DOI 10.5162/iCCC2024/P6

Monitoring of conveyor belt tension for condition monitoring using sensor fusion on embedded devices

Marcel Jongmanns, Fraunhofer Institute for Photonic Microsystems IPMS, Dresden, Germany
marcel.jongmanns@ipms.fraunhofer.de
Sharda Devi, Fraunhofer Institute for Photonic Microsystems IPMS, Dresden, Germany

Abstract

A conveyor belt was equipped with a sensor to measure vibrations. Based on these measurements, the belt tension should be determined. A low tension leads to slipping of the belt, while high tension strains the bearings of the belt. Both can lead to faster degradation of the mechanical parts of the conveyor belt such as the bearings. Two approaches to evaluate the data using artificial intelligence (AI) models on edge devices were tested. A microcontroller (STM32F4) running random forest classifier could not determine the state with high accuracy, while the trained model suggested during validation, that the accuracy would be >90%. A NVIDIA Jetson Orin Nano running a CNN also could not hold up to the expected accuracies from the theoretical validation of the model. However, using transfer training, the pre-trained model could be adapted to achieve the goal to determine the tension of the conveyor belt.

1 Introduction

Within the iCampus ForTune project, a toolbox was proposed for condition monitoring using sensor fusion and AI based evaluation algorithms [1]. The presented application builds upon the idea of this measurement system but focuses on local data processing on embedded hardware instead of using cloud-based services. In some cases, e.g. when data privacy of machine data or process data is an issue, or when mobile systems are used in environments without network connectivity, cloud solutions are not desirable.
A conveyor belt has been chosen as model to implement the condition monitoring system. In the presented application, the tension of the belt is monitored. At low tension, the belt is slipping and not running smoothly. At high tension, the belt exerts high forces on the bearings at the pulleys, which leads to a faster degradation of the bearings. Predicting the tension of the belt would give an indication, whether it is in a good state or gradually reaching a dangerous state, where it might slip of the track or cause a fire due to the friction in the bearing. If a non-normal condition is detected, a predictive measurement can be taken to prevent any damages to the system.

2 Measurement setup

A miniature conveyor belt (Vetter Kleinförderbänder GmbH) is equipped with an inertial measurement unit (IMU, STM LSM6DSOX) (Fig. 1). It is mounted via a 3D-printed adapter to the front of the conveyor belt near the idler pulley. The IMU combines an accelerometer, which measures lateral movements in 3 dimensions, and a gyrometer, which measures angular motions in 3 dimensions. Combined, this is a 6DOF sensor, which measures vibrations caused by the belt. The data is sampled at 3,333Hz over a period of 100ms for each sensor and dimension. The belt tension can be set using tensioning screws. This mechanism is equipped with a marker, which allows to set the tension to a predefined level. These tension levels are labeled with numbers from 3 to 7. At level 3, the belt tension is so low, that the belt does not move smoothly and slips. Level 5 is defined as the good state, where the belt tension is at a medium level. At level 7, the belt tension is increased to a level, where the current consumption of the motor increases by more than 10%. This has been validated by supplying the conveyor belt with a laboratory power supply and observing the current.

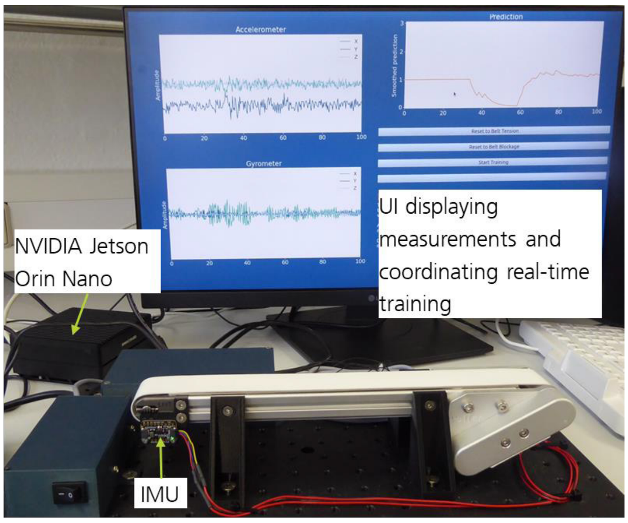
Accelerometer
Prediction
Gyrometer
NVIDIA Jetson Orin Nano
UI displaying measurements and coordinating real-time training
IMU

Figure 1: Laboratory setup of the measurement system. The IMU sensor is attached via a 3D printed adapter to the frame of the conveyor belt. The screen shows the measurement data of the accelerometer (top left) and gyrometer (bottom left).

The data is transmitted to a NVIDIA Jetson Orin Nano single board computer. This mini-PC features a 6-core ARM processor, 8GB RAM, a GPU with 32 tensor cores with a maximum power consumption of 15W. This device is used to acquire the data for the AI model and to implement a software for real-time evaluation of the conveyor belt state.
The acquisition of the data for AI training was divided into 10 measurement series. In each series, 100,000 measurements were performed for each of the following states: the belt is *off*, the belt is running at tension level *3*, *4*, *5*, *6* and

7. This approach includes changes to the system due to environmental parameters and slight changes in the belt tension for each category. In a first screening of the data it became apparent, that the measured data changed over the first 3 measurement series. This comparison was based on statistical parameters such as amplitude and frequency of the individual time series data for each sensor. The data for the *off* category was comparable between all 10 datasets, however the data for the running belt changed. We suspect that the repeated changing of the tension from very low to very high wore the belt down, until it reached a stable state after the first few measurement series. The final data evaluation is done on the last 7 dataset, giving a total of 7 series * 6 categories per series * 100,000 datapoints per category = 4.2M datapoints. The full data set has been published in the Fraunhofer Fordatis repository [2].

3 Evaluation

In a first step, classic machine learning (ML) approaches, such as random forest classifier and regressor, have been tested. The calculated parameters from the sensors are adapted from Hamadache et al. [3]. From the time series, the following parameters are calculated: median, mean, root-mean-square, root-mean-square error, and skewness. In addition, the following are calculated from the Fourier transform of the data: center frequency, root-mean-square, and root-mean-square error. The ML training and validation have been performed on a NVIDIA DGX System. The inference has also been tested on an ARM Cortex M4 controller (STM NUCLEO-F446RE development board) in a real-time application.
In a second approach, neural networks (NN) have been tested. Instead of calculating parameters manually, the time series data of the sensors is evaluated using a convolutional neural network (CNN). Since the sensor data consists of 6 series, 3 each for the accelerometer and gyrometer, the CNN has 6 input layers. Each of them has the same layout with a CNN, pooling and normalization layer, but each of the 6 units is trained specifically on one time series. These 6 layers are then combined using a concatenation layer and flattened, so that a final prediction of the state can be made using dense layers (Fig. 2). The training and validation were also performed on the NVIDIA DGX system, but the

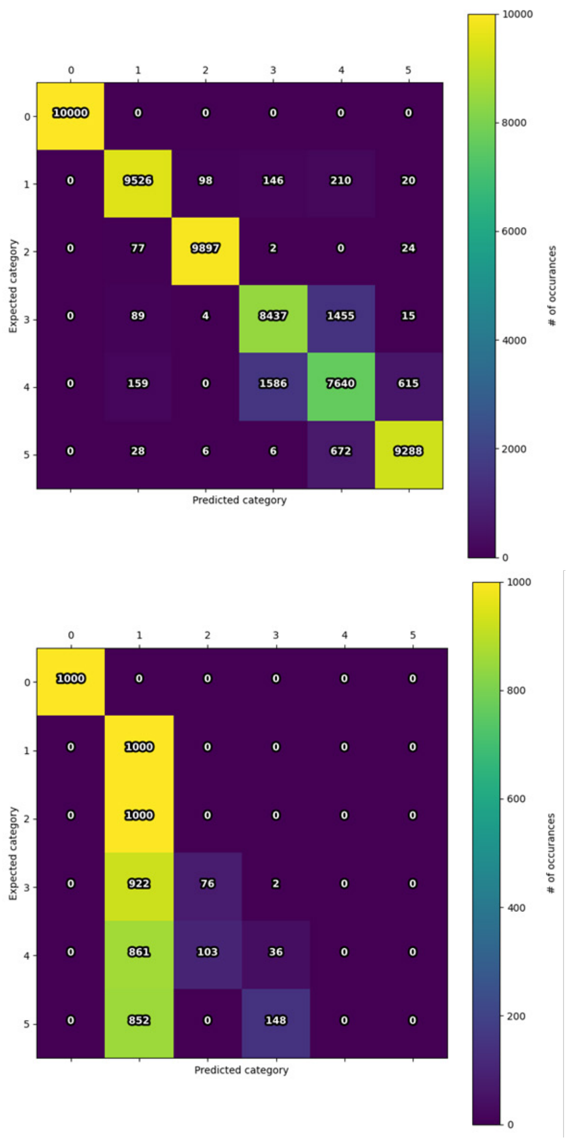

Figure 3: The confusion matrices for the RFC during the validation process (top) and real-time application (bottom). The classes (0 to 5) refer to state *off* (0) and tension levels *3* (1) to *7* (5).

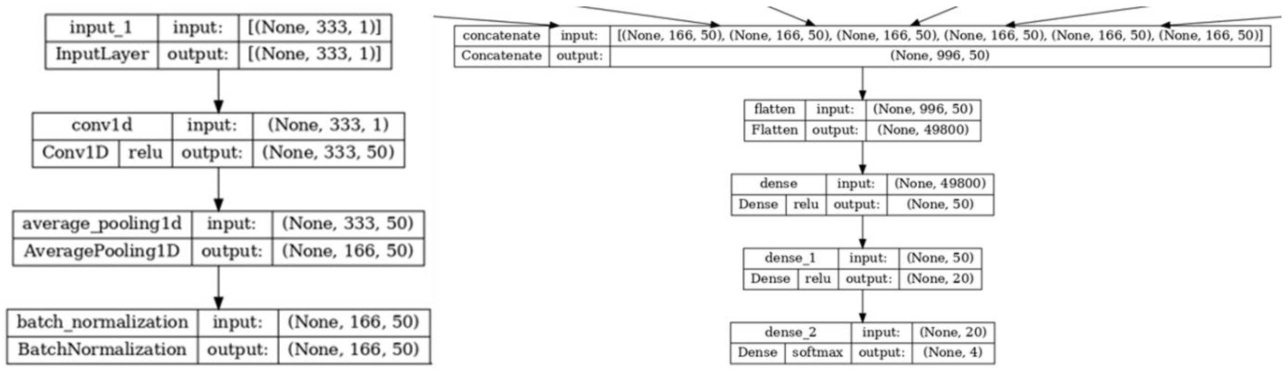
input_1 | input: | [(None, 333, 1)]
InputLayer | output: | [(None, 333, 1)]
conv1d | input: | (None, 333, 1)
Conv1D | relu | output: | (None, 333, 50)
average_pooling1d | input: | (None, 333, 50)
AveragePooling1D | output: | (None, 166, 50)
batch_normalization | input: | (None, 166, 50)
BatchNormalization | output: | (None, 166, 50)
concatenate | input: | [(None, 166, 50), (None, 166, 50), (None, 166, 50), (None, 166, 50), (None, 166, 50), (None, 166, 50)]
Concatenate | output: | (None, 996, 50)
flatten | input: | (None, 996, 50)
Flatten | output: | (None, 49800)
dense | input: | (None, 49800)
Dense | relu | output: | (None, 50)
dense_1 | input: | (None, 50)
Dense | relu | output: | (None, 20)
dense_2 | input: | (None, 20)
Dense | softmax | output: | (None, 4)

Figure 2: Layout of the CNN. Each of the 6 data series of the sensors is processed by their respective CNN (left). These are then combined and evaluated using dense layers (right).

real-time inference was performed on a NVIDIA Jetson Orin Nano system.
The training was done in Python 3.8.10. For the NN, TensorFlow 2.11.0 was used [4]. The same software setup was used on the NVIDIA Jetson Orin Nano for the real-time validation. The ML was trained using scikit-learn 1.3.2 [5]. The resulting classifier was transformed to C code using emlearn 0.18.1 [6]. The code for data acquisition and inference on the STM NUCLEO-F446RE was written in STM32 CubeIDE 1.13.1.

4 Results

The ML model did overall perform well in theory, i.e. training and validation using the collected data, but did not perform well in the real-time application (Fig 3). Several random forest classifiers and regressors with varying hyperparameters were tested. Since the models were evaluated on an ARM Cortex controller, the model needed to be compact to fit in the memory of the controller. The presented model has 10 estimators with a maximum depth of 10 per estimator. It reaches an accuracy of 91,3% in the validation, but only 33,4% in the real-time application running the inference on the controller. It can be detected when the belt is *off*, but all 5 different tension levels are mostly predicted to be the lowest tension level, with only a few exceptions.
The CNN model reached an accuracy of 94,1% in the validation process (Fig. 4). Like the previous case, the accuracy in the real-time application on the NVIDI Jetson Orin Nano was much lower with only 59,9%. The predicted tension is lower than the set tension as well, but there are some differences between the levels and not every case defaults to the lowest tension. A differentiation between *low*, *medium* und *high* tension could be made.
To simplify the model and make it more reliable in the real-time application, the following steps have been taken. First, the number of classes have been reduced. Instead of 6 classes (*off* and tension level *3* to *7*), only 4 classes (*off*, tension *low*, *medium* and *high*) have been used. To do this, tension levels 3 and 4 were combined to *low*, 5 and 6 to *medium* and 7 was labeled *high*. By doing this, the accuracy from the CNN increased to 99,4% during validation. Further, a transfer training was implemented in the software. This allows to adapt the CNN on the NVIDIA Jetson Orin Nano. The calculation-intensive CNN layers are fixed and precalculated on the NVIDIA DGX system based on the initial data. The DNN layers are re-trained based on recent measurement data on the embedded system. With this approach, the NN can be calibrated. For each of the 4 result classes belt *off*, *low* tension, *medium* tension, and *high* tension, 100 datapoints are collected. The whole process from collecting the data to finalizing the transfer training takes less than 5 minutes. The belt tension needs to be adapted manually between each step and takes most of the time.

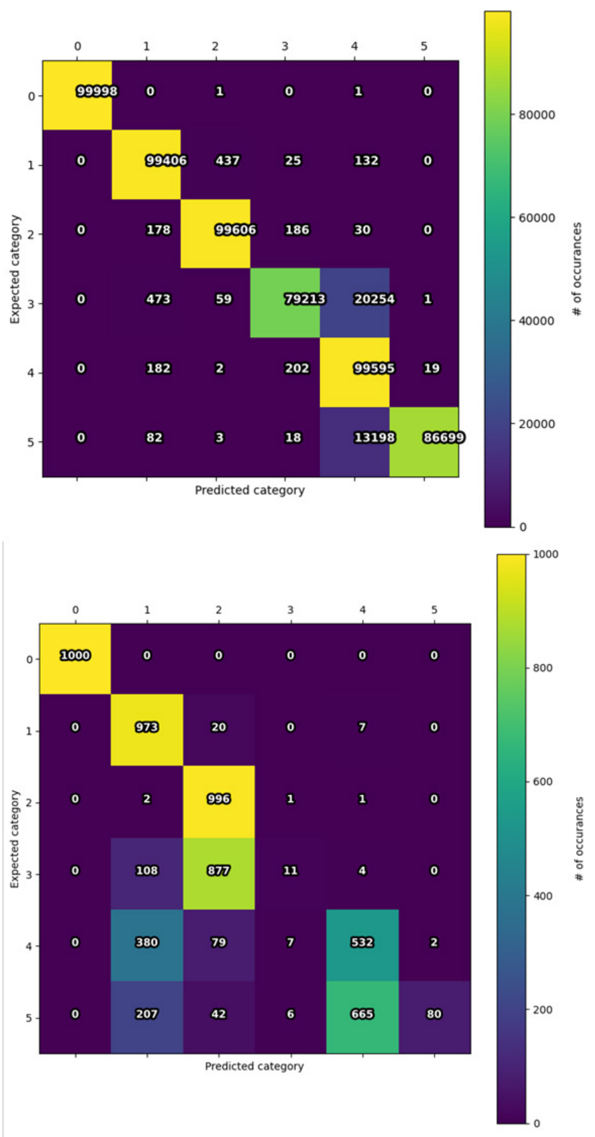

Figure 4: The confusion matrices for the CNN of the validation process (top) and real-time application (bottom). The classes (0 to 5) refer to state *off* (0) and tension levels *3* (1) to *7* (5).

5 Discussion

The idea of the iCampus ForTune toolbox has been tested on embedded systems. The ML approaches based random forest classifiers did not reach the same accuracies in the real-time tests as in the initial validation. Due to the limited possibilities, no further optimizations have been performed. The accuracy of the CNN models was in the real-time application also lower than in the initial validation of the model.
In all cases, the tension is predicted to be lower than it is. This suggests a change in the overall system as already mentioned in the measurement setup. During the collection of the required training data, the system was strained too much. Thus, the data evaluation needed to be adapted accordingly.

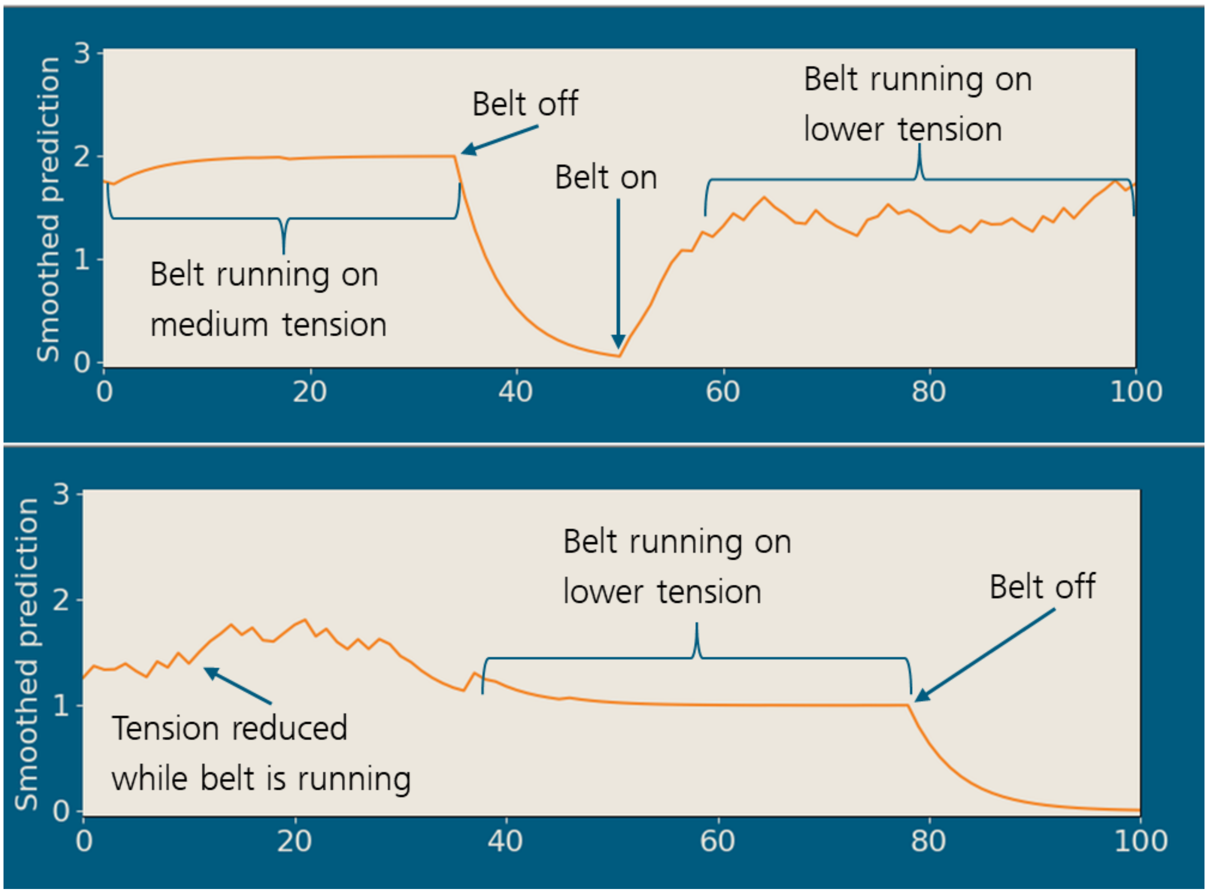

Figure 5: The output of the predicted tension of the belt after transfer training. the X-axis shows the last 100 predictions and the y-axis the predicted class. **Top**: The belt is running, then turned off (class 0). The belt tension is lowered. Then the belt is activated again. Due to the smoothing, the tension is now detected between low (class 1) and medium (class 2). **Bottom**: The tension is reduced while the belt is running. Afterwards, the tension is predicted to be low (class 1). Then the belt is turned off again (class 0).

However, due to the transfer training capabilities of the NN, a real-time calibration is possible. Using this approach it is possible to differentiate between a *low*, *medium* and *high* tension of the conveyor belt. Since the result is presented as moving average, a regression to show steps between the defined classes can also be achieved (Fig. 5).
This application shows that the idea of a toolbox-base condition monitoring could overall be realizable, but the implementation might be more challenging than expected. The prediction of the AI models needs to achieve a minimum accuracy to be reliable. Training a of compact AI model for edge devices was not able to achieve this without further calibration steps.

6 Acknowledgment

This work has been funded by the Federal Ministry of Education and Research of Germany (BMBF) within the project iCampus Phase 2, grant number 16ES1128K.

7 Literature

[1] M. Assafo, M. Lautsch, P. Suawa, M. Jongmanns, M. Reichenbach, M. Hübner, C. Brockmann, D. Reinhardt and P. Langendörfer, "The ForTune Toolbox: Building Solutions for Condition based and Predictive Maintenance Focusing on Retro Fitting," in MikroSystemTechnik Kongress 2023, Dresden, Germany, 2023.

[2] M. Jongmanns and S. Devi, "Data set for AI assisted detection of the belt tension on a conveyer belt for condition monitoring," 2023. https://fordatis.fraunhofer.de/handle/fordatis/347. DOI: 10.24406/fordatis/289

[3] M. Hamadache, J. H. Jung, J. Park and B. Youn, "A comprehensive review of artificial intelligence based approaches for rolling element bearing PHM: shallow and deep learning," JMST Advances, vol. 1, pp. 125-151, 2019. DOI: 10.1007/s42791-019-0016-y

[4] TensorFlow, https://www.tensorflow.org/, Accessed 2024

[5] scikit-learn - Machine Learning in Python, https://scikit-learn.org/, Accessed 2024

[6] J. Nordby, M. Cooke and A. Horvath, emlearn: Machine Learning inference engine for Microcontrollers and Embedded Devices, 2019. DOI: 10.5281/zenodo.2589394