AML term project

Introduction

The main purpose of this project was to introduce a way to use microcontrollers in machine learning. Initially I set my goal as to deploy a model in hardware that is able to classify something based on data from connected sensors. At the time I'm working on a exoskeleton project and one of the final goals of it is to design a control system for exoskeleton based on the data from sensors(IMU, angular encoder, force sensor).

So after searching the internet I found a few usefull material in form of articles and books, then I've come up with idea of detecting different hand movements using IMU sensor.

The final solution of this project could be potentially used in fencing(swordsplay)-like game where based on type of movement a player performs different action. (like defending or attacking).

Here is the list of used informations sources:

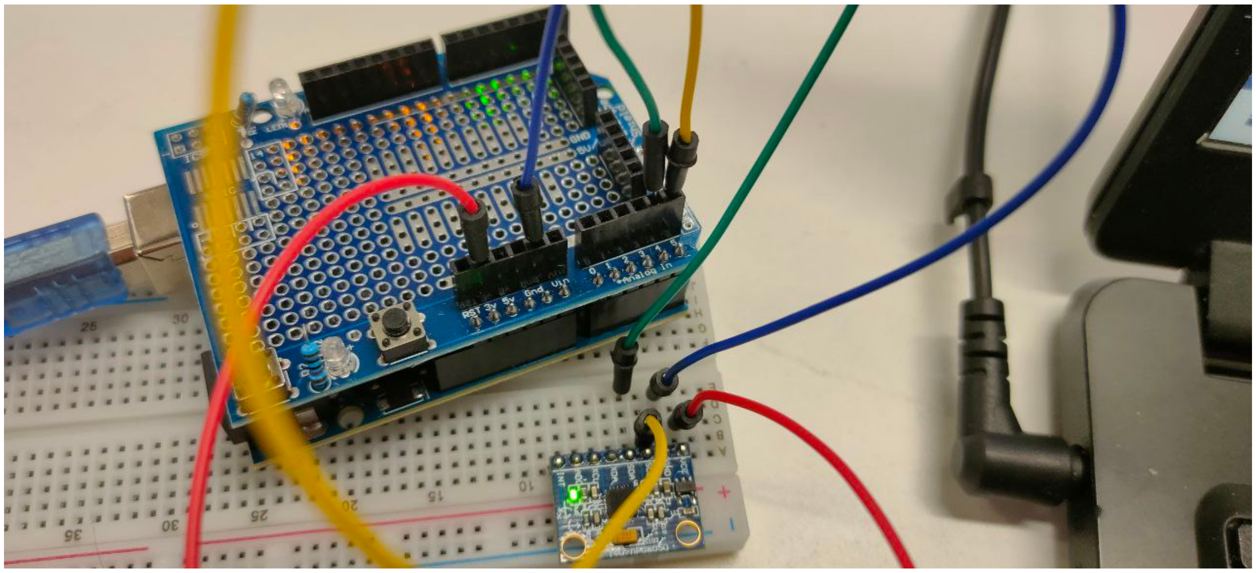
- An article on using ML on ESP32 for recognizing different movements
- A book about techniques for using machine learning in microcontrollers
- An article on how to deploy your model on arduino controllers

Step 1: Data collection

For this project I decided to use an IMU sensor connected to Arduino UNO V3. The model of my sensor - MP6050. It has a gyroscope and accelerometer, which provide angular velocities and angular accelerations. Since I used PIO in visual studio on Windows for working with arduino I decided to take data from serial port using PuTTY app.

The code for collecting the data can be found in the github of the project. It simply prints the timestamp in ms from the start of the program and the data of accelerometr and gyroscope for this moment from the IMU sensor.

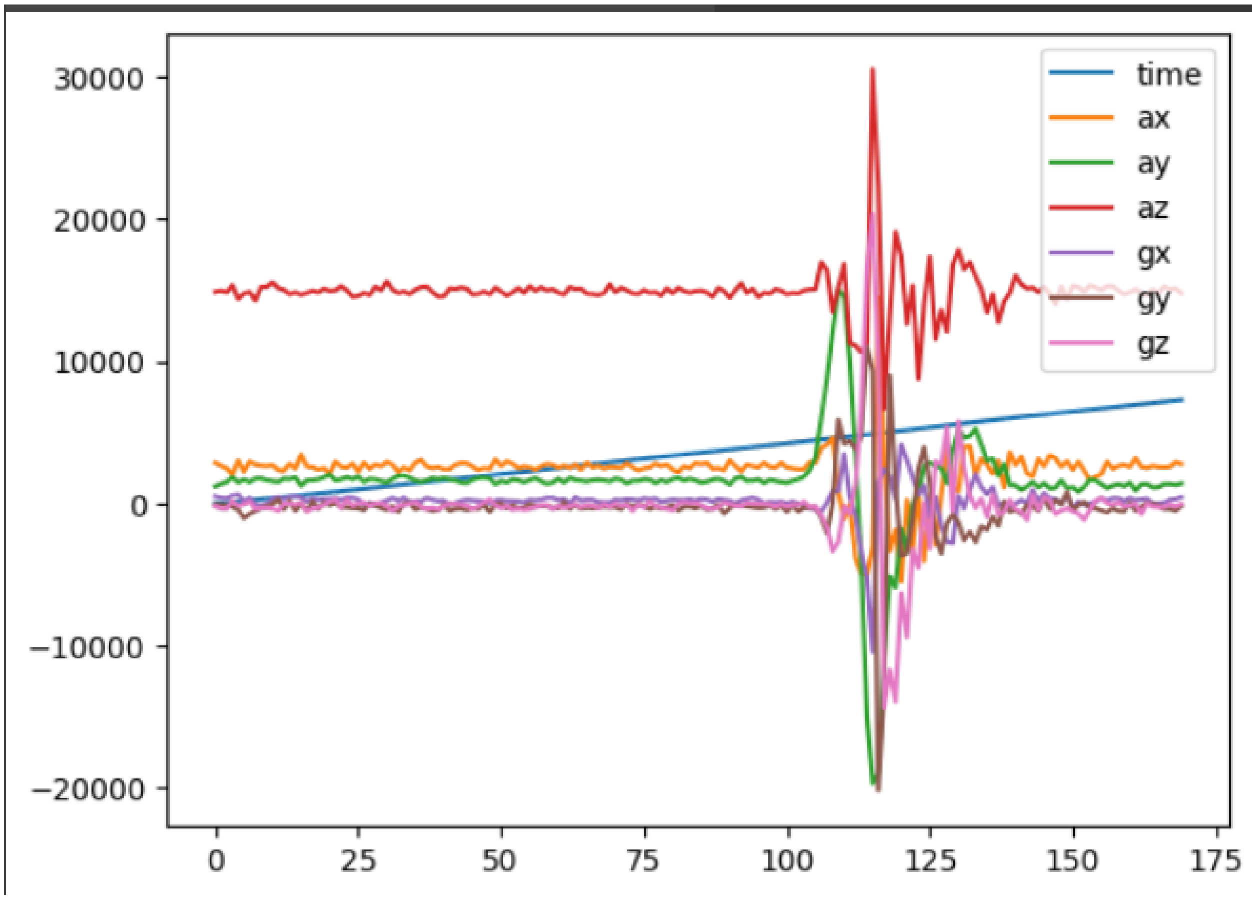
Each file represents exactly one movement (for circular movements exactly one cycle). Each file name consist of a name of a movement and time when it was started (seconds from the start of recording)



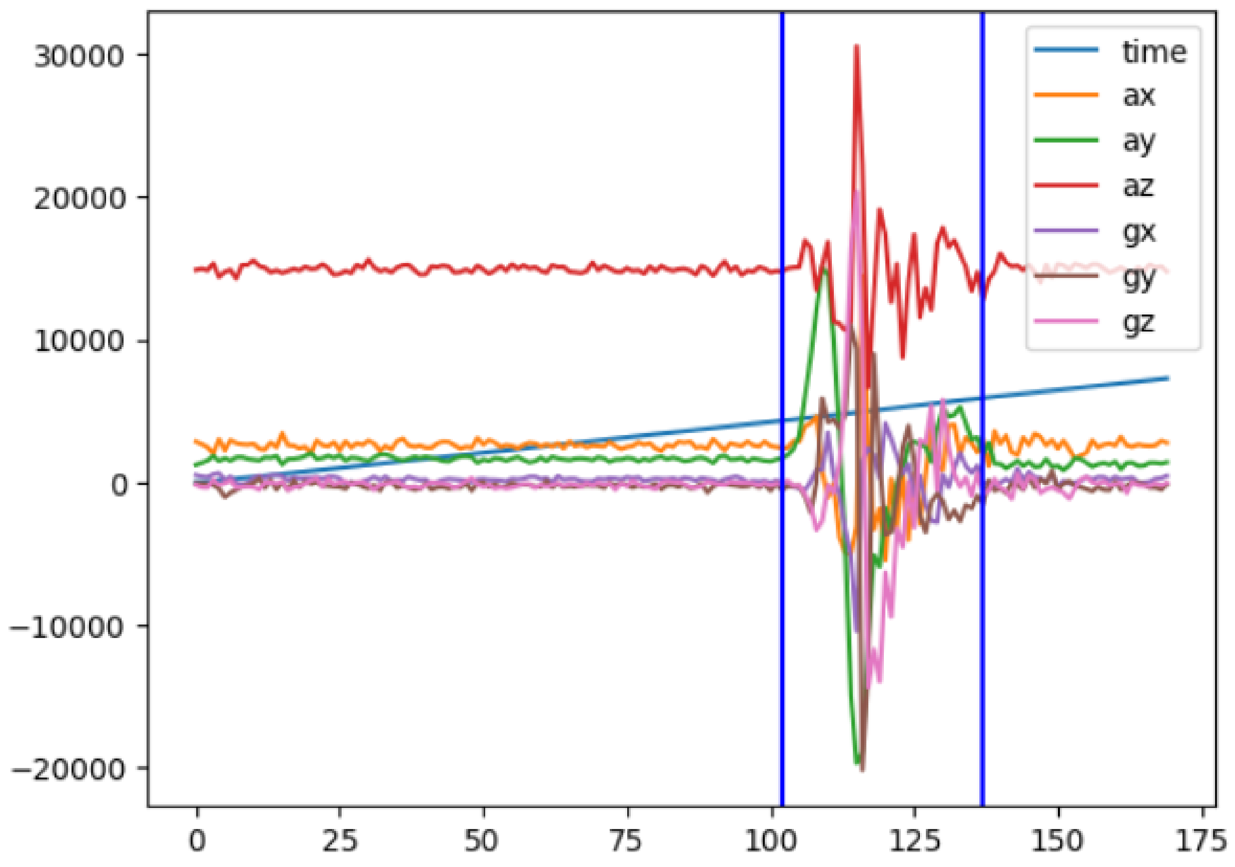
Step 2: Data processing

After collecting data as .csv (actually .txt with comma separated values) I had to label it. There are different resources that can help you to label time-series data. One that was presented in the article - label studio. However I didn't succeed in creating my own template within a reasonable time, so I did it manually in collab.

The idea is straightforward - I plot the data from .csv file



Choose appropriate boundaries for the spikes in the plots



And then simply add the target column where everything outside of the boundaries classified as “rest” and inside as a specified movement (given in the name of a file)

Step 3: Training model

Since microcontrollers are much less powerful in terms of computations and memory we have several limitations on the models.

Obvious one - the model cannot be too large - therefore I cannot use very complex neural networks for it.

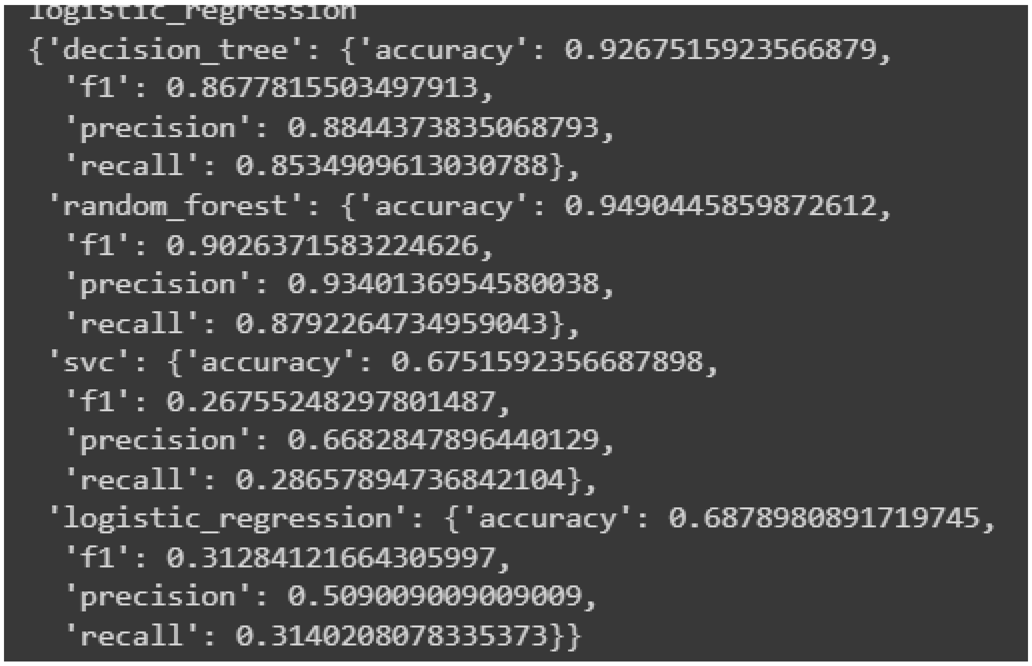
Less obvious - if the model deployed onboard, then the inference time should be much less than the sampling ratio.

There are also requirements for the input data. Two different movements can have the same segments (like moving up in clockwise or counterclockwise direction), so I have to use a window of time in my time series data. Therefore I have to modify my data for that task.

I decided to take a window of timestamps and flatten the values into one row that will represent an input for my model. So since I have 6 measurements from my sensor the input shape is 6* WINDOW_SIZE.

After combining all the labeled .csv files and converting them to the described above dataframe (and also encoding the target) I started to try different models.

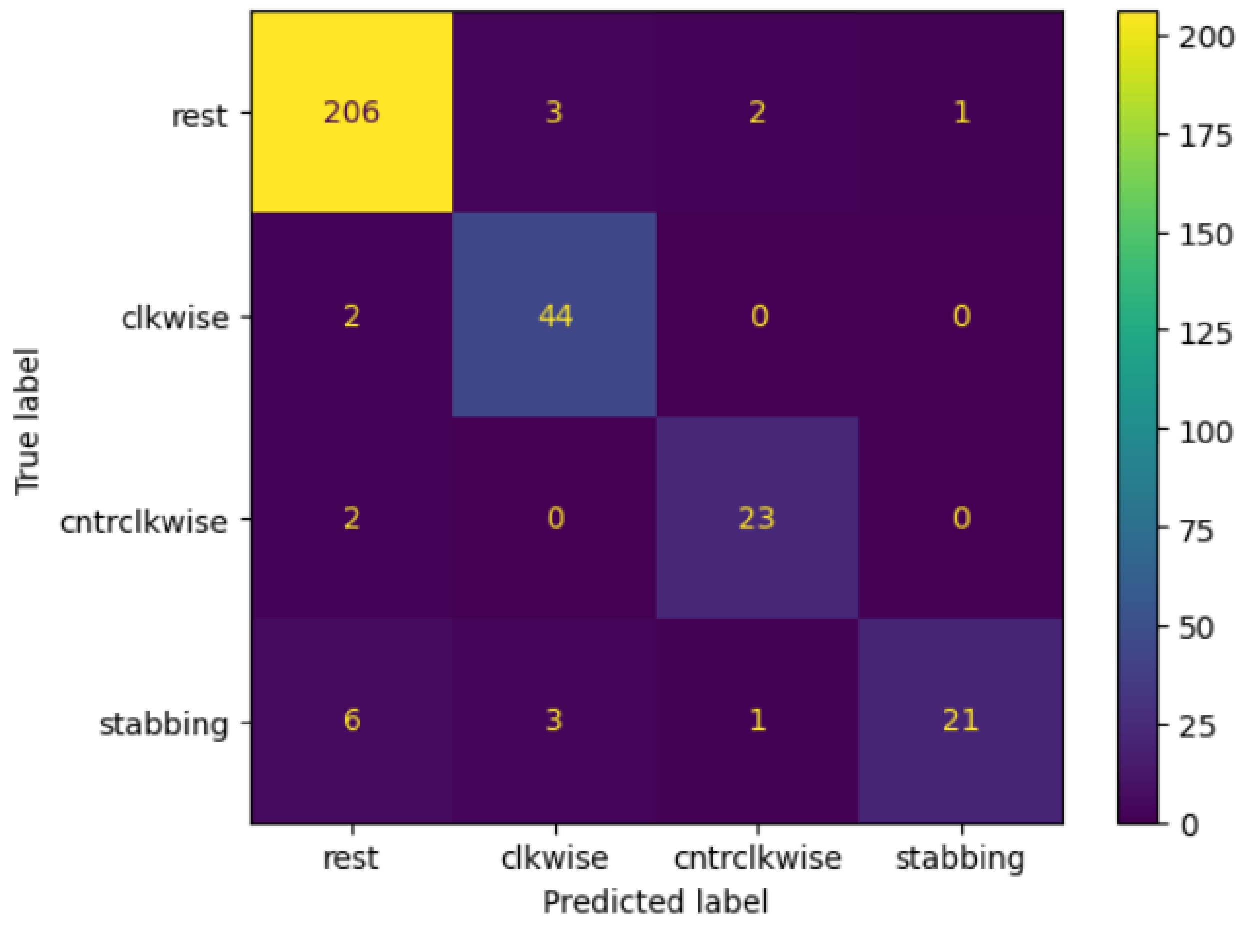
Here are the results



```
logistic_regression
{'decision_tree': {'accuracy': 0.9267515923566879,
  'f1': 0.8677815503497913,
  'precision': 0.8844373835068793,
  'recall': 0.8534909613030788},
 'random_forest': {'accuracy': 0.9490445859872612,
  'f1': 0.9026371583224626,
  'precision': 0.9340136954580038,
  'recall': 0.8792264734959043},
 'svc': {'accuracy': 0.6751592356687898,
  'f1': 0.26755248297801487,
  'precision': 0.6682847896440129,
  'recall': 0.2865789473684210 4},
 'logistic_regression': {'accuracy': 0.6878980891719745,
  'f1': 0.31284121664305997,
  'precision': 0.509009009009009,
  'recall': 0.3140208078335373}}
```

As you cans see decision trees are actually quite good. For the random forest I used 4 estimators, so the model is not that big. So I choose to use it.

Here is the confusion matrix of my final model



From it you can see why the recall is quite small - because the stabbing movement has almost 33% missclassified. You'll see how it affect the deployment.

Step 4: Deployment

Finally I have the trained model, but I have to somehow put it on my microcontroller. There was a proposed solution to use m2cgen library, that allows you to convert scikit-learn models to different languages, however currently there is no conversion to pure c++. You can use conversion to pure C language, however I run into problems with compiling the code to arduino. Because of that I used another library - micromlgen that allows to convert skelearn classifiers to model.h files for arduino boards.

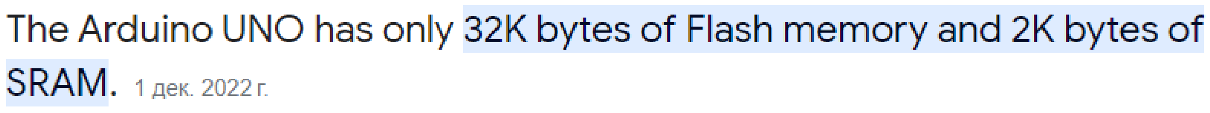
Here is a view of the generated code for my random forest classifier

Then I had just to write a few functions the will automatically sample my data in almost the same frequency as for data collection and predict the current state based on the sampled data

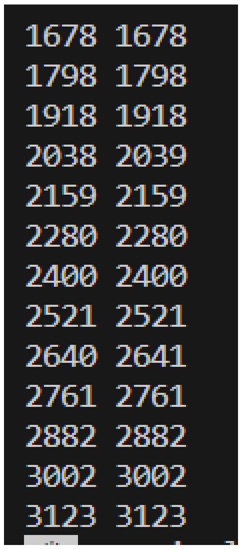
Here is a video example

Step 5: Problems overview

The current model is small enough to fit in the Arduino UNO and to have small inference time



The Arduino UNO has only 32K bytes of Flash memory and 2K bytes of SRAM. 1 дек. 2022 г.



```
1678 1678
1798 1798
1918 1918
2038 2039
2159 2159
2280 2280
2400 2400
2521 2521
2640 2641
2761 2761
2882 2882
3002 3002
3123 3123
```

The time in ms from the start of the program before and after model prediction

So the main problem - accuracy. As you may notice from the video and model metrics - my model has problems with predicting the stabbing movement. Since I used random forest classifier I could use some interoperable methods to find the problem. However since my input data is in a format of 10 different time samples of the same sensor it could be hard to determine why ax0 is more important than gy5. But anyway I could check it in the future.

When working with the model I can try to obtain more different data (in terms of displacement from the initial position), so the stabbing movement is less restricted to exactly one coordinate.

From the point of end-device optimizations - I could introduce a debouncing filter, that after detecting the stabbing movement was less agile to changes in the data, so it not that frequently changes the state.