

# Contents

<b>1</b>	<b>Planning and design of early warning system</b>	<b>2</b>
1.1	Hardware . . . . .	3
1.1.1	Nucleo-F767ZI . . . . .	4
1.1.2	Wisent Edge tracker . . . . .	4
1.1.3	Flir Lepton 2.5 camera module and Lepton breakout board .	6
1.1.4	PIR Sensor . . . . .	8
1.2	Firmware . . . . .	8
1.2.1	Tools and development environment . . . . .	8
1.2.1.1	Development environment for STM32f767ZI . . . . .	8
1.2.1.2	Development environment for NRF52840 . . . . .	9
1.2.2	Architecture design . . . . .	9
1.2.3	MicroML and build system . . . . .	11
1.2.4	Running inference on a microcontoller . . . . .	12
1.2.5	Wisent board control firmware . . . . .	13

# 1Planning and design of early warning system

General structure and tasks of an early detection system were already described in chapter ???. As it was mentioned before, an early detection system consists of two different components:

1. Several small embedded devices, deployed in the field. They capture images with thermal camera, process them and send results over wireless network.
2. One gateway, which is receiving results, and relays them to an application server over internet connection.

In this chapter we focus on the structure and design of deployed embedded system, both from hardware and firmware point of perspective. We also describe construction of an application server, how received data is processed, stored and presented.

The general block diagram of an embedded system with a thermal camera is presented on the Figure 1.1

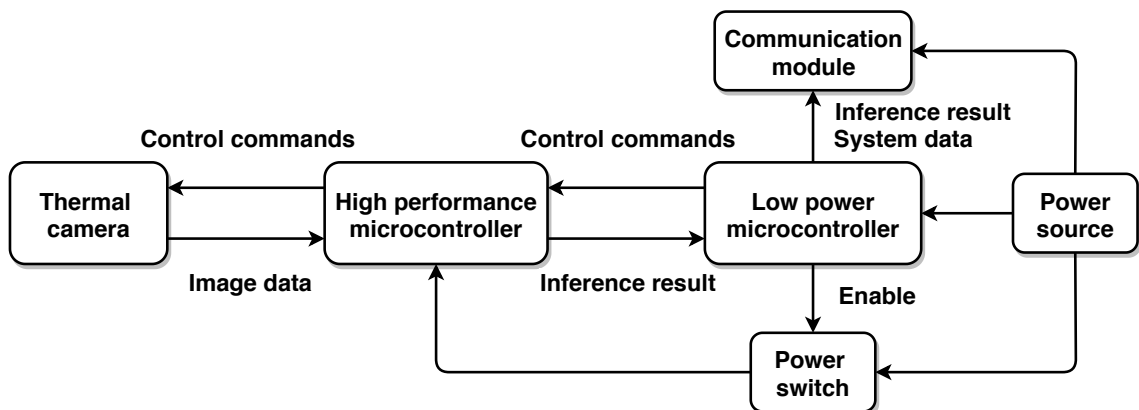


Figure 1.1: General block diagram of an embedded system

Embedded system will consist of two different microcontrollers with two distinct tasks, a thermal camera, PIR sensor, wireless communication module, power switch and battery.

Powerful, high performance microcontroller and thermal camera are turned off, to conserve battery life. A less capable, but low power microcontroller will spend most the time in sleep, waiting for a trigger from PIR sensor. PIR sensor will point in the same direction as the thermal camera and will detect any IR radiation of a passing object.

If an object passes PIR's field of vision, it triggers it, which in consequently wakes up a low power microcontroller. Microcontroller will then enable power supply to high performance microcontroller and thermal camera, and send a command request for image capture and processing.

Thermal camera only communicates with high performance microcontroller, which configures it and requests image data. That data is then inputted into neural network algorithm and an probability results are then returned to a low power microcontroller. low power microcontroller then packs the data and sends it over radio through wireless communication module. Power source to high performance microcontroller and thermal camera is then turned of to conserve power. Diagram of described procedure can also be seen on Figure 1.2.

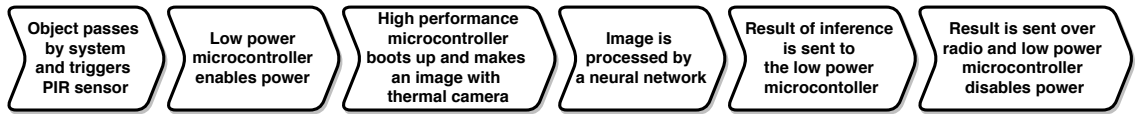


Figure 1.2: Diagram describing behavior of embedded early detection system

## 1.1 Hardware

In this section we present concrete components that we used to implement the embedded part of the early detection system. Hardware version of embedded system diagram is presented on the Figure 1.3. It should be noted that we did not include specific power source into the diagram. Wisent Edge tracker board is general enough

to work with different power sources, such as non-chargeable or chargeable batteries and or solar cells.

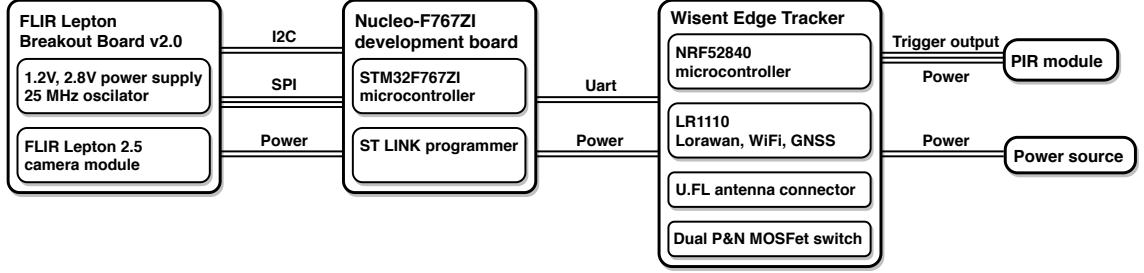


Figure 1.3: Hardware diagram of embedded early detection system

### 1.1.1 Nucleo-F767ZI

Nucleo-F767ZI (seen on Figure 1.4) is a development board made by STMicroelectronics. Board features STM32F767ZI microcontroller with Cortex-M7 core, which has 2 MB of flash, 512 kB of SRAM and can operate at clock speed of 216 MHz. It also features different memory caches and flash accelerator, which provide extra boost in performance. It is convenient to program it, as it includes on board ST-LINK programmer circuit.

We chose this microcontroller simply because it is one of more powerful general purpose microcontrollers on the market. As we knew that neural networks are computationally expensive to compute and that models can be quite large in terms of memory, we selected it knowing that we can always scale down, if we have to.

### 1.1.2 Wisent Edge tracker

For the part of the system which had to contain low power microcontroller, communication module and power control for Nucleo-F767ZI board we decided to use Wisent Edge tracker board. Wisent Edge (seen on Figure TODO ADD IMAGE) is a tracker solution, specifically developed for conserving endangered wildlife animals. It is one of many tracker solutions that were a product of open-source<sup>1</sup>collaboration between Irnas and company Smart Parks, which provides modern solutions in anti-

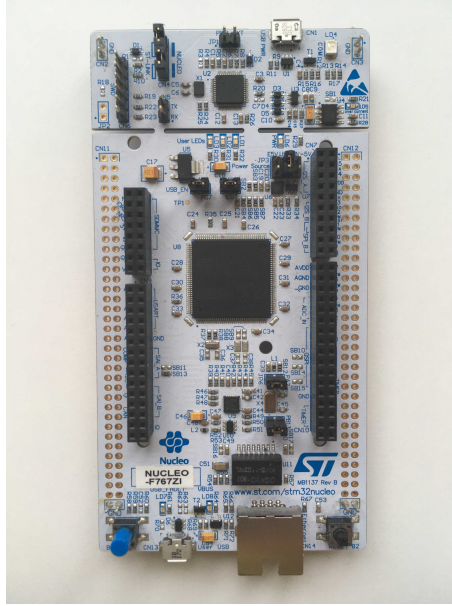


Figure 1.4: Nucleo-F767ZI development board

poaching and animal conservation areas.

The main logic on the board is provided by Nordic Semiconductor’s NRF52840 microcontroller with Cortex-M4 core, which has 1 MB of flash, 256 kB of RAM and Bluetooth 5 support. NRF52840 has consumption of 0.5  $\mu$ A in sleep mode, which makes it ideal for our purpose.

Wisent Edge also features Semtech’s LR1110 chip (which acts as a LoRa transceiver, GNSS and WiFi location module) and another GPS module, U-blox’s ZOE-M8G<sup>2</sup>. There is a ceramic GPS antenna on board and a U.FL connector to which a dual band Wifi, Bluetooth and LoRa antenna can be attached.

As geopositioning of system was not primary concern, GNSS functionalities were not used, however they might be usefull in future.

Power control of a Nucleo-F767ZI board and FLIR camera is provided by a dual channel p and n MOSFET, circuit can be seen on Figure TODO ADD FIGURE. Circuit functions as a high side switch, with microcontroller pin driving enable line.

<sup>1</sup>As a part of OpenCollar project, the design of Wisent Edge is open-source and available on GitHub [1], alongside other hardware and firmware tracker projects.

<sup>2</sup>Reason for two GNSS modules is that although LR1110 chip can provide extremely power efficient location information, it’s accuracy is smaller when compared to ZOE-M8G and it can only be resolved after sending it to an application server [2].

When enable line is low, n MOSFET is closed, therefore p MOSFET is also closed, as it is pulled high by resistor TODO ADD NUMBER. When enable line is high, n MOSFET is opened, therefore gate of p MOSFET is grounded, which opens the MOSFET.

### 1.1.3 Flir Lepton 2.5 camera module and Lepton breakout board

In section ?? it was described what kinds of thermal cameras exist and how do they work, and in section ?? it was described why FLIR Lepton 2.5 was chosen. However, not much was said about what sort of support circuitry FLIR camera needs and how do we actually make images with it.

FLIR Lepton camera needs to be powered from two different sources, 1.2 V and 2.8 V, as well it needs a reference clock of 25 MHz. All of this is provided by Lepton breakout board, which can be seen on the Figure 1.5. Front side of the breakout board contains a FLIR module socket and back side has two voltage regulators and an oscillator. Breakout board can be powered from 3.3 to 5 V and also conveniently breaks out all communication pins in form of headers.

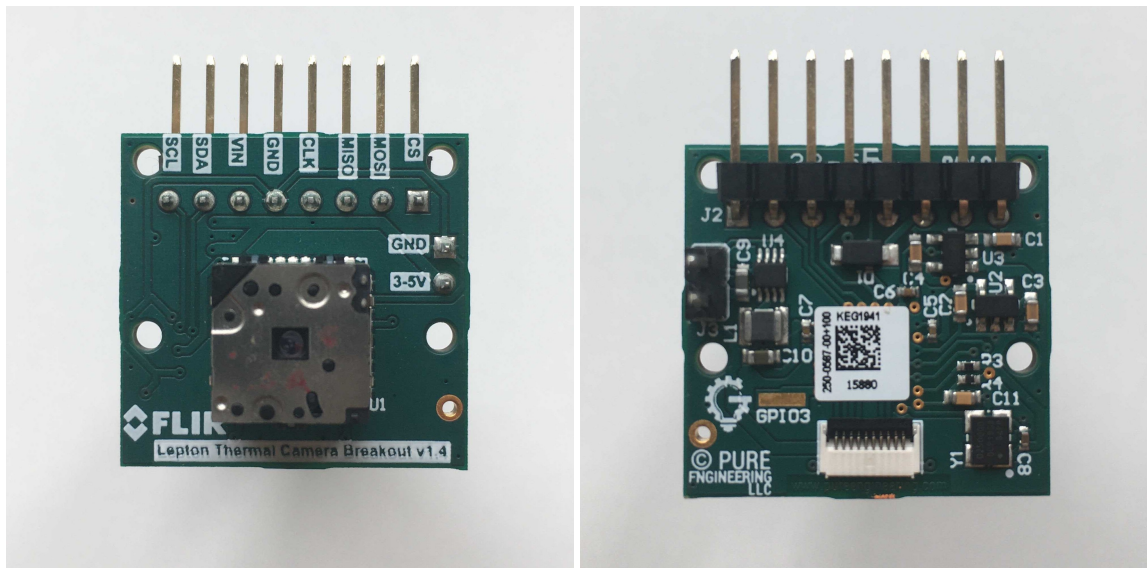


Figure 1.5: Front and back side of Flir Lepton breakout board with thermal camera module inserted.

FLIR Lepton module itself contains five different subsystems that work together and can be configured:

- AGC – Automatic Gain Control, affects image contrast and quality
- SYS – System information
- VID – Video processing control
- OEM – Camera configuration for OEM customers
- RAD – Radiometry

AGC subsystem deals with converting a dynamic range of an IR sensor into a compact range that is more suitable for storing and displaying images. In case of FLIR Lepton this is a 14-bit to 8-bit conversion. For our purposes AGC subsystem was turned on, as the input to our neural network were 8-bit values.

Microcontroller communicates with FLIR camera over two interfaces: two wire interface (TWI) is used for sending commands and controlling the FLIR camera and Lepton's VoSPI protocol is used for image transfer.

TWI is a variation of an I2C protocol, instead of 8 bits, all transfers are 16 bits. Internal structure of Lepton's control block can be seen on the Figure 1.6. Whenever we are communicating with FLIR camera we have to specify which subsystem are we addressing, what type of action we want to do (get, set or run), length of data and data itself.

Lepton's VoSPI protocol is used only to stream image data from camera module to the microcontroller, which means that MOSI line is not used. Each image fits into one VoSPI frame and each frame consists of 60 VoSPI packets. One VoSPI packet contains an 2 bytes of an ID field, 2 bytes of an CRC field and 160 bytes of data<sup>3</sup>, that represents one image line.

Refresh rate of VoSPI frames is 27 Hz, however only every third frame is unique from the last one. It is a job of the microcontroller to control the SPI clock speed and process each frame fast enough so that next unique frame is not discarded.

---

<sup>3</sup>Because images pixel values fit into 14-bit range by default, it means that one pixel value needs two bytes of data (two most significant bytes are zero). That means that each image line (80 pixels) is stored into 160 bytes. If AGC conversion is turned on, each pixel is then mapped into 8-bit range, however the size of one line in VoSPI packet still remains 160 bytes, 8 most significant bits are simply zeros.

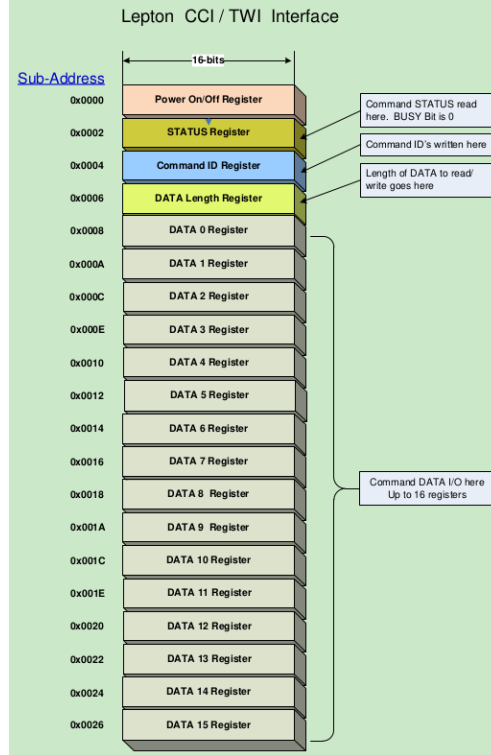


Figure 1.6: Command and control interface of FLIR Lepton camera.

#### 1.1.4 PIR Sensor

### 1.2 Firmware

#### 1.2.1 Tools and development environment

For our firmware development we did not chose any of various vendor provided integrated development environments. We instead used terminal text editor Vim for writing and editing the code.

As we were programming two different microcontrollers, we were using different tools with each one.

##### 1.2.1.1 Development environment for STM32f767ZI

For building our firmware programs we used GNU Make, build automation that builds software according to user written *Makefiles*. For compilation we used Arm embedded version of GNU GCC. To program binaries into our microcontroller we



used OpenOCD.

As a hardware abstraction library we used libopenm3, which is a open-source low level library that supports many of Arm’s Cortex-M processors cores, which can be found in variety of microcontroller families such as ST’s STM32, Toshiba’s TX03, Atmel’s SAM3U, NXP’s LPC1000, Silabs’s EFM32 and others. Libopenm3 provided us with linker files, startup routines, thinly wrapped peripheral drivers and a starting template makefile, which served as a starting point for our project.

As libopenm3 does not provide `printf` functionality out of the box we used excellent library by GitHub user mpaland [3]

#### **1.2.1.2 Development environment for NRF52840**

To develop firmware for NRF52840 we decided to use The Zephyr OS, which is a small kernel, designed for IoT embedded systems. Besides usual RTOS functionalities such as tasks, mutexes, semaphores it also provides common driver API for supported microprocessors.

### **1.2.2 Architecture design**

STM32F767ZI firmware was designed to be very efficient and lean, only truly necessary parts of firmware were implemented.

As seen on Figure 1.7 we split the firmware into two hardware and application modules.

Both modules are lightly coupled, which enables us reuse of the application module on a different hardware sometime in the future. Hardware specific module is mostly using libopenm3 API to set the system clock and initialize peripherals. Small function wrappers had to be written to make use of various peripheral drivers more abstract. To profile execution of our code we first wrote a timer driver based on a Arm’s systick timer, however we later decided to use data watch trigger (DWT). DWT does not use interrupts, therefore it does not introduce overhead of calling interrupt routines like systick timer does.

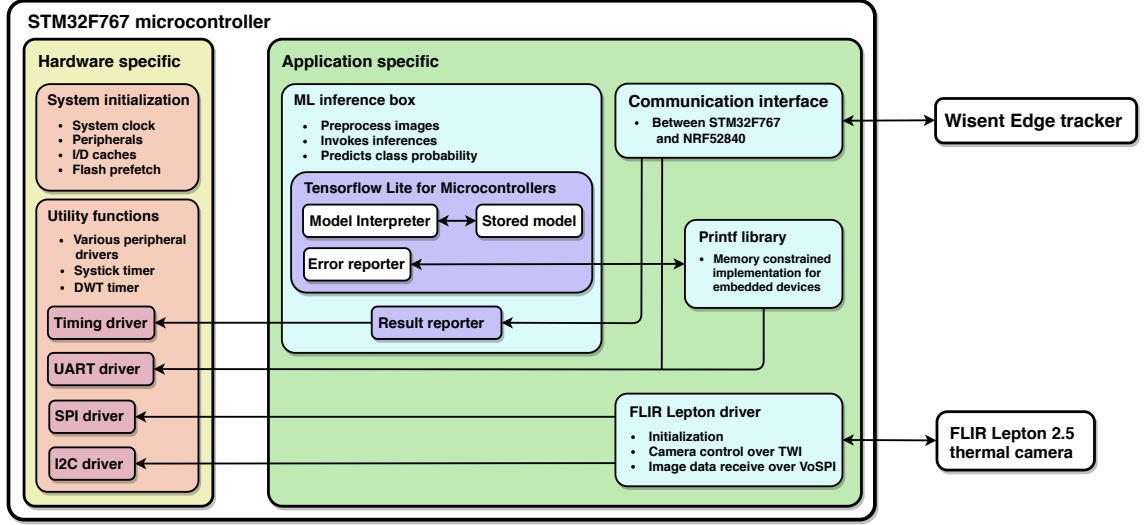


Figure 1.7: Architecture diagram of the firmware that is running on a STM32F767ZI microcontroller.

FLIR Lepton driver was written from scratch, as many libraries provided either by camera manufacturer or open source communities were too complex and implemented many features, which we did not require.

Thanks to TFLite Micro API, ML inference module could be written as a simple black box. Image data goes in, predictions come out.

TODO describe communication interface between THIS AND WISENT EDGE.

The architecture diagram for NRF52840 can be seen on Figure 1.8. For NRF52840 microcontroller, we did not had to write any peripheral drivers, as they were provided by Zephyr itself. Priority was to achieve low power consumption, for which NRF52840 had to spend most of its time in sleep mode, such behavior was easily configurable in Zephyr.

This functionality is encapsulated in inference requester module, which is also waking up the microcontroller when PIR trigger signal is received and controlling the MOSFET.

We also wrote communication module, which takes care of controlling the LR1110 chip, joining LoRaWAN network, preparing outgoing messages and sending them over LoRaWAN network.

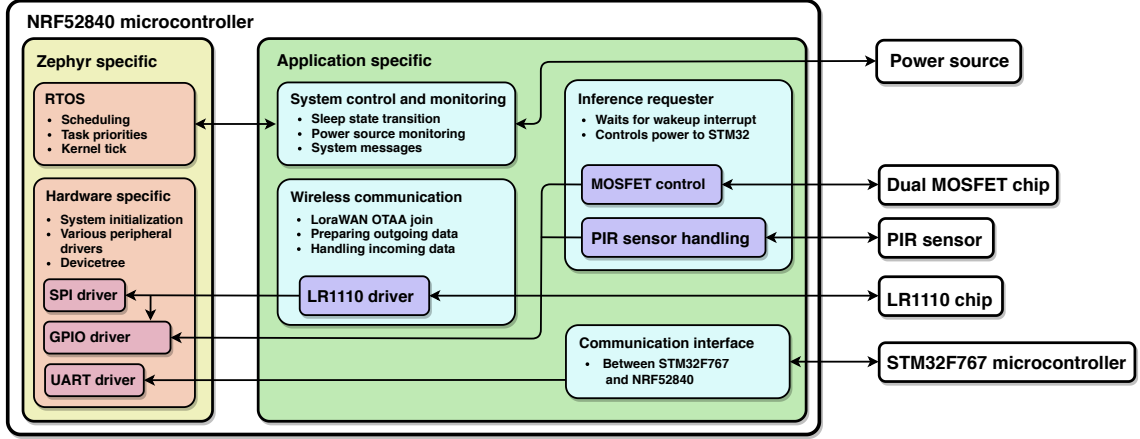


Figure 1.8: Architecture diagram of the firmware that is running on a NRF52840 microcontroller.

TODO describe communication interface between THIS AND WISENT EDGE.

### 1.2.3 MicroML and build system

Large part of this thesis was concerned with porting TFLite Micro to libopenm3, our platform of choice. To understand how this could be done, we first had to analyze how the code is build in TFLite Micro.

To compile source files and build binaries TFLite Micro uses GNU Make. Main makefile that includes several platform specific makefiles dictates how firmware is built and several scripts which download various dependencies. By providing command line arguments users decide which example has to be compiled and for which platform. The build system makes some assumptions about locations of the platform specific files, which in case of example projects are scattered over the whole TensorFlow GitHub repository.

We learned a useful principle while observing the build process. Each time we would build an example for a new platform, Make would first compile all TensorFlow files, create a static library out of them, compile specific example source files and then link against library in linking stage. If we wanted to build firmware for a different example, Make would only had to compile source files of that example and it could reuse previously made library. As compiling of static library took quite some time, this was an efficient option.

After analyzing the TFLite Micro's build system we created a list of requirements that we wanted to fulfil on our platform.

1. We wanted to keep project specific code, libopenm3 code and TFLite Micro code separated.
2. We wanted a system, where it would be easy to change a microcontroller specific part of building process.
3. We wanted to reuse static library principle that we saw in TFLite Micro build process.

Covering different platforms and use cases made main TFLite Micro makefile quite complex and hard to understand. This meant that it would be hard to reuse it while porting to a new platform and we needed a different approach or reuse something else.

To solve our problem we started building a small framework that we called MicroML<sup>4</sup>. MicroML enables users to develop ML applications on libopenm3 supported microcontrollers. Frameworks directory structure can be seen on Figure YOU STOPED HERE

In section TODO ADD REFERENCE we describe our build system that keeps library and application specific code separated in a clear and concise manner.

Why you created it, what problems does it solve that using tensorflow introduced, how does it work (makefiles), diagram problems that you had. Generally you should follow your github page. TEST SETUP FOR RUNNING INFERENCE ON A COMPUTER

#### 1.2.4 Running inference on a microcontroller

explain how tensorflow works on a micro, interpreted approach vs boilerplate code.

---

<sup>4</sup>Project is open-source and hosted on GitHub [4].

### 1.2.5 Wisent board control firmware

flow diagram cli interface (this should be put somewhere, or not?)

## Bibliography

- [1] OpenCollar, Collection of open-source conservation solutions, GitHub repositories. Available on: <https://github.com/opencollar-io>, [26.10.2020].
- [2] Mustafa L., Sagadin M., LR1110 chip: one solution for LoRa and GNSS tracking. Available on: <https://www.irnas.eu/lr1110-chip-one-solution-for-lora-and-gnss-tracking/>, [26.10.2020].
- [3] Mpaland, A printf / sprintf Implementation for Embedded Systems, GitHub. Available on: <https://github.com/mpaland/printf>, [27.10.2020].
- [4] Sagadin M., MicroML, Quick-start machine learning projects on microcontrollers with help of TensorFlow Lite for Microcontrollers and libopenm3, GitHub repository. Available on: <https://github.com/MarkoSagadin/MicroML>, [27.10.2020].