and prepare it for the next stages.

Tuning

The Tuning class uses the created model and the preprocessed dataset from their respective classes and applies either Random Search, Bayesian Optimisation, or Hyperband, depending on the given argument. If the optimisation option is selected and the method is provided, this class either uses the fixed model or the model that implements the optimised hyperparameters, which is indicated by the number of parameters to tune, which is passed through the arguments. The class returns the best performing hyperparameter values.

Training

The Training class only has one function which trains the model with the prepared dataset. First, it must be checked whether the training process uses the predefined hyperparameter values or whether it gets the values from the previous tuning process. While training the neural network, Early Stopping is applied to stop the training iterations before overfitting starts. Additionally, the training data is logged using TensorBoard from TensorFlow. This way, the training can be inspected further by looking at development graphs. Once the model is fully trained, it is saved and can be found using the path provided through the `-model` argument.

Evaluating

After the training cycle is completed, the testing dataset is loaded and tested on the trained model. This way, the performance of the CNN is evaluated by using image data that is new to the model. The final evaluation is saved in separate reports for documentation purposes. The resulting accuracy is the test accuracy and will most likely differ from the training accuracy, as this challenges the functionality of the

classifier. The classification report not only shows the final average accuracy but also a breakdown of the separate collision-aware contact state accuracies. As there are only three different states, the final accuracy is the average of these three separate accuracies.

Read Data

If a .h5 file is provided as an argument value for `-predictions` instead of an image directory, then the images for the final prediction process need to be generated first. Hence, the user is prompted with a request to enter the specific joint and time frame, so that the images are precise. An example user input could be the joint 17, from millisecond 21000 to 21101. This way, exactly 100 raw spectrograms are created and saved in a separate directory for further labelling.

Predicting

The Predicting class serves as the last stage of the framework. Here, the trained model and the data to label are loaded. It does not make a difference when exactly the model has been trained, as this class can work as a standalone class. By choosing the `pred` option for the argument `-train`, the images in the provided directory are trained with the model provided through the path from the argument `-model`. This way, the entire training, testing, and optimising process can be ignored and the framework can be implemented as the sole classifier to use on the humanoid robot. The predicted labels will be written onto the image and saved in the same directory.

4.4.2. Prediction Process

As already described in section 4.4.1, the prediction process can work on its own when provided with the necessary data. This way, the framework is not only a pipeline that offers all functions needed to successfully train, test, and optimise a convolution neural network for collision-aware contact classification but adding to this, it also provides the final tool to use on the humanoid robot.

In order to show the label in a clear manner that makes it easy to check its accuracy, the label is written onto the image, as depicted in Figure 4.5. The label is saved in a separate variable which is available for further uses. The example also shows that the classifier is able to distinguish between the three states even though they share similarities.

0. No Contact 1. Contact 2. Collision

Figure 4.5.: Example of predicted images.

The accuracy of the prediction process depends on the model that is passed through the `-model` argument. For this reason, the default of the framework is to just predict images. For easy access, only the model path must be passed and the predicted images are be saved in a separate directory. There is no limit as to how many images can be predicted, except for the time available. In order to make the prediction process as quick as possible, the images are resized and then passed to the model for prediction. After several iterations of stopping the time it takes for one single prediction, a pattern is identified. Table 4.7 shows the different times it takes the model to predict the label. It is important to mention that the first prediction takes the longest because it incorporates the first interaction of the dataset with the trained model. The table also underlines the fact that it makes a difference in how the model is trained in the first place and how the different accuracies impact the prediction times.

	Time in ms	
Optimisation Method	First Prediction	Following Predictions
Bayesian Optimisation	280 - 470	100 - 130
Hyperband	270 - 650	100 - 170
Random Search	380 - 410	110 - 150
No Optimisation	200 - 350	90 - 130

Table 4.7.: Comparison of prediction times.

It can be seen that the prediction process is the fastest when there has been no hyperparameter optimisation during the training process of the model. However, the prediction times after the first prediction has been made, are very close and only

differ by less than 100 ms. Nevertheless, the first prediction can take up to six times longer using, for instance, the Hyperband model. Generally, it can be said that the first prediction will take at least 200 ms no matter which model is used. Thus, on average it will take the classifier a little over 1 second to predict the labels of 10 images.

5. Evaluation

After fully training, testing, and optimising the convolutional neural network, the results are implemented into the final framework. This framework offers the tools needed to apply the collision-aware contact classifier to any given dataset from the humanoid robot. The following sections focus on comparing and analysing the attained results.

5.1. Results

The results of each stage have been discussed separately within their sections in chapter 4. However, at the end, there are five different classifiers in total that achieve their best average accuracies, respectively. The classifiers are shown in Table 5.1 with their corresponding hyperparameters.

Classifier	Accuracy	Hyperparameters
Threshold	68.7%	Grayscale 3 (pixel: [280, 400])
Base CNN	72.0%	Learning Rate, Batch Size, Epochs
Bayesian Optimisation	97.6%	Learning Rate
Hyperband	98.0%	Learning Rate, Filters, Units, Dropout
Random Search	98.1%	Learning Rate, Filters, Units, Dropout

Table 5.1.: Comparison of the best accuracies from all stages.

It is shown that the lowest accuracy is achieved by the threshold classifier, when creating a state-of-the-art classifier for collision-aware contact classification. Evidently, the goal is to show that the state-of-the-art classifier is not good enough for these applications and to propose a new approach by implementing a CNN. However, once the base CNN is created, the accuracy is already improved by 3.3%. Thus, the goal is already partially accomplished. Nonetheless, there is a lot of potential to optimise the hyperparameters of the CNN by applying specific optimisation methods. It is

recommended to always optimise a neural network before implementing it [3]. When concentrating only on tuning the *learning rate* value, Bayesian Optimisation increases the performance of the base CNN by 25.6%. As can be seen, the optimisation of the CNN through only one hyperparameter already has an impact on the quality of the classifier. Compared to the threshold classifier, the optimisation improves the recognition accuracy by 28.9%. By adding more hyperparameters to the optimisation dimensions, Hyperband achieves 98.0% accuracy and Random Search 98.1%. In essence, by creating an optimised CNN to classify collision-aware contact states, the state-of-the-art classifier's performance is exceeded by almost 30%.

5.2. Analysis and Discussion

After looking at the results from section 5.1, it is established that the collision-aware contact classifier has a significant impact on the recognition accuracy. The improvements of the accuracy happen in two phases. In the first phase, the threshold classifier is upgraded by adding a neural network that learns from example data, which increases the recognition accuracy by 3.3%. In the second phase, the CNN is optimised utilising several methods, which leads to a maximum increase of the recognition accuracy by another 26.1%. Evidently, the reason for this strong improvement is the implementation of the CNN and the connected hyperparameter optimisation. Researchers have shown that using neural networks for collision detection achieves a better performance in general [13], and adding a Fourier-based approach makes it easier to distinguish Collision from Contact [19]. Thus, by combining the CNN with data that is first modified by the Fourier Transform, the performance of the classifier is improved by 29.4%. Only this combination makes this result possible, as the state-of-the-art classifier already uses the modified data, however, it shows that the heuristic approach with the thresholds is not as successful as a CNN.

Moreover, the optimisation of the implemented CNN makes the biggest difference with increasing the performance to over 97% recognition accuracy. The results from section 4.3.3 also show that even the worst performance through an optimiser is still at 85.8% accuracy after less than three minutes of optimising the CNN. Hence, instead of carefully choosing thresholds and trying out different hyperparameter values manually with unsatisfying results, investing more time in automatically optimising the classifier's hyperparameter values proves to be more efficient in the long run [3].

Furthermore, when preparing the classifier for further usage, the accuracy depends on the number of hyperparameters that are supposed to be tuned and the optimisation

method, which tunes them. As can be seen from all conducted experiments in chapter 4, Hyperband and Random Search perform best with a wider range of hyperparameters to tune. Comparatively, Bayesian Optimisation performs as well as the other two methods if the hyperparameter space is limited to one parameter only, as for instance portrayed in Table 5.1. Adding to this, the difference between their respective accuracies lies within 0.1% and 0.4%, which is not as significant as the difference between using thresholds only and using the CNN. Nonetheless, the differing performances can be explained by looking closely at the algorithms that describe the methods. As stated in section 2.1.1, Random Search iterates through randomly chosen hyperparameter values and returns the best performing set after a specified number of trials [1]. Hyperband works in a similar manner, as it trials with different configurations until the best one is returned at the end [24]. Hence, both methods already implement values for all hyperparameters during each iteration, which is applied in this case as well. The final result of the methods is a set of values that achieves the best possible accuracy. However, Bayesian Optimisation is a sequential model-based approach, thus, it works from one hyperparameter to the next and jumps back again to take previous measurements into account [40]. This means that Bayesian Optimisation works most efficiently with a small number of hyperparameters to tune. As a result, all three methods are built into the framework, so that either optimisation method can be used for future alterations to the classifier.

Advantages

There are certain advantages that come with this specific collision-aware contact classifier. Firstly, it is structured as a fully functional framework that can be embedded within several systems. The framework offers the entire infrastructure that instantly creates a new dataset when provided with the needed torque data. This data is then applied to the training, testing, and optimisation processes of the classifier to adjust it to new environments and systems. The dataset can either be added to the already existing one, or used to create a completely new classifier, which only works on the new dataset. The instant data generation also makes it possible to detect collisions and contacts in real time and pass the predicted label on for following reactions. This way, damages to the robot and its environment can be limited. These advantages let the robot work in a coexisting environment with humans and even other robots and offer more flexibility of the tasks. Depending on the planned reactions after the successful recognition of collisions, the robot becomes more independent and is able to work more autonomously, as it learns from past experiences.

Disadvantages

A CNN is a complex structure that deals with a significant amount of bias and noise. The only information the CNN trains with, is the transformed torque data from the robot. As already mentioned before, no greater context is delivered, as the data is passed in a time window of 100 ms to ensure real time collision detection. However, there are instances when a collision is soft and slow or a desired contact is quick and strong. Within the limited time frame, their spectrograms might look similar, which makes it harder for the classifier to distinguish them. This can only be avoided by using training and testing datasets that involve different scenarios showing several kinds of collisions and contacts. There are also other difficulties that need to be acknowledged. The torque values used for the dataset originate from their respective torque sensors that are connected with a joint of the robot. Depending on the occurring task, the received torques might consist of a noisy signal. The Fourier Transform will help with filtering out the uncertainties, however, when collating the dataset, it is important to consider which joints to include. For instance, this classifier is trained with torques from the left arm joints only, as these are the most important ones, specifically for the task of wiping a panel and experiencing a collision. However, as Figure 3.4 has previously shown, the right arm torque sensors still measure values, but these torques do not show any sign of contact or collision. When using the classifier for other tasks or environments, the torques must be evaluated before deciding which values to use for the dataset. Nevertheless, the more data is available during the development phase of the classifier, the more accurate the classifier becomes.

All in all, it can be said that the framework offers a complex infrastructure for a successful and independent collision-aware contact classifier. Several methods to improve the performance are included for future adjustments. Adding to this, it has become evident that using thresholds only to detect collisions is not efficient enough. Hence, a combination of data modified through the Fourier Transform and a CNN create a novel collision classification framework.

6. Conclusion

In this thesis, a novel approach for collision detection has been introduced. This new collision-aware contact classifier is able to efficiently recognise the three different contact states. The classifier is also part of a framework that allows future implementations and alterations for changing environments and robots. In the following sections, all findings are summarised and potential extensions of this classifier are discussed.

6.1. Summary

The proposed collision-aware contact classifier has been fully trained, tested, optimised, and implemented with a final recognition accuracy of 98.1%. This has been achieved by first creating a state-of-the-art classifier, which recognises collisions through fixed thresholds. The state-of-the-art classifier achieves an average recognition accuracy of 68.7%. However, as the main objective of this thesis is to enhance the threshold classifier and improve its accuracy by implementing a CNN that reads torque data in the form of spectrograms, a base recognition accuracy of 72% has been achieved. Automating the process of optimising the hyperparameters of the classifier with three different methods, results in recognition acccuracies over 97%. Hence, all three optimisation methods are incorporated into the final framework. The framework itself offers all methods needed to train, test, and optimise the classifier in case of future alterations or extensions. Furthermore, it also provides the final classifier that instantly labels the input data and predicts their current contact state. All in all, it can be said that the recognition accuracy of the standard collision detection method using thresholds only has now been improved by 29.4%. Instead of readjusting thresholds in new environments, the framework is implemented to retrain the classifier and optimise it automatically to achieve optimal results. This way, standard collision detection approaches that use thresholds [46] or Fourier Transform only [20] [19] have been combined and enhanced by adding a CNN [13] and supporting the learning process of the humanoid robot Rollin' Justin.

6.2. Outlook

As efficient collision detection adds to the autonomy of a humanoid robot, there is a lot of potential for many future extensions of this collision-aware contact classifier. First of all, this classifier's training and testing dataset consists of over 4,000 spectrograms from only six sample log files. However, this amount can be increased by adding more historic torque data from the robot's log entries. More available data leads to better recognition accuracies. If enough data is available, then torques from other actions and other joints can be added to the dataset, as this classifier only focuses on the wiping action of the robot and the measured torques from the left arm. Moreover, the classifier can also be trained to predict an additional contact state, for instance when a collision is happening at the same time as an intended contact. However, in order for this to be successful, enough experiments must be conducted to collate a new dataset, specifically for this new contact state. Comparatively, the prediction time of an occurring contact state can be improved and optimised in future experiments. The classifier has been trained with 100 ms windows that need 100 ms on average for prediction. Hence, when applying the classifier online on the robot, there is a delay of 200 ms, which can be minimised by additional enhancements. Ericson suggested that collision detection is more efficient if there are specialised classifiers for specific scenarios and joints instead of one all-encompassing classifier [9]. Putting this into a context, if enough data is provided, the framework can be used to create specific classifiers that work best when focusing on one joint only. Applying this to the wiping action scenario, it would mean that each joint of the left arm of the robot has its own optimised collision-aware contact classifier. This way, better accuracies could be achieved.

In conclusion, all the aforementioned extensions of the current collision-aware contact classifier will benefit the autonomy and flexibility of the humanoid robot. It can be seen that the framework offers all the tools needed to perform those enhancements and enable the robot to better adjust to unfamiliar environments and handle collisions in a manner that limits potential damages. This way, scientists will be one step closer to humanising robots for good and preparing them to explore even more unknown and unpredictable places.

Bibliography

[1] J. Bergstra and Y. Bengio. "Random search for hyper-parameter optimization." In: *Journal of Machine Learning Research* 13 (2012), pp. 281–305. URL: https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.

[2] Chang-Nho Cho et al. "Collision Detection Algorithm to Distinguish Between Intended Contact and Unexpected Collision." In: *Advanced Robotics* 26.16 (2012), pp. 1825–1840. DOI: 10.1080/01691864.2012.685259. eprint: https://doi.org/10.1080/01691864.2012.685259. URL: https://doi.org/10.1080/01691864.2012.685259.

[3] Hannah M. Claus et al. "Facial Recognition Application with Hyperparameter Optimisation." In: *Big Data Intelligence for Smart Applications*. Ed. by Youssef Baddi et al. Springer International Publishing, 2022, pp. 141–172. ISBN: 978-3-030-87954-9. DOI: 10.1007/978-3-030-87954-9_6. URL: https://doi.org/10.1007/978-3-030-87954-9_6.

[4] W.T. Cochran et al. "What is the fast Fourier transform?" In: *Proceedings of the IEEE* 55.10 (1967), pp. 1664–1674. DOI: 10.1109/PROC.1967.5957.

[5] B. Csaji and H. Ten Eikelder. "Approximation with Artificial Neural Networks." 2001. URL: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.101.2647&rep=rep1&type=pdf.

[6] Alessandro De Luca et al. "Collision Detection and Safe Reaction with the DLR-III Lightweight Manipulator Arm." In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2006, pp. 1623–1630. DOI: 10.1109/IROS.2006.282053.

[7] T. Dietterich. "Overfitting and undercomputing in machine learning." In: *ACM Computing Surveys* 27 (1995), pp. 326–327. URL: https://dl.acm.org/doi/abs/10.1145/212094.212114.

[8] Ana M. Djuric, R.J. Urbanic, and J.L. Rickli. "A Framework for Collaborative Robot (CoBot) Integration in Advanced Manufacturing Systems." In: *SAE International Journal of Materials and Manufacturing* 9.2 (2016), pp. 457–464. ISSN: 19463979, 19463987. URL: http://www.jstor.org/stable/26267460 (visited on 05/10/2022).

[9] C. Ericson. *Real-Time Collision Detection*. Morgan Kaufmann series in interactive 3D technology. CRC Press, 2004, pp. 12–13. ISBN: 9780080474144. URL: https://books.google.de/books?id=4wTNBQAAQBAJ.

[10] K. Fukushima. "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position." In: *Biological Cybernetics* 36 (1980), pp. 193–202. URL: https://www.cs.princeton.edu/courses/archive/spr08/cos598B/Readings/Fukushima1980.pdf.

[11] Stefan Grushko et al. "Improved Mutual Understanding for Human-Robot Collaboration: Combining Human-Aware Motion Planning with Haptic Feedback Devices for Communicating Planned Trajectory." In: *Sensors* 21.11 (2021). ISSN: 1424-8220. DOI: 10.3390/s21113673. URL: https://www.mdpi.com/1424-8220/21/11/3673.

[12] Sami Haddadin, Alessandro De Luca, and Alin Albu-Schäffer. "Robot Collisions: A Survey on Detection, Isolation, and Identification." In: *IEEE Transactions on Robotics* 33.6 (2017), pp. 1292–1312. DOI: 10.1109/TRO.2017.2723903.

[13] Young Jin Heo et al. "Collision Detection for Industrial Collaborative Robots: A Deep Learning Approach." In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 740–746. DOI: 10.1109/LRA.2019.2893400.

[14] Seyed Hosseini and Khaled Goher. "Personal care robots for older adults: An overview." In: (2017).

[15] D.H. Hubel and T.N Wiesel. *Brain and Visual Perception: The Story of a 25-Year Collaboration*. Oxford University Press, 2004. ISBN: 9780198039167. URL: https://books.google.de/books?id=8YrxWojxUA4C.

[16] H. Jabbar and R. Khan. "Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study)." In: *Computer Science, Communication and Instrumentation Devices* (2014). URL: https://www.researchgate.net/publication/295198699.

[17] Kevin Jamieson and Ameet Talwalkar. "Non-stochastic best arm identification and hyperparameter optimization." In: *Artificial intelligence and statistics*. PMLR. 2016, pp. 240–248.

[18] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. DOI: 10.48550/ARXIV.1412.6980. URL: https://arxiv.org/abs/1412.6980.

[19] Alexandros Kouris, Fotios Dimeas, and Nikos Aspragathos. "A Frequency Domain Approach for Contact Type Distinction in Human-Robot Collaboration." In: *IEEE Robotics and Automation Letters* PP (Jan. 2018), pp. 1–1. DOI: 10.1109/LRA.2017.2789249.

[20] Alexandros Kouris, Fotios Dimeas, and Nikos Aspragathos. "Contact Distinction in Human-Robot Cooperation with Admittance Control." In: Oct. 2016. DOI: 10.1109/SMC.2016.7844525.

[21] Thomas Kurbiel and Shahrzad Khaleghian. *Training of Deep Neural Networks based on Distance Measures using RMSProp*. 2017. DOI: 10.48550/ARXIV.1708.01911. URL: https://arxiv.org/abs/1708.01911.

[22] Y. LeCun et al. "Backpropagation Applied to Handwritten Zip Code Recognition." In: *Neural Computation* 1 (1989), pp. 541–551. URL: http://yann.lecun.com/exdb/publis/pdf/lecun-89e.pdf.

[23] Daniel Sebastian Leidner. "Cognitive reasoning for compliant robot manipulation." PhD thesis. Springer, 2017.

[24] Lisha Li et al. "Hyperband: A novel bandit-based approach to hyperparameter optimization." In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 6765–6816.

[25] S.-W. Lin et al. "Particle swarm optimization for parameter determination and feature selection of support vector machines." In: *Expert Systems with Applications* 35 (2008), pp. 1817–1824. URL: https://www.sciencedirect.com/science/article/abs/pii/S0957417407003752.

[26] Stephen T. Mahon et al. "Soft Robots for Extreme Environments: Removing Electronic Control." In: *2019 2nd IEEE International Conference on Soft Robotics (RoboSoft)*. 2019, pp. 782–787. DOI: 10.1109/ROBOSOFT.2019.8722755.

[27] Ivan Margolius. *The Robot of Prague*. 2017. URL: https://czechfriends.net/images/RobotsMargoliusJul2017.pdf.

[28] C. Mouton, J. C. Myburgh, and M. H. Davel. "Stride and Translation Invariance in CNNs." In: *Artificial Intelligence Research*. Ed. by A. Gerber. Cham: Springer International Publishing, 2020, pp. 267–281. ISBN: 978-3-030-66151-9.

[29] NumPy. *Numpy*. 2022. URL: https://numpy.org.

[30] Tom O'Malley et al. *BayesianOptimization Tuner*. 2019. URL: https://keras.io/api/keras_tuner/tuners/bayesian/#bayesianoptimization-class.

[31] Tom O'Malley et al. *Hyperband Tuner*. 2019. URL: https://keras.io/api/keras_tuner/tuners/hyperband/.

[32] Tom O'Malley et al. *KerasTuner*. 2019. URL: https://github.com/keras-team/keras-tuner.

[33] Tom O'Malley et al. *RandomSearch Tuner*. 2019. URL: https://keras.io/api/keras_tuner/tuners/random/#randomsearch-class.

[34] OpenCV.AI. *OpenCV.AI*. 2022. URL: https://www.opencv.ai.

[35] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[36] Stephen M. Pizer et al. "Adaptive histogram equalization and its variations." In: *Computer Vision, Graphics, and Image Processing* 39.3 (1987), pp. 355–368. ISSN: 0734-189X. DOI: https://doi.org/10.1016/S0734-189X(87)80186-X. URL: https://www.sciencedirect.com/science/article/pii/S0734189X8780186X.

[37] V. Romanuke. "Appropriate Number and Allocation of ReLUs in Convolutional Neural Networks." In: 1 (2017).

[38] D. Scherer, A. Mueller, and S. Behnke. "Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition." In: 2010.

[39] N. Scrivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." In: *Journal of Machine Learning Research* 15 (2014). URL: http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf.

[40] Bobak Shahriari et al. "Taking the Human Out of the Loop: A Review of Bayesian Optimization." In: *Proceedings of the IEEE* 104.1 (2016), pp. 148–175. DOI: 10.1109/JPROC.2015.2494218.

[41] Y. Shen et al. "Learning Semantic Representations Using Convolutional Neural Networks for Web Search." In: WWW 2014, Apr. 2014. URL: https://www.microsoft.com/en-us/research/publication/learning-semantic-representations-using-convolutional-neural-networks-for-web-search/.

[42] Patrice Y Simard, David Steinkraus, John C Platt, et al. "Best practices for convolutional neural networks applied to visual document analysis." In: *Icdar*. Vol. 3. 2003. 2003.

[43] Julius O. Smith. *Mathematics of the Discrete Fourier Transform (DFT)*. W3K Publishing, 2007. ISBN: 978-0-9745607-4-8. URL: http://www.w3k.org/books/.

[44] Vahid Sotoudehnejad et al. "Counteracting modeling errors for sensitive observer-based manipulator collision detection." In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 4315–4320. DOI: 10.1109/IROS.2012.6386198.

[45] Ilya Sutskever et al. "On the importance of initialization and momentum in deep learning." In: *International conference on machine learning*. PMLR. 2013, pp. 1139–1147.

[46] S. Takakura, T. Murakami, and K. Ohnishi. "An approach to collision detection and recovery motion in industrial robot." In: *15th Annual Conference of IEEE Industrial Electronics Society*. 1989, 421–426 vol.2. DOI: 10.1109/IECON.1989.69669.

[47] The Matplotlib Development Team. *Matplotlib: Visualization with Python*. 2022. URL: https://matplotlib.org.

[48] TensorFlow. *Conv2D*. 2022. URL: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D.

[49] TensorFlow. *Dropout*. 2022. URL: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dropout.

[50] TensorFlow. *Early Stopping*. 2022. URL: https://tensorflow.google.cn/api_docs/python/tf/keras/callbacks/EarlyStopping.

[51] TensorFlow. *Flatten*. 2022. URL: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam.

[52] TensorFlow. *Image Data Generator*. 2022. URL: https://tensorflow.google.cn/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator.

[53] TensorFlow. *MaxPool2D*. 2022. URL: https://www.tensorflow.org/api_docs/python/tf/keras/layers/MaxPool2D.

[54] TensorFlow. *Softmax*. 2022. URL: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Softmax.

[55] TensorFlow. *TensorBoard*. 2022. URL: https://www.tensorflow.org/tensorboard/get_started.

[56] TensorFlow. *TensorFlow*. 2022. URL: https://www.tensorflow.org.

[57] TensorFlow. *The Sequential Model*. 2022. URL: https://www.tensorflow.org/guide/keras/sequential_model.

[58] John Ulmen and Mark Cutkosky. "A robust, low-cost and low-noise artificial skin for human-friendly robots." In: *2010 IEEE International Conference on Robotics and Automation*. 2010, pp. 4836–4841. DOI: 10.1109/ROBOT.2010.5509295.

[59] M.V. Valueva et al. "Application of the residue number system to reduce hardware costs of the convolutional neural network implementation." In: *Mathematics and Computers in Simulation* 177 (2020), pp. 232–243. ISSN: 0378-4754. DOI: https://doi.org/10.1016/j.matcom.2020.04.031. URL: https://www.sciencedirect.com/science/article/pii/S0378475420301580.

[60] Jonathan Vorndamme, Moritz Schappler, and Sami Haddadin. "Collision detection, isolation and identification for humanoids." In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 4754–4761. DOI: 10.1109/ICRA.2017.7989552.

[61] S. van der Walt et al. "scikit-image: image processing in Python." In: *PeerJ* 2 (June 2014), e453. ISSN: 2167-8359. DOI: 10.7717/peerj.453. URL: https://doi.org/10.7717/peerj.453.

[62] S.-C. Wang. "Artificial Neural Network." In: *Interdisciplinary Computing in Java Programming. The Springer International Series in Engineering and Computer Science* 743 (2003), pp. 81–100. URL: https://link.springer.com/chapter/10.1007/978-1-4615-0377-4_5.

[63] Python Package Wiki. *Imutils*. 2022. URL: https://package.wiki/imutils.

[64] B. Zhao et al. "Convolutional neural networks for time series classification." In: *Journal of Systems Engineering and Electronics* 28.1 (2017), pp. 162–169. DOI: 10.21629/JSEE.2017.01.18.

A. Appendix

Hannah M. Claus

Automated Optimisation of Collision-aware Contact Classification in Robotics

Introduction

- Robots have become an essential part of our everyday lives.
- Humanoid robots are designed to take over simple human tasks to collaborate with humans, or assist in extreme environments [1].
- To improve their autonomy, these robots have to be able to discriminate unintended collisions from intended contact.

Solution:

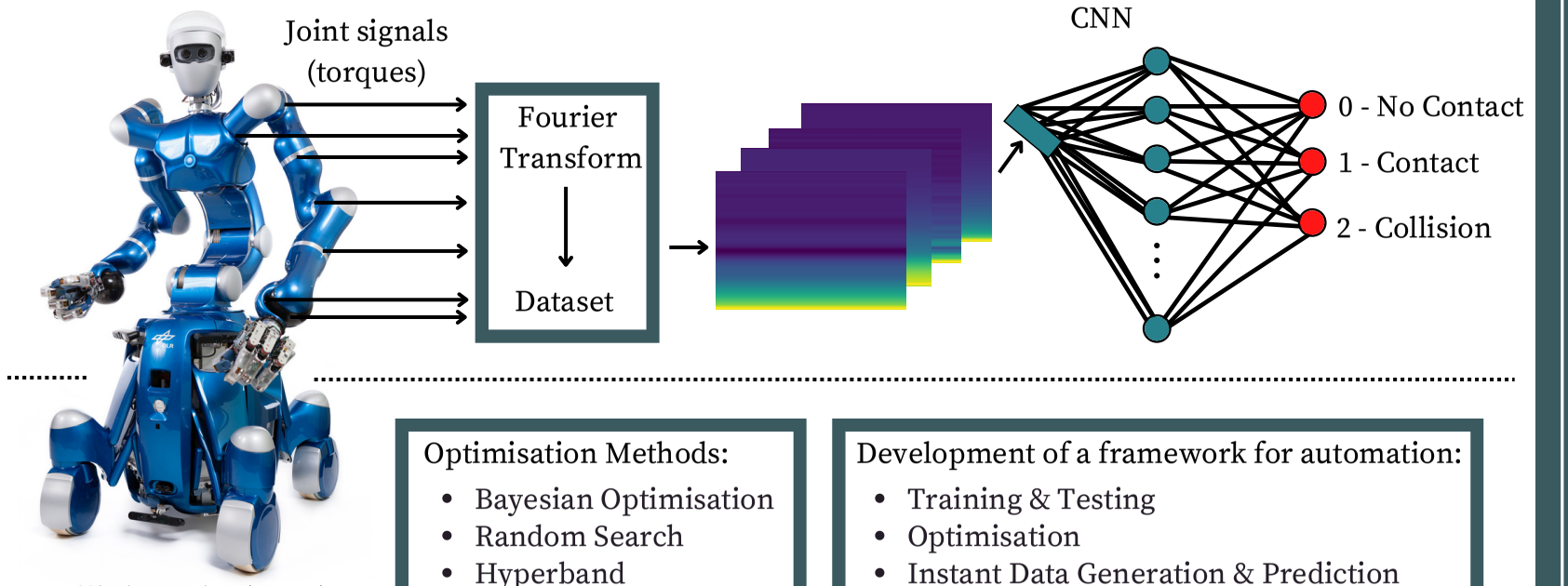
➤ **Collision-aware Contact Classifier**

Related Work

- The state-of-the-art collision detection on humanoid robots suggests setting up specific constant thresholds.
- Once a measured torque signal from the robot surpasses the threshold, it can be classified as a collision [2].
- However, this approach is tedious and not prone to changes.
- Creating a **state-of-the-art collision classifier** using grayscale thresholds, only a maximum of **68.7% recognition accuracy** is achieved.

➤ Hence, by implementing a convolutional neural network (CNN), that reads torques from the robot and applies Fourier Transform (FT), a new classifier is introduced.

Methodology



Optimisation Methods:
- Bayesian Optimisation
- Random Search
- Hyperband

Development of a framework for automation:
- Training & Testing
- Optimisation
- Instant Data Generation & Prediction

Results

The best average recognition accuracies achieved by the classifier

68.7% — State-of-the-art classifier

Manual Optimisation of the CNN (Base Accuracy) — 72%

Automated Optimisation of the collision-aware contact classifier:

97.6% — Bayesian Optimisation

Hyperband Optimisation — 98%

98.1% — Random Search Optimisation

Author

Hannah M. Claus

2022

Bachelor's Thesis, AI & Robotics
University of Bedfordshire, UK
hannahmclaus@gmail.com

Supervisors:
Vitaly Schetinin, Ph.D., University of Bedfordshire
Daniel Leidner, Ph.D., German Aerospace Center (DLR)

Conclusion

1. By implementing a CNN and FT, the collision-aware contact classifier's recognition accuracy has been improved significantly.
2. Optimisation is important in order to achieve better results that lead to safer work environments.
3. The framework offers a complex infrastructure that allows future extensions of the classifier.

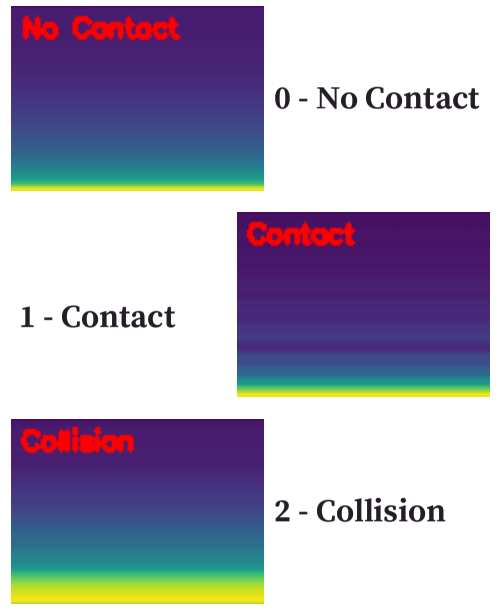
Future Work

- Add a new contact state: **Contact + Collision**
- Increase the dataset for the training and testing procedures
- Decrease the prediction time to let the robot react more quickly to collisions
- Create collision-aware contact classifiers for each joint and each action of the robot to be more precise [3]

Analysis

- The recognition accuracy has been **improved** by **+29.4%**
- The **3** contact states: **No Contact, Contact, Collision** can be successfully classified using a CNN and spectrograms (images from the FT)
- The framework automates the **5** most important processes
- Through the framework, the classifier can be **used on any robot and any joint**

Predictions



References:

[1] Djuric, A.M., Urbanic, R.J. and Rickli, J.L. (2016). A Framework for Collaborative Robot (CoBot) Integration in Advanced Manufacturing Systems. SAE International Journal of Materials and Manufacturing, [online] 9(2), pp.457–464.

[2] S. Takakura, T. Murakami and K. Ohnishi, "An approach to collision detection and recovery motion in industrial robot," 15th Annual Conference of IEEE Industrial Electronics Society, 1989, pp. 421-426 vol.2, doi: 10.1109/IECON.1989.69669.

[3] Ericson, C. (2004). Real-Time Collision Detection. [online] Google Books. CRC Press.

FACULTY OF CREATIVE ARTS, TECHNOLOGIES AND SCIENCE

Form for Research Ethics Projects (Ethics Form)

Student Name	Hannah-Melkemaryam Claus
Student Number	1900146
Degree Pathway	BSc Artificial Intelligence and Robotics
Supervisor name	Dr Vitaly Schetinin
Supervisor Signature	
Title of project	Automated Optimisation of Collision-aware Contact Classification in Robotics

SECTION B Check List

Please answer the following questions by circling **YES** or **NO** as appropriate.

Does the study involve vulnerable participants or those unable to give informed consent (e.g. children, people with learning disabilities, your own students)?	**YES** **NO x**
Will the study require permission of a gatekeeper for access to participants (e.g. schools, self-help groups, residential homes)?	**YES** **NO x**
Will it be necessary for participants to be involved without consent (e.g. covert observation in non-public places)?	**YES** **NO x**
Will the study involve sensitive topics (e.g. obtaining information about sexual activity, substance abuse)?	**YES** **NO x**
Will blood, tissue samples or any other substances be taken from participants?	**YES** **NO x**
Will the research involve intrusive interventions (e.g. the administration of drugs, hypnosis, physical exercise)?	**YES** **NO x**
Will financial or other inducements be offered to participants (except reasonable expenses or small tokens of appreciation)?	**YES** **NO x**
Will the research investigate any aspect of illegal activity (e.g. drugs, crime, underage alcohol consumption or sexual activity)?	**YES** **NO x**
Will participants be stressed beyond what is considered normal for them?	**YES** **NO x**
Will the study involve participants from the NHS (patients or staff) or will data be obtained from NHS premises?	**YES** **NO x**

If the answer to any of the questions above is "Yes", or if there are any other significant ethical issues, then further ethical consideration is required. Please document carefully how these issues will be addressed.

Signed (student):
Date: 2nd November 2021

Countersigned (Supervisor):
Date:

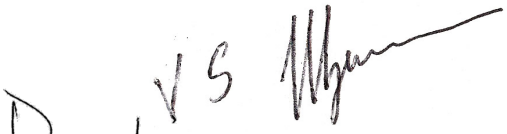
BSc Project Screening Form: Guidelines

Part 1 – Project Proposal

Student Name	Hannah-Melkemaryam Claus
Student Number	1900146
Degree Pathway	BSc Artificial Intelligence and Robotics
Supervisor Name	Dr Vitaly Schetinin
Course Coordinator name	Dr Dayou Li
Title of Project	Automated Optimisation of Collision-aware Contact Classification in Robotics
Abstract of the project	Contact classification is of great importance when it comes to robotics and aeronautics. Robotic manipulators should be able to quickly register contact and act accordingly in order to prevent or limit damages through collisions. Whether it is a desired contact or unintended collision, the robot has to determine its nature and prepare to come to a sudden halt. For this determination, a contact classifier is needed. For this thesis, a Fourier-based contact classifier will be implemented. Moreover, the pipeline for collision-aware contact classification will be optimised in its labeling and training process. Usually, these processes are time-consuming, hence, automation techniques will be proposed and integrated into the learning pipeline of the humanoid robot Rollin' Justin from the German Aerospace Center (DLR). The aim is to improve the ease of use of such classifiers in future applications.
Project deliverables	▪ Fourier-based contact classification algorithm (FBCC) ▪ Optimisation algorithm for collision-aware contact classification (CACC) ▪ Automation pipeline for labeling, training, and optimisation
Description of your artefact	Context: ▪ Contact classification has many great applications and its importance will increase over time due to the growing use of robotic manipulators in industry and science. In industry, contact classification focuses on the interaction with the industrial environment, including human-robot interaction. However, in scientific environments, such as space exploration, that is not enough because robots might encounter unfamiliar circumstances. Hence, by using specific classifiers, robotic systems can be trained and tested on their reaction to collisions in order for damages to be limited. However, the usage of such classifiers needs to be simplified and accelerated to make them more accessible to non-expert users. This can be achieved by automating certain stages of the development phase. Aim & objectives of the project: ▪ Improve the ease of use of contact classifiers in future applications through developing a pipeline for automation processes List of features: ▪ FBCC ▪ CACC ▪ Automation pipeline (AP) Added value: ▪ Better results when using classifiers ▪ Better understanding of the different development stages ▪ Automated integration of new experience data into the learning process Intellectual challenges: ▪ Understanding the principles of contact classification in applied robotics ▪ Conceptualise and implement a complex contact classification algorithm ▪ Improve performance of the classification algorithms through AI-based optimisation techniques ▪ Investigation of cutting-edge continuous integration and deployment for process automation

What methodology (structured process) will you be following to realise your artefact?	Approach: ▪ Review state of the art ▪ Implement FBCC ▪ Optimisation of classifier parameters ▪ Automate labeling and training process of the classifiers ▪ Develop a pipeline for use of these automations Justification: ▪ In order to use classifiers in applications, they need to undergo certain training and testing stages to ensure safety and quality ▪ However, these stages are time-consuming and therefore, need automated techniques, so that more time can be spent on the actual applications of said classifiers ▪ The pipeline will be created, so that in future arbitrary classifiers (e.g. for computer vision) can undergo the same automated processes as the developed contact classifiers	
How does your project relate to your degree course and build upon the units/knowledge you have studied/acquired	▪ Pipeline and algorithms are written in Python using object-oriented programming taught in Year 1 ▪ Both automation and optimisation processes utilise classifiers using machine learning algorithms which were taught in Year 2	
Resources	▪ Programming: Python, GTK+ ▪ Robotics: log data from the humanoid robot Rollin' Justin (DLR)	
Have you completed & submitted your ethics form?	YES x	NO
If the project is a development of previous work by yourself or others, give details below. Failing to declare such previous work here may be treated as an academic offence		

Student Signature: 

Supervisor Signature: 

Course Coordinator Signature: 

<u>Part 2 – List of relevant resources</u>

1. *Journal Papers*
   a. *A. Kouris, F. Dimeas and N. Aspragathos, "Contact Distinction in Human-Robot Cooperation with Admittance Control," 2016, doi: 10.1109/SMC.2016.7844525.*
   b. *A. Kouris, F. Dimeas and N. Aspragathos, "A Frequency Domain Approach for Contact Type Distinction in Human-Robot Collaboration," 2018. IEEE Robotics and Automation Letters, doi: 10.1109/LRA.2017.2789249.*
   c. *F. Franzel, T. Eiband and D. Lee, "Detection of Collaboration and Collision Events during Contact Task Execution," 2020 IEEE-RAS 20th International Conference on Humanoid Robots (Humanoids), 2021, pp. 376-383, doi: 10.1109/HUMANOIDS47582.2021.9555677.*
   d. *M. Lippi and A. Marino, "Enabling physical human-robot collaboration through contact classification and reaction," 2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN), 2020, pp. 1196-1203, doi: 10.1109/RO-MAN47096.2020.9223580.*

Collision-aware Contact Classification

This framework provides the full pipeline for a collision-aware contact classifier that reads torque data as spectrograms created using the Fourier Transform, and returns an evaluation of whether a collision or a contact is occuring, or whether there is no intersection at all.

Overview

In the following, the structure of the framework is described in details and a manual for easy implementation is provided.

Table of Contents:

Preparations

This section describes all necessary steps to create the correct working environment that ensures the functionality of this framework.

It is recommended to first create a virtual environment, so that already existing systems or data on a device are not disturbed. The following steps need to be taken in order to estblish the right environment for the classifier. It is expected that the latest version of Python 3.8 is already installed, together with pip.

1. Install Virtualenv:

```
$ pip3 install virtualenv virtualenvwrapper
```

2. Edit `~/.bashrc` profile

```
$ vim ~/.bashrc
```

and then:

```
# virtualenv and virtualenvwrapper
export WORKON_HOME=$HOME/.local/bin/.virtualenvs
export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3
export
VIRTUALENVWRAPPER_VIRTUALENV=$HOME/Library/Python/3.8/bin/virtualenv
source $HOME/Library/Python/3.8/bin/virtualenvwrapper.sh
```

3. Source venv

```
$ source ~/.bashrc
```

New terminal commands:

- Create an environment with `mkvirtualenv`
- Activate an environment (or switch to a different one) with `workon`
- Deactivate an environment with `deactivate`
- Remove an environment with `rmvirtualenv`

More information can be found in the docs.

4. Create new venv:

```
$ mkvirtualenv classifier -p python3
```

5. Install packages:

```
$ workon classifier
$ pip install opencv-contrib-python
$ pip install numpy
$ pip install scikit-learn
$ pip install scikit-image
$ pip install imutils
$ pip install matplotlib
$ pip install tensorflow # or tensorflow-gpu
```

Note:

A virtual environment is not necessary in order to run the framework, however, it is recommended in order to protect the data and devices.

If working without a virtual environment, please make sure that all necessary libraries are installed. If not, install them as follows:

```
$ pip3 install opencv-contrib-python
$ pip3 install numpy
$ pip3 install scikit-learn
$ pip3 install scikit-image
$ pip3 install imutils
$ pip3 install matplotlib
$ pip3 install tensorflow # or tensorflow-gpu
```

It is essential that all these packages are installed before running the classifier, so that no errors occur.

The Dataset

The directory `data` contains the entire dataset that is used to train and test the collision-aware contact classifier. It is divided into:

- `Meta`: This stores a clean image of each contact class that corresponds with the `Meta.csv` file.
- `Train`: Here, the images are grouped by their contact class and given a separate directory. Each class shows 1,134 images of one single contact state from different joints and logs, which are all annotated in `Train.csv`.
- `Test`: This directory consists of 690 images (230 per class) to test and evaluate the accuracy of the model. Some of these images include contact states that

might be easily misclassified because of their resemblance with other contact states to see if the network is trained well enough. The images are annotated in `Test.csv`

- `Predict`: This is an example directory for applying the classifier to new and unknown data to predict the correct contact label, which is connected to the `Predict.csv` file.

The structure of the dataset is simple to support further extensions. The file `labels.csv` includes the information about the three possible contact classes:

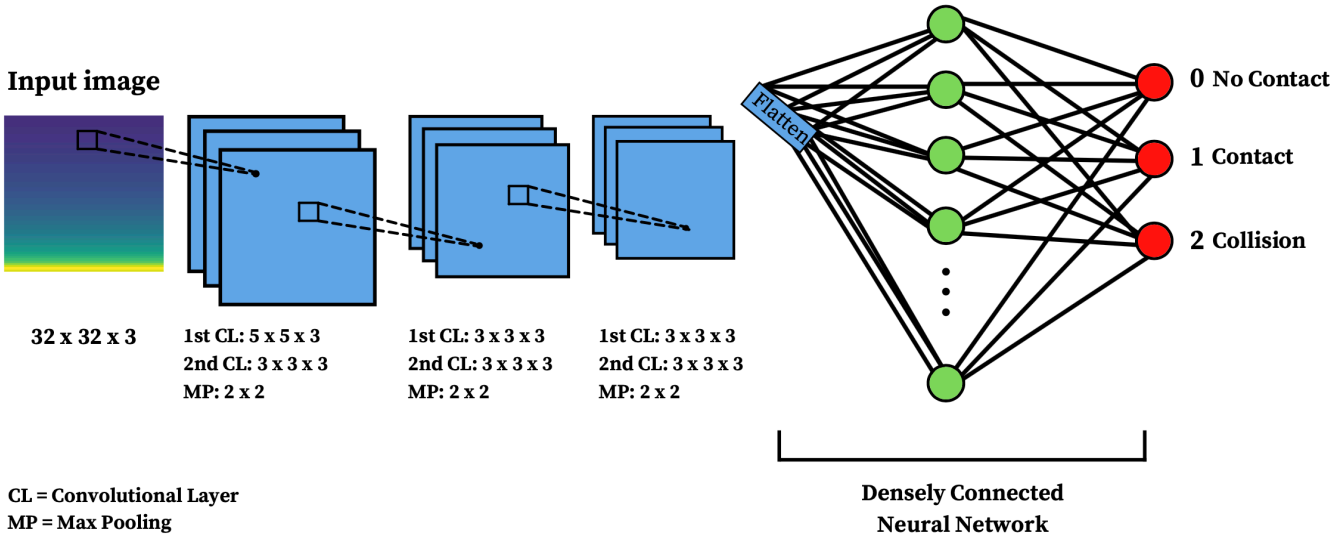
- 0: No Contact
- 1: Contact
- 2: Collision

Each image is only labelled as one of these contact states.

The Framework

The directory `framework` will offer the neccessary functions and methods needed in order to build, train, test, and optimise the convolutional neural network (CNN). The functions can also be used to instantly generate new data to predict labels.

The architecture of the CNN is shown in the image below:



The framework can be used by specifying the intentions through the arguments in the Terminal. The files are created in an object-oriented structure to establish a clear infrastructure. The files work as follows:

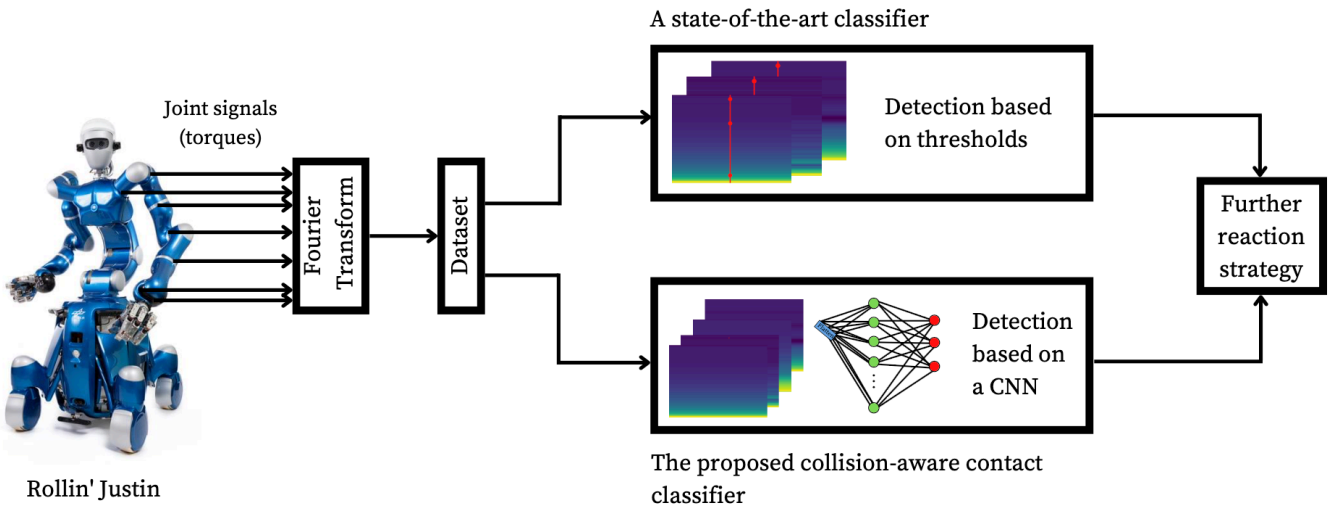
- `arguments.py`: specify the arguments that are passed to `main.py`,

- `build_model.py` : build the CNN layer by layer,
- `evaluating.py` : evaluate and test the trained classifier,
- `helper` : provide helping functions that can be used by each class,
- `inner_opt.py` : establish the inner optimisation method,
- `main.py` : run the framework as commanded through the Terminal,
- `predicting.py` : predict the labels of new images,
- `preprocessing.py` : process the images and prepare them for the training, testing, or prediction processes,
- `read_data.py` : instantly generate new data,
- `training.py` : train the classifier,
- `tuning.py` : optimise the classifier with one of the three optimisation methods.

The State-of-the-art Classifier

As further described in the thesis, a state-of-the-art classifier is created at the beginning to later compare it with the performance of the novel classifier.

The image below shows the functionality of both classifiers:



The threshold classifier is implemented by using grayscale values. The code is saved in the `threshold_classifier` directory, together with a summary of the attained results.

This classifier can be run by adjusting the image paths in the code and typing

```
$ python3 grayscale.py
```

into the Terminal.

Tensorboard Logs

The folder `tensorboard_logs` entails all logs collected through the different training, testing, and optimisation stages of the classifier. They can be used to analyse and evaluate the performance in TensorBoard. Further instructions can be found on Tensorboard's website.

Optimisation Documentation

During the automatic optimisation of the collision-aware contact classifier, the three used methods (Bayesian Optimisation, Random Search, and Hyperband) documented their trials in the directory `optimisation_trials`.

Output Models

The `output` folder consists of the saved models. As there are four different models (one without optimisation, and three with optimisation) it is necessary to separate them. With the final accuracy of over 98%, the random search model can be implemented immediately.

This is the path that needs to be passed through the main command, so that the classifier knows which model to use for the prediction of new images.

Generating Data

Using the torque data from humanoid robot, a specific .h5 file can be added to the Terminal command, which will automatically create new data to predict labels.

Prediction

The predicted images will be loaded into the folder `predictions` or, if they are attained from other directories labeled as `Predict`, will be saved in the respective directories.

Run the Framework

In order to run the framework successfully, several steps need to be included.

1. Start Virtual Environment (if necessary)

```
$ source ~/.bashrc
$ workon classifier
```

2. Change directories

```
$ cd collision_aware_cc/framework
```

3. Specify arguments:

There are 6 arguments that need to be added when running the code. The `class Args()` can be seen in `arguments.py`.

The main structure of a command is as follows:

```
$ python3 main.py -m -op -inop -d -pr -tr
```

where all arguments are optional. The arguments are described below:

- `-m` or `--model` to add the path to the output model,
- `-op` or `--optimiser` to add the optimisation methods to use, can be either `bayesian`, `random`, or `hyperband`,
- `-inop` or `--inner_optimiser` to add the inner optimisation method, can be either `adam`, `sgd`, or `rms`,
- `-d` or `--dataset` to add the path to the input dataset for training and testing the classifier,
- `-pr` or `--predictions` to add the path to the output prediction directory or add a `.h5` file from the robot's log files for instant data generation,
- `-tr` or `--train` to indicate the number of hyperparameters to optimise during the training process, or to skip the training entirely to just predict labels, can be either `all`, `one`, `none`, or `pred`.

4. Run command in the Terminal

The framework can be run through a command in the terminal. The `main.py` file will only run with the correct arguments.

The following command will train and test the neural network with the assigned dataset without any optimisation:

```
$ python3 main.py --model ../output/neural_net.model --dataset ../data
--predictions ../predictions --train none
```

When optimising the neural network, add the `optimiser` argument as shown below.

5. Run optimisations

Bayesian Optimisation:

This command will train, test, and optimise the CNN by tuning all hyperparameters using Bayesian Optimisation and the inner optimiser Adam.

```
$ python3 main.py --model ../output/bayesian.model --optimiser bayesian
--inner_optimiser adam --dataset ../data --predictions ../predictions -
-train all
```

If only the `learning rate` value should be tuned with Bayesian Optimisation, the argument can be changed as follows:

```
$ python3 main.py --model ../output/bayesian.model --optimiser bayesian
--inner_optimiser adam --dataset ../data --predictions ../predictions -
-train one
```

Hyperband Optimisation:

This command will train, test, and optimise the CNN by tuning all hyperparameters using Hyperband Optimisation and the inner optimiser Adam.

```
$ python3 main.py --model ../output/hyperband.model --optimiser
hyperband --inner_optimiser adam --dataset ../data --predictions
../predictions --train all
```

Random Search:

This command will train, test, and optimise the CNN by tuning all hyperparameters using Random Search and the inner optimiser Adam.

```
$ python3 main.py --model ../output/random_search.model --optimiser
random --inner_optimiser adam --dataset ../data --predictions
../predictions --train all
```

Make sure to always provide the correct model path in combination with the correct optimiser.

If a different inner optimiser is wanted, the argument can be changed to:

```
$ python3 main.py --model ../output/random_search.model --optimiser
random --inner_optimiser sgd --dataset ../data --predictions
../predictions --train all
```

6. Run predictions only

If there is already a fully trained and optimised model that is saved on a specific path, the trianing process can be skipped, by directly using the collision-aware contact classifier to predict labels of provided images.

As the classifier is the main product of the framework, this is the default setting. Hence, by writing

```
$ python3 main.py
```

into the Terminal, it will use the default model, in this case the CNN optimised with Hyperband, which achieves 98.0% accuracy. The framework already provides a directory with sample images that can be used as a new dataset for predictions.

This will save the predicted images in the separate `Predict` folder.

However, if a specific model is desired to be used on a specific dataset, it can be run as follows:

```
python3 main.py -m ../output/hyperband.model -pr Predict -tr pred
```

By passing a file that ends with `.h5` instead of an image directory, the framework

knows that it needs to generate new data first, which will then be used to predict labels. The following command shows an example:

```
python3 main.py -pr ../h5files/collision/log_20200916_130141.h5
```

Results

The output in the shell will indicate the current state of the programme.

A classification report will appear with the prediction accuracy, if the training option is chosen.

The predicted images can then be viewed in the assigned folder.

The results may look like this:

0. No Contact 1. Contact 2. Collision