# IAS0360 Final project report: Up-down moving garage door cycle detection model

Lilian Hunout, Ondřej Schejbal

Department of Computer Systems
Tallinn University of Technology
Tallinn, Estonia

January 2022


Figure 1: Orientation of the axes of the garage door

# Contents



# List of Figures

# 1 Introduction

This final project consists of detecting the position of a garage door. This detection was done by preparing a system for recognition of the different states and detect cycles the door has gone though in observed data.

First we needed to analyze the data to fully explore and understand what we are working with. We have explored what features have the most significant impact for door state prediction and which features can be omitted.

Then we had to come up with a suitable neural network model. We have decided to divide the task into two parts. One part was door state detection which we solved by preparing simple neural network model and the second part was to process output of our model and detect the number of cycles that occurs in the observed data sequence.

At the end we converted our model into C code representation and prepared working system which runs on STM32 embedded device. We also needed to create simple UI to visualize the outputs.

# 2 State of the art

State of the art describes the most recent techniques of this particular technology. It is the best technology available because it has been developed using the most modern techniques and technologies.

To have all the necessary resources during our project, we will work with the help of recognized sources. For current state of the art for our task we have used book *"TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers, Pete Warden, D. Situnayake"* and the article *"Inference speed and quantisation of neural networks with TensorFlow Lite for Microcontrollers framework"*.

Pete Warden and Daniel Situnayake discuss how to train models small enough to fit into any environment. This book is ideal for software and hardware developers who want to build embedded systems using machine learning. The book does not require experience in machine learning or any previous knowledge of microcontrollers, which makes it more intuitive for the reader. It guides you step by step in creating a series of TinyML projects. It concerns ultra low power microcontrollers. We have learned about the basics of ML and how to train our own models to understand audio, image and accelerometer data. There is also part dedicated to TensorFlow Lite for microcontrollers. We have also learned how to optimize latency, power consumption, model and binary size.[1]

We have also studied current techniques used for embedded devices that focus on compressing and optimizing the weights of the neural network architecture so the model fits and runs better on the embedded devices. Our source was mainly the article *Inference speed and quantisation of neural networks with TensorFlow Lite for Microcontrollers framework*.

In the article the authors describe how they were able to achieve significant size reduction of their neural network architecture which consisted of a large number of neurons and they also experimented with the effects of weight optimization on different devices.

The paper is mainly focused on effects of quantization and convertion of the model to TensorFlow Lite. The authors describe how they used the quantization technique and that they discovered that the model has to be rather huge in order to reach four times reduction in size. They describe that more usual size ratio of the regular and quantized model reaches approximately around 3.5.[2]

# 3 Description of the methodology

The methodology used was to learn about the subject first, in particular by reading existing scientific articles about current state of the art techniques which are used for solving similar problems.

Then, we planned to analyze the available resources in order to establish the specifications of our system for solving the problem. If we will find the provided format as insufficient or that it contains some unnecessary information, then we will modify it to match our needs. We will also visualize the data and then describe our observations.

We will then prepare a prediction model in Python and we will convert the prepared neural network to it's C representation using the **STM32CubeMX** and **SW4STM32** tools. After having the model prepared we will deploy it to the selected **1-core ARM MCU**.

## 3.1 Data analysis

We had access to a set of data including the acceleration values along the X, Y and Z axes as well as the pressure value of the instants spaced 40ms apart (25Hz frequency). In order to facilitate our study, we performed an analysis of the data. So we started by doing a principal component analysis (PCA), which is an analysis of statistical data that reduces the number of variables and makes the information less redundant.

It was noticed that 98.3% of the information was carried by the acceleration along the X axis. The acceleration along Y carried 1.6% of the information but this corresponded to noise (vibrations) around from the initial position. This is because the door does not move along the Y axis, making this component unnecessary. The acceleration along the Z axis did not carry any information, its value changed at the same time as that of the X axis. Finally, the value of the pressure did not carry any information, it is something that we could predict because no variation in the value was noticeable when observing the different cycles on Matlab.

This analysis therefore reduced our study to a one-dimensional problem, meaning that we now seek to describe the state and cycles of the garage door by reading only one input data.

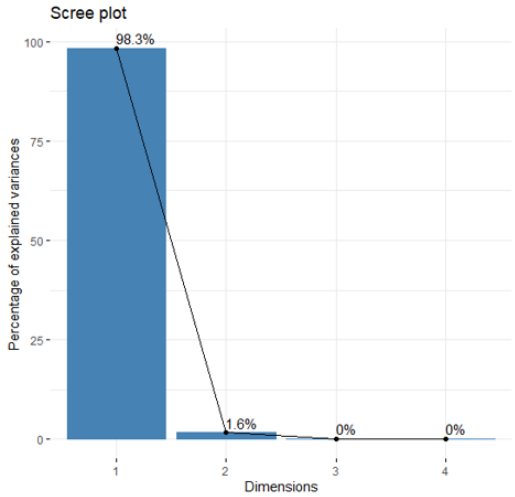



Figure 2: Principal Component Analysis results

## 3.2 Task division and applied approaches

In this section we would like to explain our approach to the problem and how we have decided to tackle it.

When looking for solutions to similar problems on the internet we were able to only find neural network model's handling sequence processing which were in most cases strictly related to Natural Language Processing problems. As the name hints, these problems are related to processing human written texts, which is not relevant for our problem which consists of a sequence of raw data values. After trying to somehow apply these approaches to our problem we concluded that we need to create our own, specific approach to solve the task. In order to do that we decided to split the task into two parts:

1. Creating NN model for predicting current state of the door for any given entry

2. Based on the output of the NN analyse the state sequences, clear them from noise and detect cycles in them

We have dedicated each of these sub-tasks it's own section, where we describe our approaches, results and notes.

# 4 Implementation

## 4.1 Neural network model

One of the main points of this project was to be able to fit trained NN model onto embedded device, in our case on the STM32 card. In order to do that we have decided to solve the first part of our problem by preparing suitable neural network for the garage door state prediction.

Provided data did not contain information about door state, so we first needed to prepare that value ourselves properly. As results of our previous data observations which is described in the section 3.1 we ended up working only with the X-axis data values. We have decided to recognize and detect 3 states of the garage door - closed door, open door and when the door is 50 % closed/open. We have found out, that the X-axis has specific values for each of these 3 states. We have used this knowledge to prepare the predicted value for our test data set.

The door state was based on following observations. Values of -10 and below are occurring only when the door is in closed state. Values above -1 represent open door state and all values between -10 and -1 we classified as the 50 % open/closed state.

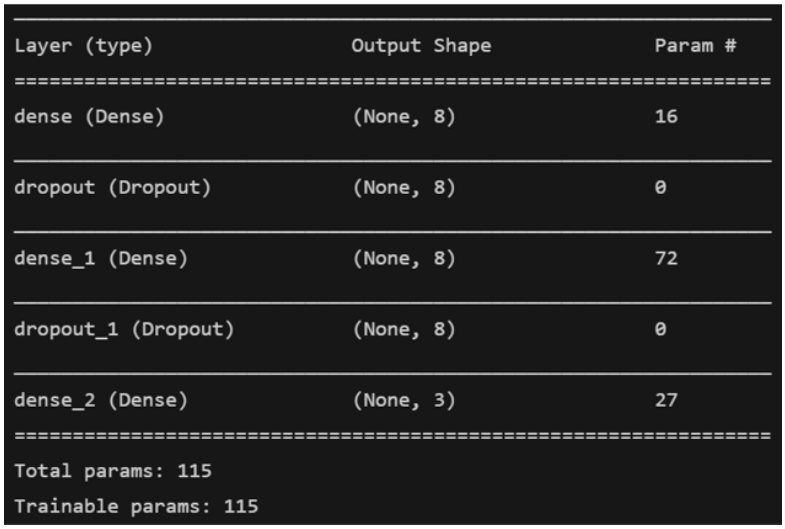
As train data, we decided to use only 1% of the total data provided. This small fraction of the data set was found to be more than sufficient for our model. Indeed, we want to predict a simple output: a state as a function of intervals of values.

It is important to note that 97% of the data corresponds to a closed state. This means that in our study, high precision will not necessarily mean a good ability to predict cycles if the predictions of states 1 and 2, representing a small part of the data, are bad.

We have experimented with different architectures of the neural network, but since the predicted value is based on only one feature we found out that even a very simple model will do just fine and is able to perform the predictions with very high accuracy. Our final model, which proved to be efficient enough with lowest possible size is shown in the Figure 3.

Our architecture is a DNN comprising: two dense layers of 8 nodes (neural network layer that feeds all outputs from previous layer to its neurons), and two dropout layers (prevents overfitting by randomly selecting nodes to be dropped out with a given input probability). The activation layers used were ReLU (Rectified Linear Unit) and softmax.

With this architecture we were able to obtain 97.69% accuracy on the remaining available data, which we found to be more than good enough. We had data from 2 different garage doors and we were able to achieve such high accuracy on data mixed from both of these doors. This way we have tried to avoid over-fitting of our model, but we agree that the predictions could be even better if we would have sensor data from even more different garage doors.



| Layer (type) | Output Shape | Param # |
|---|---|---|
| dense (Dense) | (None, 8) | 16 |
| dropout (Dropout) | (None, 8) | 0 |
| dense_1 (Dense) | (None, 8) | 72 |
| dropout_1 (Dropout) | (None, 8) | 0 |
| dense_2 (Dense) | (None, 3) | 27 |

Total params: 115
Trainable params: 115

Figure 3: Final neural network architecture

It's good to mention that we are aware that this problem could be solved and maybe it would be even more suitable to solve it by more simple prediction model for example by decision trees, but one of our goals was to successfully convert neural network model into C representation. That was the reason why we stuck with the neural network solution even when it was not necessary for our task.

## 4.2 Post-processing

The last step was to predict the different possible cycles using the prediction of the state of the door at each moment. This data post-processing was carried out in three stages.

We will try to carry out a demonstration with a recording sequence of 100,000 points visualized on Matlab (Figure 4). In the plot we can recognize 4 consecutive cycles of "closed open closed" type. Let's see if we can find this same number using our algorithm.

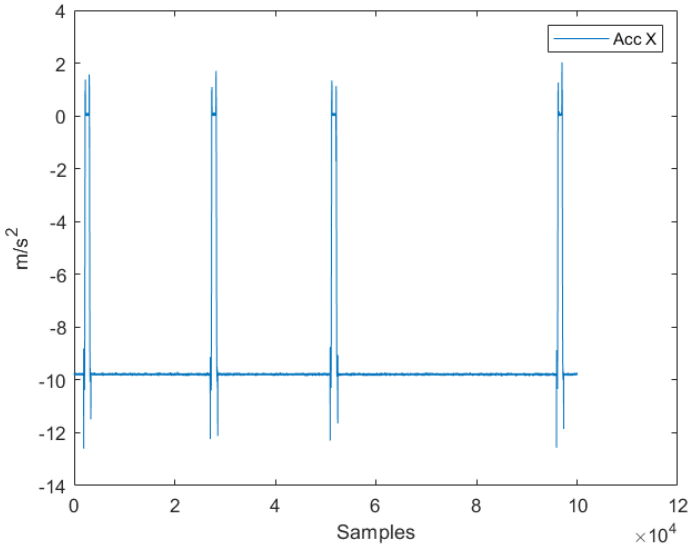



Figure 4: Observation of a recording sequence on Matlab

1. We get the prediction output for each of the input values sent to the neural network. The prediction we get is a probability of being in each of the three recognized states. Round off the prediction to get a value of 1 (dominant state) and two values of 0 (low probabilities). We count the number of points without discontinuity of the predicted class (Figure 5).

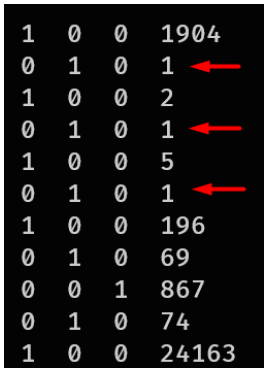



Figure 5: Rounding and grouping

2. Cycle analysis showed us that noise was present with every change of state. The most problematic noise is that of the points predicted as in the transition state while these are in the closed or open states (red arrows on Figure 5). We noticed that the transition states corresponded to a sequence of at least 70 points (transition time greater than 3 seconds). We have imposed the following rule: if a series of less than 60 points belongs to the transition class and the preceding and following series belong to the same class, then this series is aberrant and its class is changed to that of the previous series. Thus, there is no longer a discontinuity. Finally, if two series of points of the same class follow one another, they are grouped together (Figure 6).

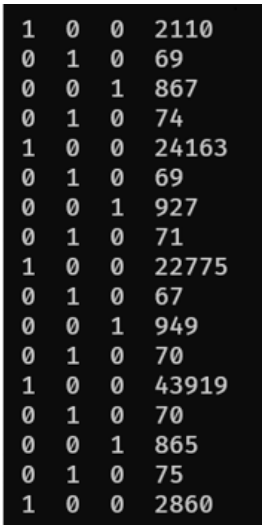



Figure 6: Noise elimination and grouping

3. The different cycles observed are counted on the assumption that the end point of a cycle is the starting point of a new cycle. We have chosen to use a decision tree which navigates each group of points one by one, and which determines the cycles according to the value of the state series present. We can thus count the number of cycles by type and have the total number of cycles (Figure 7).

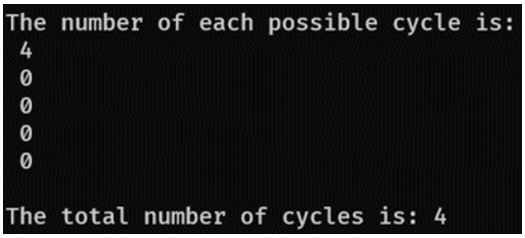



Figure 7: Cycle detection and counting

First, the implementation of post-processing was done in python for more efficiency, in particular thanks to the use of libraries like numpy. Then the goal was to convert this code to C or rewrite it since the code had to be compiled into C, according to the specifications.

We have tried using Cython which is a programming language and compiler that makes it easy to write compiled extensions for Python. Cython supported the used libraries well, but compilation was complicated for code like ours, especially because of the need to add many flags which were device dependant. We abandoned this interesting route and decided to finally rewrite the code entirely in C.

## 4.3 Application to the STM32 card

One of our main goals was to try to run our prepared model and post-processing on the selected integrated device. In class, we worked with the STM32 card, which we used as representative hardware to try and test our system even though it was not the best choice for our project, as we would expect our system to run on a device that can be connected to the garage door or at least one device with much larger memory, which can process the data recorded from the garage door sensor.

In the STM32 IDE, we have inserted the .tflite file generated using Python. The X-CUBE-AI expansion pack generated different C language files corresponding to the created neural network. The values of the acceleration along the X axis were stored in an input table at test phases. An empty array with the same number of rows has been initialized for state prediction. We configured the card to perform the inference on the map for each of the elements in the input array after pressing the blue button. From the prediction table, the post-processing step determined the number of cycles present in the input sequence.

As the output user interface for our task we decided to go for a very simple design that only displays the output of our system - the number of cycles detected in the processed data. The image of the user interface of the device can be seen in Figure 8.

At that time, we were well able to predict, using the map, each state and the cycles of a given sequence. However, we had a big memory problem, which was exceeded as soon as the number of acceleration values exceeded 2500. It should be remembered that the training data does not have less than 6 million lines. Indeed, the STM32 card has only a small memory capacity of 256 KBytes of RAM, it cannot store and calculate thousands of values at a time.

We know our model works and is just currently limited by the memory capacity of the card. The solution would be to send each input value to the

card through UART rather than storing them in a static array. Unfortunately, we didn't have time to test this alternative because we had to make the maps experimental.

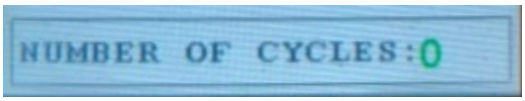



Figure 8: STM32 card user interface

# 5 Conclusions and future research

We were able to successfully fulfill all our goals. We were able to analyze the data and pre-process them for our model. We have also successfully prepared working neural network model with high accuracy rate for garage door state detection. Then we were also able to implement necessary post-processing in C to predict number of cycles based on the outputs of our model. In the end we were able to convert and fit our model and post-processing onto the STM32 card and prepare a working example of our system.

We noticed few things, which we think can be improved, but we didn't have enough time for them. One of them was the fact, that the embedded device we selected for our demonstration (STM32) didn't have enough memory to process larger amount of data. So for our demonstration we were only able to work with statically defined data of small size. As we have found out, this problem could be solved by reading the input directly from UART and loading it into dynamically allocated memory, but we had no time to learn how to read from UART and prepare sufficient solution so we leave this as a possibility for improvement that can be done for our project. But as we have said, we think that selection of another embedded device would be more beneficial rather that implementing reading from UART.

We found this project rewarding. The previous homework really allowed us to have all the bases necessary for the accomplishment of our tasks, to find the solutions to our problems, and to be more autonomous. Having a detection accuracy greater than 95%, we believe we are eligible for the price offered by the teachers.

# References

[1] *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers, Pete Warden, D. Situnayake* [online], [cit. 2019-12], Available at: `oreilly.com/library/view/tinyml/9781492052036`

[2] *Inference speed and quantisation of neural networks with TensorFlow Lite for Microcontrollers framework* [online], [cit. 2020-12-21], Available at: `ieeexplore.ieee.org/abstract/document/9221846`