Machine Learning on Distributed Platforms

# Introduction

With an increasingly use of drone in different areas such as agriculture, construction and parcel delivery there is also a growing number of use cases for machine learning algorithms. Due to reduced hardware costs and advantages that come with multiple drones working together – swarms of drones where the drones communicate with each other and deploy machine learning algorithms together are going to be used more and more in the coming years. The use of swarms can prevent the unprecedented growth of data collections as well as reducing the complexity of algorithmic solutions needed for solving complex problems.

However, before implementing complex machine learning algorithms on swarms there are several challenges that must be tackled as for example parallelisation of algorithms, statistical heterogeneity or the limited performance of the single drones.

In order to build a solid foundation for the implementation of other machine learning algorithms, the Fachprojekt “Digital Design for Machine Learning” focuses on the implementation of a machine-learning-framework. This framework will provide the basic functionality for the deployment of more complex algorithms on STM32 hardware (and other hardware) that can be used for well scaling drone swarms.

# Motivation (kann auch weggelassen werden als überschrift)

(Alongside the distribution of machine learning algorithms on distributed platforms come several challenges that must be considered.

* The ML-Algorithms need to have a high degree of parallelism, so we can make use of the distributed computational power.
* The performance of the hardware and the workload on each drone needs to be considered.
* Communication between the drones is highly resource expensive and therefore needs to be reduces to an absolute minimum.
* Data collected from different drones might show statistical heterogeneity if the collection is not well implemented.

Problems that normally come with distributed machine learning are privacy and system heterogeneity, but because we assume using the same hardware for all the drones, we can ignore problems that come with system heterogeneity. Privacy concerns are negligible as well because we are going to reduce the communication to the sending results of aggregation and updated weights only (see more in chapter Schneider) and therefore no sensitive data can be derived from this data. )

For testing the framework, we will implement random forest as machine learning algorithm in order to classify the data received.

# Theory

## How we tackle the problems

## Advantages of distributed machine learning

## Topology

## Random forest

## Federated learning for future projects

## Current solutions

# Implementation

## Platform

As platform for testing the implementation we used the STM32F407 Discovery board. It comes with a 32-bit Arm based, single core processor, gets powered by an external power supply of 3 or 5 Volts and is compatible to a variety of different sensors such as temperature, video and audio sensors. Already onboard are an accelerometer, LEDs that we used for debugging (read more at chapter Debugging) and full support for the STM32CubeIDE that is developed by STMicroelectronics as well.

## Software for Implementation

### IDE

As mentioned, the board is we used is compatible with the STM32CubeIDE, that we used for implementing the framework as well as the machine learning algorithm. The IDE is based on Eclipse and supports all the add-ons that come with Eclipse.

The STM32CubeIDE comes with a lot of helpful features such as:

* The built in STM32CubeMX that is used for configuring the Pinout, clock of the processor and peripheral hardware.
* A code generator that makes the definition and addressing of the different Pins easier.
* Advanced debug features like CPU fault analysis, register views and real-time tracing.

For this project we used the 64-bit version for Windows 10 and used the STM32CubeProgrammer in addition for erasing the memory before debugging.

### Debugging

There are different ways for debugging the STM32 boards:

* Using the serial cable that we also use to power the STM32F407
* Using the special STLink Debugging cable
* Using the built-in LEDs
* Reading values directly out of the memory

Unfortunately, we could not get the console input and output to run, therefore we used the built-in LEDs in order to debug our code part by part. The implementation for using the LEDs is quite simple as the right declaration and initialisation is already done by the built-in code generator. We simply used the pre-defined method HAL\_GPIO\_TogglePin() in order to turn on a LED.

### Programming Languages

As programming language, we used C++ because its performance advantages and our previous knowledge. There is also the possibility of using C as well as deploying TensorFlow and, hence using python. However, when using TensorFlow we could not have made use of the features that come alongside the IDE but would have access to a lot of useful libraries for more complex machine learning algorithms.

### Implementing the framework and random forest

When generating a new project within the STM32CubeIDE it auto-generates the main file consisting of the main method and the initialisation of the pins used.

#### Decision tree class

In addition to the main class we created another class called decisiontree.cpp that contains the implementation of the Table, Node and DecisionTree objects.

(Code einfügen?)

The DecisionTree object is later used to build up a decision tree on each drone. When building the tree the drone uses ….. to guess ….. . (Bitte ergänzen was der da genau macht, da hast du mehr durchblick)

(Code einfügen?)

#### Main method

The main method basically consists of two main parts, the Initialisation and the while loop.

Within the initialisation part we set up empty tables that are then used to learn and save the results. Therefore, we simply initialize the table objects table, obstable (observation table) and restable (result table). Furthermore, a Boolean variable called master is declared and set. This variable determines whether a drone is only there for building a decision tree and later sending it, or if the drone is the master drone/central drone that is responsible for aggregating all the results gathered from the other drones.

When entering the while loop, first the role of the drone is checked. If the drone does not need to do the aggregation, it simply builds the decision tree by calling the corresponding method and afterwards sends the leafs of the tree that was generated.

If on the other hand the drone needs to do the aggregation, it first of all receives all the results generated by the other drones and then builds the modus (in case of random forest with modus) of all the results. The result can then be sent back to the drones (This will give future projects the possibility to send updated weights when using “reenforced – das ist falsch” machine learning)

The while loop runs indefinite and repeats the same steps again for new data provided by the sensors.