

# Machine Vision in a Marine Environment

**Van Havermaet Toon**

professional bachelor Electronics-ICT (Telecommunications Engineering)

2023-2024

**Häme University of Applied Sciences**

Tietotie 1

37630 Valkeakoski

Finland



## Power of attorney

## **Abstract**

(T. Van Havermaet)

This project, a collaboration between Odisee University of Applied Sciences, Häme University of Applied Sciences (HAMK), and the city council of Valkeakoski, focuses on deploying machine vision technology for boat classification and tourism monitoring. The project falls within HAMK's Electrical and Automation Engineering program, as well as Odisee's Electronics-ICT program.

The aim is to assess water traffic during the touristic season in Valkeakoski, a city located where the lakes Mallasvesi and Vanajavesi meet. These lakes, among the 50 largest of Finland, attract tourist activity during summer.

The project seeks to accurately identify different types of boats. After detections, an image of the boat is stored locally and offline, on an SSD and can be used to enhance the accuracy of the model in the future. Furthermore, the system displays real-time boat count data on a self-hosted dashboard using Node-RED and logs this data daily.

The paper details the strategic placement of the project for optimal performance, the development and validation of a YOLO machine vision model, and the integration of LoRaWAN for offline, long-range wireless communication. It delves into the world of the Raspberry Pi SBC and camera module and uses the Python programming language for seamless integration between the different components.

Keywords: Machine Vision, Machine Learning, YOLO, LoRa, LoRaWAN, Node-RED, Raspberry Pi, Python

[Toon.vh13@gmail.com](mailto:Toon.vh13@gmail.com)

<https://github.com/Bonsa-BE>

---

# Exchange Details

---

**Exchange Student**

Toon Van Havermaet

**Course**

Electrical and Automation Engineering

**Academic year**

2023/2024

**Internship**

12/02/2024 - 30/05/2024

**Supervisor**

Maxim De Donder

**Work placement**

Häme University of Applied Sciences

Tietotie 1

37630 Valkeakoski Finland

**Mentors**

Jussi Horelli

Juhani Henttonen

## Table of Contents

List of Figures.....	4
List of Code Fragments .....	5
List of Tables.....	6
Introduction .....	8
1 Placement of the Project .....	9
1.1 Linnosaari Island .....	9
1.2 Enclosure .....	9
2 Machine Vision Model .....	11
2.1 Data Acquisition.....	12
2.2 Data Cleaning and Annotation.....	13
2.3 Training of the Model.....	14
2.4 Validation of the Model .....	16
2.5 Testing and Evaluation of the Model .....	19
2.6 YOLO and licensing .....	21
3 LoRaWAN .....	22
3.1 Terminology behind LoRa and LoRaWAN architecture .....	22
3.2 Regional Parameters for EU863-870 MHz Band.....	23
3.2.1 Channels .....	24
3.2.2 Spreading Factor and Coding Rate .....	24
3.2.3 Data Rate .....	24
3.2.4 Maximum Transmission Power.....	25
3.2.5 Maximum Payload Size .....	25
3.2.6 Duty Cycle .....	25
3.3 LoRaWAN in Finland .....	25
3.4 Project Implementation: LoRaWAN Module .....	27
4 Raspberry Pi .....	29
4.1 Raspberry Pi Foundation and Raspberry Pi Ltd.....	29
4.2 Motivation for Raspberry Pi 4B .....	29
4.3 Project Implementation: Raspberry Pi Modules .....	30
5 Camera module .....	31
5.1 Project Implementation .....	31
5.2 Alternatives for Future Use .....	31
6 Dashboard.....	33
6.1 MQTT .....	33
6.2 Node-RED .....	33
6.3 Project Implementation: Visualization .....	34
6.4 Project Implementation: Data Logging .....	37

7	Main Code Analysis and Explanation .....	40
7.1	Systemd.....	40
7.2	Crontab.....	40
7.3	Send_data.py.....	41
7.4	Lora.py.....	43
7.5	Random_folder_generator.py.....	44
7.6	Main.py.....	44
7.7	Heartbeat.py.....	46
	Conclusion .....	48
	Appendixes .....	49
	Appendix A: System Diagram.....	49
	Appendix B: Create Global Variables Function Node .....	49
	Appendix C: Translator Function Node .....	49
	Appendix D: Counter Function Node .....	50
	Appendix E: Get Daily Counters Function Node.....	51
	Appendix F: Get Total Counters Function Node.....	51
	Appendix G: Reset Daily Counters Function Node .....	51
	Appendix H: Set Data Function Node .....	52
	Appendix I: Filename Generator Function Node .....	52

## List of Figures

Figure 1: situation of Linnosaari island [2] .....	9
Figure 2: overview of IP ratings [3] .....	10
Figure 3: the polycarbonate enclosure used in this project [4] .....	10
Figure 4: a greatly simplified CNN [7] .....	11
Figure 5: K-fold cross validation with five folds [16] .....	15
Figure 6: example annotation file created by LabelImg.....	15
Figure 7: manually traced annotations the model chose to self-evaluate .....	16
Figure 8: the outcome of the self-evaluation .....	16
Figure 9: the normalized confusion matrix .....	17
Figure 10: the absolute confusion matrix .....	18
Figure 11: label figure showing the instances of each class .....	18
Figure 12: image of a perfectly captured boat.....	20
Figure 13: image of a false detection.....	20
Figure 14: image of a late captured boat.....	20
Figure 15: comparison of different wireless network technologies [21].....	22
Figure 16: overview of the LoRaWAN network [26] .....	23
Figure 17: relationship between Data Rate, Spreading Factor and bit rate [27] .....	24
Figure 18: the Maximum Payload size, dependant on the Data Rate [27] .....	25
Figure 19: spread of the Things Stack devices and gateways in Europe [31] .....	26
Figure 20: LoRaWAN coverage in Finland [32].....	27
Figure 21: the M5Stack LoRaWAN868 module used in this project .....	27
Figure 22: Price comparison between Raspberry Pi 4B 8 GB and NVIDIA Orin NX [39] .....	30
Figure 23: the two Raspberry Pi's used in this project.....	30
Figure 24: the Raspberry Pi High Quality camera module [42].....	31
Figure 25: the global variables flow .....	34
Figure 26: the Read MQTT flow .....	35
Figure 27: the dashboard, filled with example data .....	36
Figure 28: the resets flow.....	37
Figure 29: the log data flow .....	37
Figure 30: configuration of the File Lister node.....	38
Figure 31: configuration of the change node.....	38
Figure 32: configuration of the csv node .....	39
Figure 33: configuration of the write file node.....	39
Figure 34: the systemd service file which runs the main code automatically after boot.....	40
Figure 35: the crontab of the Raspberry Pi running the machine vision code .....	41



## List of Code Fragments

Code Fragment 1: imported libraries for send_data.py .....	41
Code Fragment 2: open_serial function of send_data.py.....	41
Code Fragment 3: function wait_response from send_data.py .....	42
Code Fragment 4: send_message function from send_data.py .....	43
Code Fragment 5: encode_msg function from send_data.py .....	43
Code Fragment 6: imported libraries for lora.py .....	43
Code Fragment 7: send_command function from lora.py.....	43
Code Fragment 8: imported library for random_folder_generator.py.....	44
Code Fragment 9: get_random_bytes function from random_folder_generator.py.....	44
Code Fragment 10: create_directory function of random_folder_generator.py .....	44
Code Fragment 11: imported libraries for main.py .....	45
Code Fragment 12: count_values function of main.py.....	45
Code Fragment 13: the main event loop of main.py .....	46
Code Fragment 14: imported library for heartbeat.py .....	46
Code Fragment 15: the working of heartbeat.py .....	47

## List of Tables

Table 1: overview of most popular Finnish boat manufacturers.....	13
Table 2: link between real world objects and classes in the model.....	14
Table 3: overview of test results .....	19

## List of abbreviations

Abbreviation	Meaning
<b>HAMK</b>	Hämeen ammattikorkeakoulu
<b>SBC</b>	Single Board Computer
<b>ML</b>	Machine Learning
<b>CNN</b>	Convolutional Neural Network
<b>AI</b>	Artificial Intelligence
<b>GPU</b>	Graphical Processing Unit
<b>TPU</b>	Tensor Processing Unit
<b>LoRa</b>	Long Range
<b>LoRaWAN</b>	Long Range Wide Area Network
<b>CSS</b>	Chirp Spread Spectrum
<b>ISM (radio band)</b>	Industrial, scientific, and medical (radio band)
<b>SNR</b>	Signal-to-noise ratio
<b>OSI</b>	Open systems interconnection
<b>MAC</b>	Media access control
<b>LPWAN</b>	Low Power Wide Area Network
<b>ADR</b>	Adaptive Data Rate
<b>HTTP</b>	Hypertext Transfer Protocol
<b>UART</b>	Universal Asynchronous Receiver-Transmitter
<b>AT (command)</b>	ATtention command
<b>SMA</b>	SubMiniature version A
<b>GUI</b>	Graphical User Interface
<b>YOLO</b>	You Only Look Once
<b>MQTT</b>	Message Queuing Telemetry Transport
<b>CSV</b>	Comma-Separated Values

## Introduction

This project is conducted between Odissee University of Applied Sciences, HAMK University of Applied Sciences and the city council collaborative in Valkeakoski, focusing on the implementation of machine vision technology for boat classification and tourism monitoring.

Häme University of Applied Sciences (HAMK) is a university situated in the south of Finland. The university has several campuses, including one in the city of Valkeakoski. This campus in Valkeakoski offers a program in electrical and automation engineering. Students gather experience in a plethora of modern technologies, where they learn about electronics, power systems, automation electronics and various electrical systems. The project presented in this paper is conducted within the frame of these studies.

The primary goal of the project is to estimate the flow of water traffic during events held in the touristic season in the city of Valkeakoski. This city is situated at the meeting point of the lakes Mallasvesi and Vanajavesi, two significant lakes in Finland, ranking among the top 50 largest lakes of the country [1]. During the summer season, tourists use these lakes for traveling to different cities. The project aims to accurately identify the various types of boats and estimate the number of tourists on each vessel. The images used in the project are processed completely locally and offline, to protect the privacy of any recorded people. Pictures are saved to a local solid-state disk (SSD), to increase the accuracy of the model in a future revision. The parts used to make the project are readily available to keep costs to an absolute minimum. In summary, the project aims to be cost efficient, simple to implement, easy to transport and easy to expand.

Real-time boat count data is shown on a self-hosted dashboard, relying on Node-RED. This data is also logged daily, to offer insight into daily boat traffic and to provide backups in the case of malfunctions.

The first part of this paper shows the placement of the project, which plays a vital role in its success, so it gets its own dedicated chapter. Next, the machine vision component of the project is shown in depth, starting from training the model to validation during training and ending with the real-life evaluation or performance of the model on a test. Third, the working principles of LoRaWAN and the LoRa modulation technique are explained, as they are implemented for offline, long range and wireless communication in this project. Fourth, the motivation for implementing Raspberry Pi minicomputers is explained. Then the story moves on to the camera module used in this project. Without the right camera, machine vision would not be possible. Next, the document outlines the principles of Node-RED and creating a dashboard from a flow. The data logging system is also demonstrated. Lastly, the main structure of the code in this project is discussed thoroughly, so future maintainers of the code have a proper understanding of its foundation. To give the reader a visual representation of how all the components work together, a system diagram can be found as Appendix A in the appendixes of this paper.

## 1 Placement of the Project

The placement of the camera is a critical aspect of this project. Without a suitable location, the performance of the model would be drastically lower.

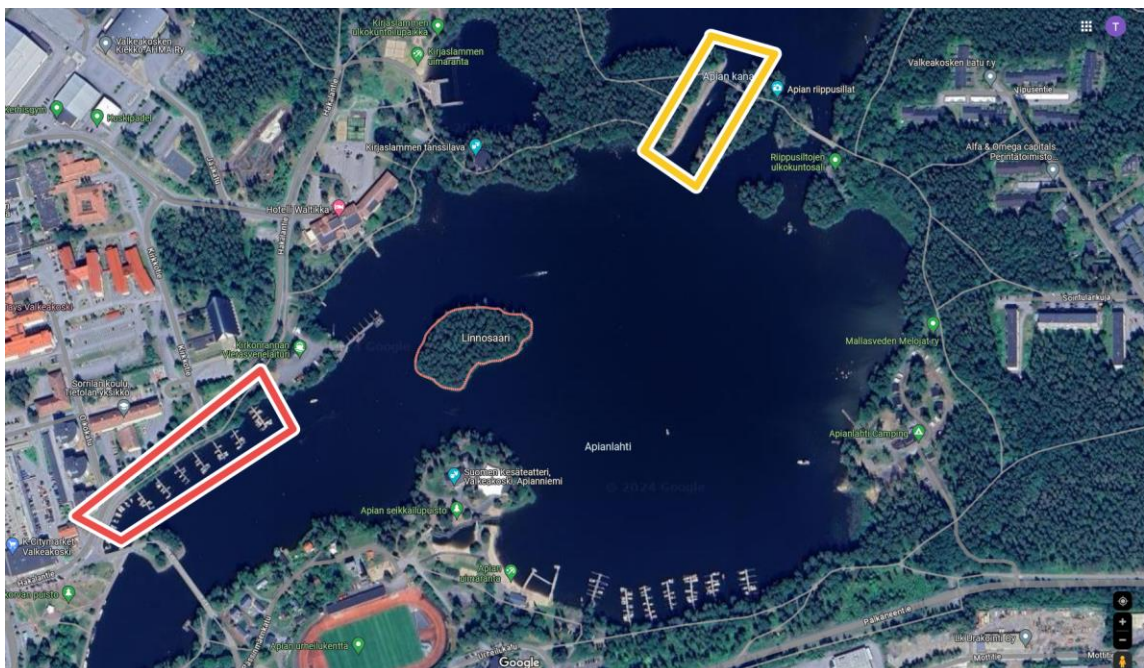
There are several natural factors to be considered: the movement of the water, the angle of the (refracted) sunlight and the power of the wind. Apart from natural factors, there are also human factors: theft and vandalism. Furthermore, the training approach for the model forces the camera to have a certain perspective. The model is trained to detect boats from a lateral viewpoint. Concrete there is need for a sheltered spot from both humans and natural elements, where boats can be viewed from a lateral perspective.

After consideration, a big rock at the northern side of Linnosaari island was selected to install the camera. To shelter the project for the natural elements, a plastic water-resistant box with a transparent lid was chosen.

### 1.1 Linnosaari Island

Linnosaari is a small island located in the city center of Valkeakoski [2]. Every boat leaving or entering the city via the Apia suspension bridges must navigate closely along the northern side of this island. There are boat docks to the south of the island, however, these docks are rented out by the city, so data on boats there is already available. The free docks are situated on the north-west side of the island, as highlighted in orange in Figure 1. The Apia suspension bridges are highlighted in yellow, and the Linnosaari island is bordered in red.

The choice to place the camera on the island is clear: the location is remote enough that it cannot be accessed on foot, and boats can be viewed from a lateral perspective.



*Figure 1: situation of Linnosaari island [2]*

### 1.2 Enclosure

To shelter the hardware of the project from the elements, a box is selected with IP67 rating. IP stands for "Ingress Protection". It is a rating developed by the IEC to grade the resistance of an enclosure against water or dust intrusion [3]. An overview of the rating can be seen in Figure 2. The IP67 rated is essential

for this project, as it ensures the installation is protected against the effects of temporary immersion in water and is fully dust-tight.

## Ingress protection (IP) ratings guide

IP ratings are represented by combining the first and second digits of the below columns




















1 <sup>st</sup> numeral - solid foreign objects			2 <sup>nd</sup> numeral - water		
0	No protection		0	No protection	
1	Protected against solid foreign objects of 50 mm Ø and greater		1	Protected against vertically falling water drops	
2	Protected against solid foreign objects of 12,5 mm Ø and greater		2	Protected against vertically falling water drops when enclosure tilted up to 15°	
3	Protected against solid foreign objects of 2,5 mm Ø and greater		3	Protected against spraying water	
4	Protected against solid foreign objects of 1,0 mm Ø and greater		4	Protected against splashing water	
5	Dust-protected		5	Protected against water jets	
6	Dust-tight		6	Protected against powerful water jets	
Example:  +  = IP 65			7	Protected against the effects of temporary immersion in water	
→ Protected against water jets			8	Protected against the effects of continuous immersion in water	
→ Dust-tight			9	Protected against high pressure and temperature water jets	

Figure 2: overview of IP ratings [3]

The box is made of polycarbonate, a plastic polymer. Choosing a plastic enclosure is motivated by its lightweight nature compared to metal or wood, making it easier to transport. Furthermore, plastic is a non-conductive material, which reduces electrical hazards. Lastly, polycarbonate can be made fully transparent, which is necessary for the camera to operate. The lid of the enclosure is fully transparent [4].

Figure 3 shows the enclosure. Note how the box is tilted 90 degrees, so the transparent lid can serve as a window for the camera lens.



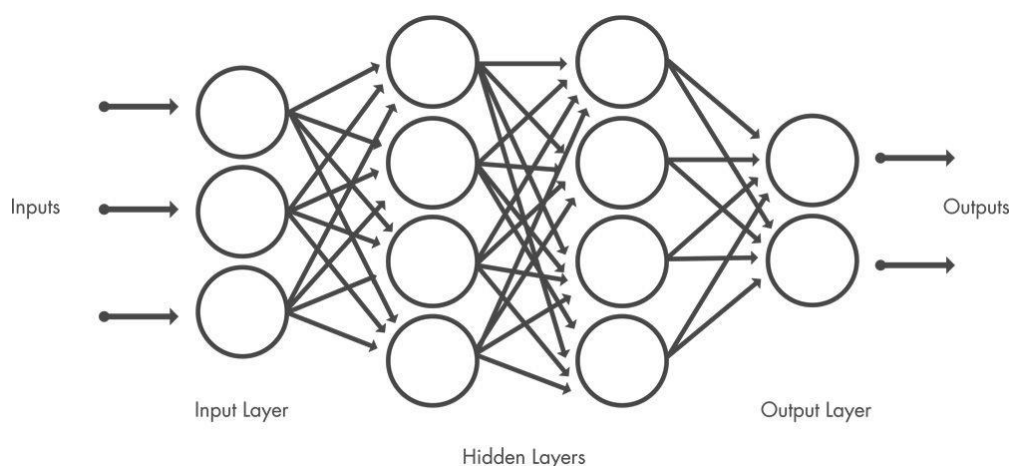
Figure 3: the polycarbonate enclosure used in this project [4]

## 2 Machine Vision Model

Machine vision is a branch of Artificial Intelligence (AI). It uses digital cameras and Machine Learning (ML) algorithms to interpret images or videos. In machine vision, the goal is to mimic the human capability to recognize patterns. To implement machine vision for a specific task, it is necessary to make a model of the task it will have to perform.

First comes the gathering of a large dataset of images. It is particularly important that these images are exemplary to the real-world scenarios the camera will encounter. Then, the shapes of features or objects the model should be able to recognize are manually traced. This step is known as the annotation of the dataset. The next step involves choosing the model architecture. This requires a lot of knowledge about the world of AI and ML. For the application of this project, the model architecture was decided on by Juhani Henttonen, an expert in data science and a Principal Lecturer in Machine Vision at HAMK. The expert's opinion was to go for the You Only Look Once (YOLO) architecture. The reasoning behind this choice is the abundance of documentation available, combined with professionally written examples. The YOLO Quickstart guide offers painless installation of the architecture [5]. Furthermore, YOLO is described as an efficient framework, balancing speed and accuracy [6].

The YOLO architecture is based on a convolutional neural network (CNN), which is a class of deep learning algorithms based on “convolutional layers”. These layers consist of filters (also known as kernels) that slide over an input image and produce so called feature maps. Then, the feature maps are used to find recognizable patterns in the existing images, so the model can search for similar patterns in new, unknown images [7]. A greatly simplified visual representation of a CNN can be found in Figure 4.



*Figure 4: a greatly simplified CNN [7]*

The convolutional layers in Figure 4 are shown as columns of circles. These circles represent the kernels, where the arrows show the flow of data through the model. Note how the data flows in only one direction. The model never goes back to compare data with previous layers. This is a defining characteristic of CNNs and the reason behind the “You Only Look Once” name.

The last step of the preparation process is training the actual model. For a reliable training process, the dataset of images gets split into two groups: training and testing. The training group is used to learn the model how to classify each desired feature and the testing group is used to verify whether the model can make accurate predictions. If necessary, there is fine-tuning possible after training. On the other hand, if the performance of the model is satisfactory, it can be deployed in the real-life application. After deployment, improvements can be made, based on recorded data.

The last part of this chapter focuses on the licensing around the YOLO architecture.

## 2.1 Data Acquisition

A machine vision model requires an enormous amount of data. The best way to gather this data would be to monitor all of the water traffic in Valkeakoski for an entire year and use the gathered data to make a model. This was not an option, due to time constraint issues. The data acquisition part of this project is undertaken during wintertime, when all boats are kept ashore and are covered up with tarps. An arbitrary goal of 1000 pictures is set, to ensure a diverse and representative sample of the objects the model should recognize. This approach ensures that the project remains within manageable time limits, enabling progress on the other parts of the project within a reasonable timeframe.

An easy alternative way of avoiding data acquisition and annotation would be to search for a premade dataset where all of the annotation is already done. Example sites for this would be Kaggle, or the Open Images Dataset from Google [8] [9]. Unfortunately, most of those readymade sets are far too complex, with an overwhelming number of classes. The majority of these classes do not apply to this model. This would make the model too complicated to run properly on a Single Board Computer (SBC), such as Raspberry Pi. The decision is made to create a new dataset rather than invest additional time in cleaning a pre-made one.

In the self-made dataset the focus is entirely on small, motorized boats from Finnish manufacturers, as these boats are the most relevant for the application of the project. An added advantage of exclusively using Finnish manufacturers is that very often, their product pictures are taken on Finnish lakes. This makes the lighting conditions and background colors of the pictures similar to those of the real-world environment of the project.

A small pre-study was conducted on Finnish boat brands in order to find the most popular manufacturers [10], as well as checking whether their boat models were suitable for the lakes surrounding Valkeakoski. Table 1 shows the outcome of that pre-study. Note that the manufacturers were ordered alphabetically and not in order of popularity.



*Table 1: overview of most popular Finnish boat manufacturers*

manufacturer	done	fit for model?	comments
AMT			S, M
aquador			no pictures downloadable on site
axopar			L
baltic yachts			too exclusive, too big, sailing yachts
bella			no pictures downloadable on site
buster			S, M
Cross			S, M
elwood			not many pictures, wood
finnmaster			no pictures downloadable on site
flipper			no pictures downloadable on site
grandezza			no pictures downloadable on site
juvi			XS, S, M
nauticat			sailing yachts, no pictures on site
nautor swan			too exclusive, too big, sailing yachts
nord star			M, L
quarken			M
sargo			M, L
saxdor yachts			M
targa			too exclusive, too big
terhi			XS, S, M
xo boats			M
yamarin			M, L

In Table 1, a color code is in effect. Green means that all models (and thus all available pictures on the website) of the brand were used for the model. Orange indicates that some, but not all models were used, and the “comments” column briefly explains why. Finally, red means that the manufacturer was not included in the dataset, again with a brief explanation in the “comments” column. XS, S, M and L stand for the relative sizes of the boats. An XS boat translates to a rowing boat or smaller, S corresponds to a small and motorized boat, M boats have some sort of interior seating compartment and L boats are similar to M boats but come with at least one additional deck.

## 2.2 Data Cleaning and Annotation

After collecting a total of 984 pictures, the annotation process can start. This means that a rectangle has to be drawn around every boat, person or duck in a picture and all of these have to be manually labeled appropriately. The chosen tool to do the annotation is Labellmg. This is an open-source tool which can create text files with the coordinates of the rectangles around each object, as well as the assigned class. All this info can be directly saved with correct formatting for YOLO. This is a step that must be taken very carefully, as a minor mistake could compromise the training. If the rectangles are not precisely drawn around each object, or are mislabeled by the annotator, the model will become less accurate.

There are 7 custom classes created: person, small boat, medium boat, paddle boat, ferry, sauna and duck. The classes are listed in Table 2.

*Table 2: link between real world objects and classes in the model*

size	description	name in model
XS	paddle boats	paddle boat
S	2-4p motor boats	small boat
M	4p-8p motor boats	medium boat
L	>8p motor boats	ferry
ferry	ferry boat of Pirkka Lines	ferry
sauna	Saunalautte in Valkeakoski and hotel Waltikka	sauna
duck	any water animal	duck

The decision is made to opt for these classes on the basis of the following reasoning:

- Paddle boat: Chosen because there are options to rent canoes and SUP-boards from various businesses in Valkeakoski [11]. While this requires the tourists to be present in Valkeakoski when they enter the water, it might be interesting to track where tourists go with rented paddle boats. The rental data could be compared to the number of counted paddle boats to keep track of how many people bring their own kayak, canoe or SUP-board to the city.
- Small boat: Selected due to their presence on the lakes.
- Medium boat: Similarly selected due to their presence on the lakes.
- Ferry: Specifically made for the Pirkka Lines ferry that passes through the city regularly during the summer season [12].
- Sauna: Chosen to avoid false detections caused by the different sauna boats in the city [13].
- Duck: Selected to avoid false detections caused by various water animals in the lakes, such as ducks and geese.

The size column of Table 2 relates to the sizes mentioned in Table 1.

The description column of the table shows what the criterium for each class is. It is important to note that the boats are solely divided into multiple categories as a failsafe method. In an ideal case, all the boats would be defined as a single category, to save on computational power and let the model primarily focus on counting the people aboard. On the other hand, larger boats frequently have more seating inside. Often, boats contain all of the main controls within the ship. As a camera would not be able to detect anyone aboard, the size of the vessel can be used as a tool to estimate that crude number.

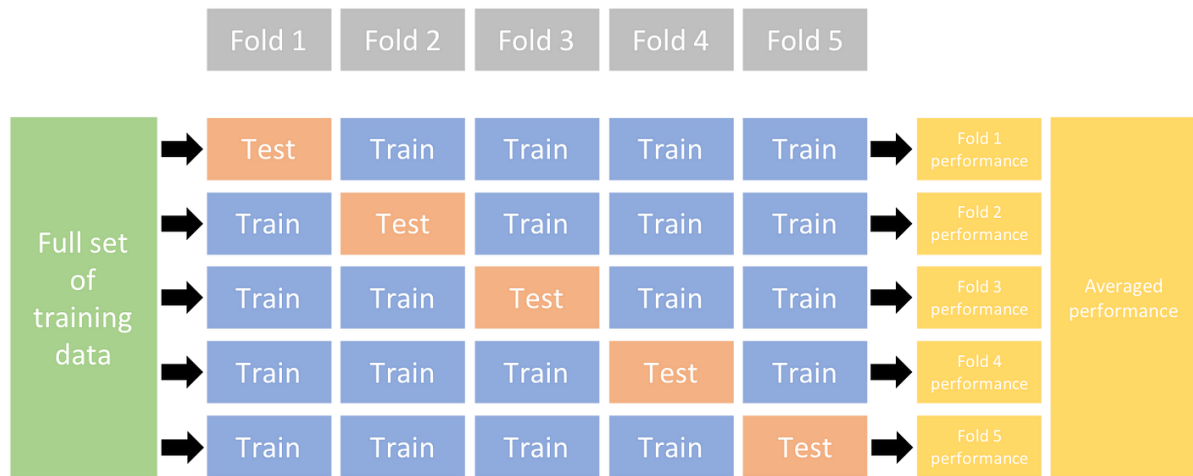
Additionally, privacy is a factor that needs to be considered. This project will run for an entire summer and save every recording of boats, to create a new, optimized version of the model in the future. People are allowed to be recorded, as this project falls under scientific or historical research in the "Public interest and authority" processing basis [14], but this project strives to keep faces unrecognizable. To avoid putting focus on the people, the model is designed to fall back to the classification of the boat types instead.

## 2.3 Training of the Model

The training of the model is conducted in Google Colab, to have greater access to more powerful Graphical Processing Units (GPUs) and Tensor Processing Units (TPUs), which significantly reduce training time. Even with a Google L4 GPU, the training took approximately four hours to complete. Through the power of cloud computing, there is little to no concern about the physical limitations of a computer, like RAM or disk space.

The training is done according to the official documentation of Ultralytics [15], the developing team behind the YOLO deep learning algorithm. Due to the dataset being relatively small for the number of constructed classes, the preferred method is K-fold cross validation. The choice is made to construct 3 folds with 70 epochs of training time each. Implementing this method of training and validation is made

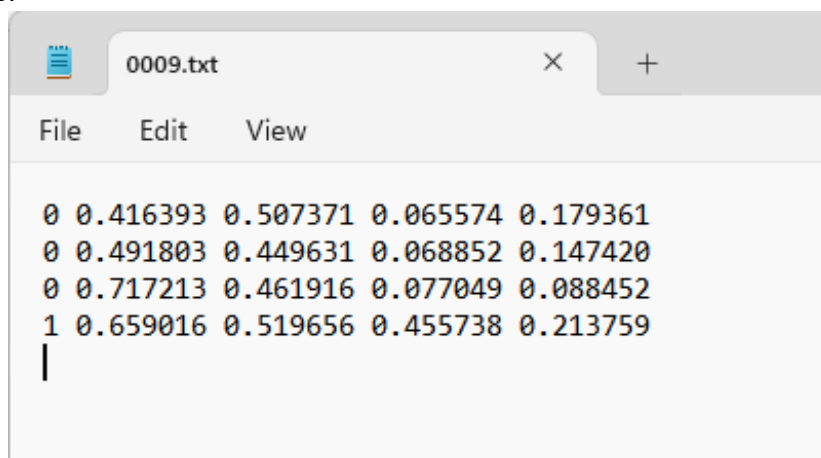
remarkably simple, as Ultralytics documentation provides a template for K-fold cross validation [16]. This documentation outlines a comprehensive overview of what exactly K-fold cross validation entails. Furthermore, this concept not only applies to machine vision, but to every branch of data science.



*Figure 5: K-fold cross validation with five folds [16]*

The documentation accurately explains how the folds are created and what happens during each step in the training process. As shown in Figure 5, the dataset is split into multiple training groups and one test group for each fold. After training on the training groups, the model will validate itself by using the unseen test set of data. This is repeated for a chosen number of epochs per fold and for a chosen number of folds. The figure shows a visual representation of five folds, this project implements three folds.

For full transparency in the training process, the used Google Colab template can be found on the GitHub repository for this project [17]. Similarly, the "google\_colab.yaml" file, which contains a dictionary of the classes and the pathways to the directories of the data, can be found there as well. The data consists of both the annotated images and their assigned labels. YOLO requires these labels to be stored as a text file, consisting of lines where each line shows a number, followed by four floating point numbers (also known as normalized xywh format). The first number stands for the class. The following two floating point numbers are the x and y coordinates of the center of the rectangle that is drawn around the object. The last two floating point numbers represent the normalized width and height respectively of this rectangle. All these coordinates are normalized between 0 and 1 [18]. An example can be found in Figure 6.



*Figure 6: example annotation file created by LabelImg*

As earlier explained in Chapter 2.2, Labellmg created these text files during the annotation process. The template then uses these text files to construct a Pandas dataframe, where each row consists of one label file and the columns show a count for each of the classes represented in that label file. Once the dataframe is constructed, the template uses KFold from the Python sklearn.model\_selection library to form the folds in the dataset and split the data in different training and validation sets. Every fold gets their own folder with a training and validation set. The template automatically creates a YAML file for each fold, as they now have become a separate dataset to train the model from. The template then copies and pastes every image over to the new folders, along with their label files. Lastly, the template trains a model from each fold, making use of the Ultralytics “model.train” function. After the separate training is done, the models are converted into one file called “best.pt”.

## 2.4 Validation of the Model

Due to the K-fold cross validation method used, the model does self-evaluation on unseen data after training each separate fold. Once the training is finished, an overview of these evaluations is saved to a separate folder and file path called “/results/detect/train”. The most interesting data extracted from this self-evaluation is shown in Figure 7 and Figure 8. Both figures are to be used together, as they oversee the same source images. The bounding boxes and classes displayed on the images in Figure 7 are made by the annotator, while in Figure 8, the outcome of the model’s prediction is displayed.

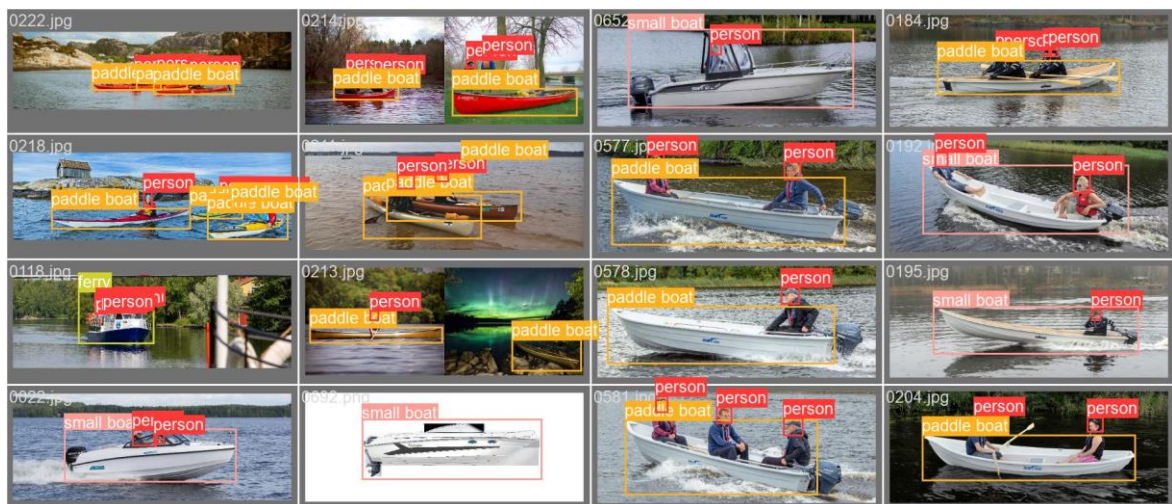


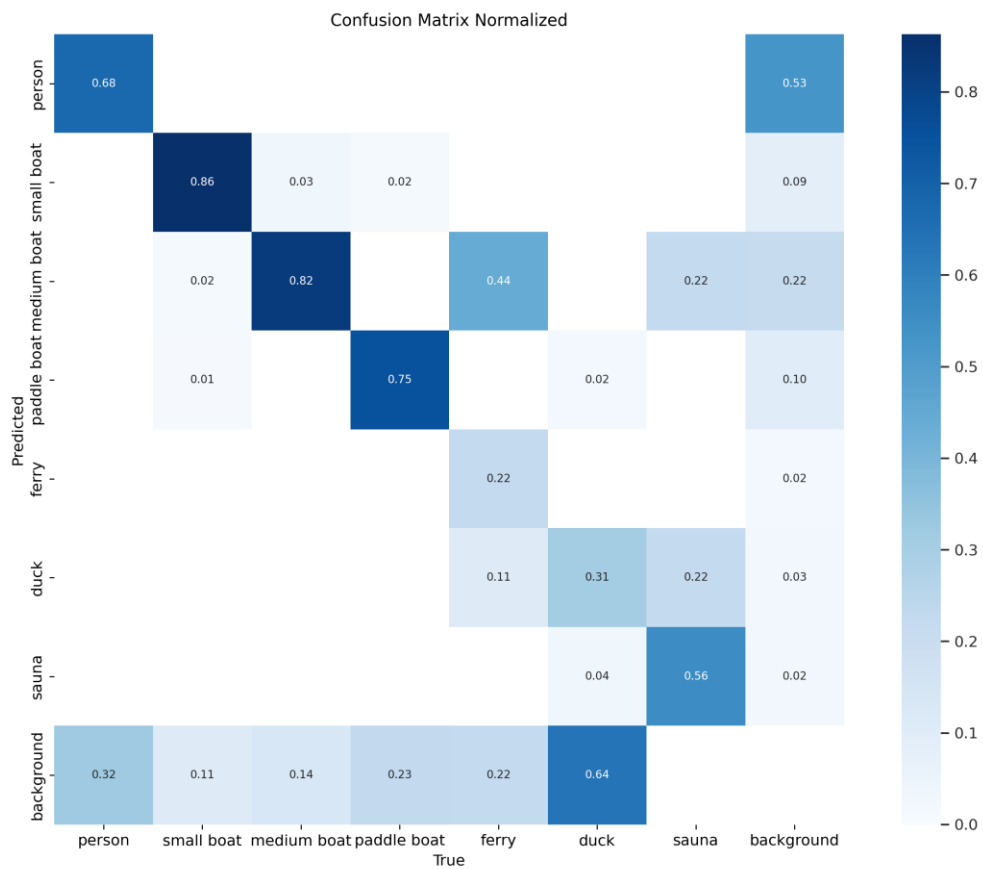
Figure 7: manually traced annotations the model chose to self-evaluate



Figure 8: the outcome of the self-evaluation

What can be derived from these pictures is that the model effectively detects the presence of people. The model can also accurately predict small boats and paddle boats, but it mixes some of them. An example picture where this happens is "o192.jpg" in Figure 7 and Figure 8. It must be noted that the boats where the mix-up occurs are edge cases themselves, because they have a similar shape and construction in comparison to a 'paddle boat'. The only exception is that the former are equipped with a motor. In the annotation phase, the decision was made to classify these vessels as "small boats".

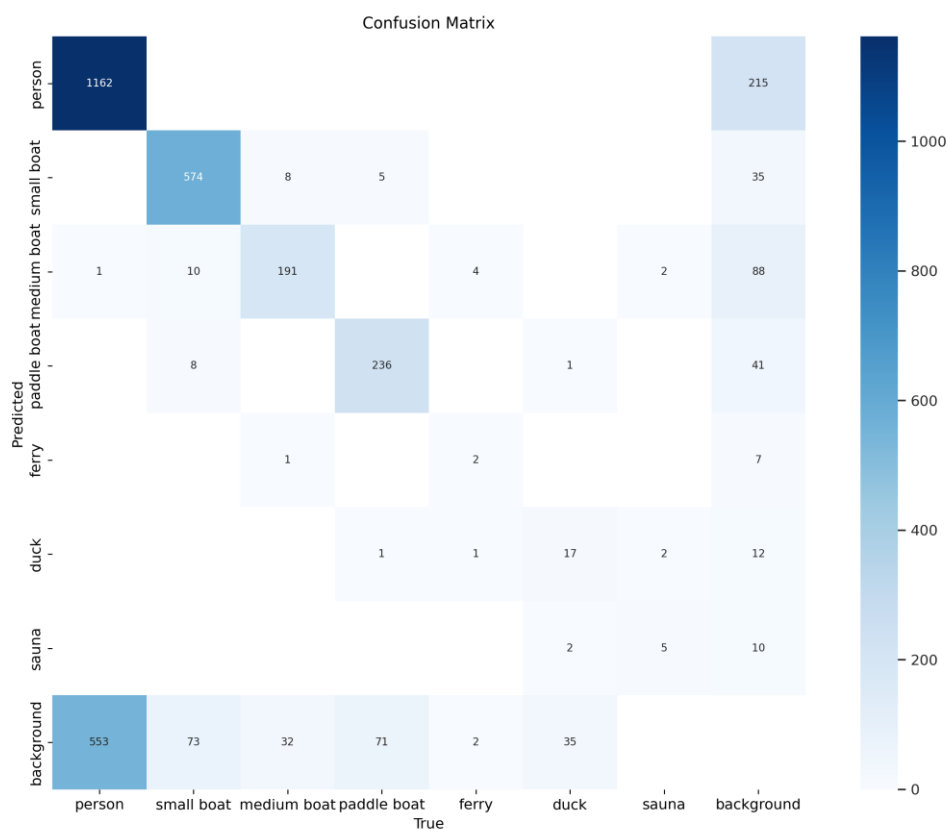
Furthermore, it becomes apparent that the model finds difficulty with accurately detecting 'medium boats', as is showcased by picture "oo22.jpg" in the figures above. The prediction score is below 0.5, meaning the model is less than 50 % convinced it classified the vessel correctly. Picture "oo22.jpg" is also a misclassified example, where a small boat gets recognized by the model as a medium boat. The confusion matrices show more insight in the mislabeling, displayed in Figure 9 and Figure 10.



**Figure 9: the normalized confusion matrix**

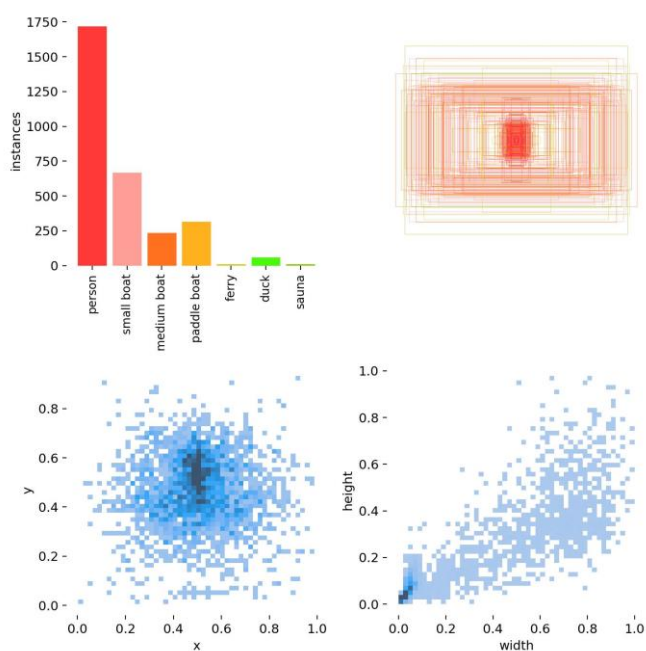
From the normalized confusion matrix, shown in Figure 9, it becomes clear that the model holds flaws. The most important metric for the project in this stage is the 'background'. This metric shows whether something was detected when it should not have been, and the other way around. In the first place, the model should be able to detect the presence of boats. At this stage of the project, it is somewhat acceptable that boats get mislabeled, as long as they are detected.





**Figure 10: the absolute confusion matrix**

From the absolute confusion matrix, as shown in Figure 10, another property becomes clear: the spread of data is uneven. The label figure, generated by the model after training, and shown as Figure 11, confirms this as well.



**Figure 11: label figure showing the instances of each class**

The number of instances of the people class is much higher than any other class. This might explain why the model performs so well in recognizing people on the validation pictures. More importantly,

there is a significant difference in the number of instances for each boat class. This explains why there is mislabeling happening between the different boat classes. The model favors the more represented class of small boat over the class of medium boat.

It is also apparent that the instances of the ferry, duck and sauna classes are far too infrequent to serve any purpose in the current version of the model. In a future revision, more time could be dedicated towards a more even spread of instances among all classes, to achieve a more robust and precise model.

It must be noted that in the real-world application of this project, there is no even spread of classes. There are significantly more small boats than medium boats on the lakes around Valkeakoski. In the annotation phase, the decision was made to favor small boats over medium boats and make them more represented in the model.

## 2.5 Testing and Evaluation of the Model

The testing of the model was conducted on Sunday, May 5, 2024. In total, there were 18 boats spotted that day. The choice was made to not evaluate the classes of "person", "ferry", "duck" or "sauna". Instead, only the classes "small boat", "medium boat" and "paddle boat" were observed. The results of the test are listed in Table 3.

*Table 3: overview of test results*

Class	Observed	Detected	Missed detections	False detections	Wrong detections
Small boat	17	11	2	0	0
Medium boat	1	8	1	8	0
Paddle boat	0	4	0	0	4

In Table 3, the total number of observed boats are listed in the column "Observed". The column "Detected" shows which classes the model detected. The "Missed detections" lists the boats that were observed, but not detected by the model. "False detections" shows detections that were done while no boats were present and "Wrong detections" shows the amount of detections that were done while there was a boat present, but the classification went wrong. For example, if there was a "small boat" present but the model classified it as "paddle boat", the count of "paddle boat" in the "Wrong detections" is incremented. If the "small boat" is not detected at all by the model, the count of "small boat" in the "Missed detections" column is incremented.

The outcome of the test shows two notable results:

1. Small boats are easily recognizable. However, the model often classifies small boats as paddle boats. The two missed detections were done on one specific boat that passed twice. As the boat was traveling at an elevated speed, the missed detections can be attributed to this factor. The boat presumably ignored the nine km/h speed limit which is in effect in the city center of Valkeakoski [19].
2. The model is quite inaccurate in recognizing medium boats. There was one medium boat present during the day, but the model did not recognize it. Furthermore, the model detected eight medium boats without any being present. However, it must be noted that these detections happened in rapid succession of each other, on two separate times of the day. The camera was being triggered by specific movements in the water. This trigger might be avoidable by placing the camera higher up and under a steeper angle, to avoid refraction of the light off the water.

Another notable outcome of the test is that the one medium boat filled more than the entire field of view of the camera. A possible explanation for the failed detection could simply be that the boat was too big to be detected. Lastly, the captured images were of high quality, but unfortunately not very reliable. Sometimes, boats are captured in the middle of the frame, but often there is only the back half of the boat visible in the frame.



*Figure 12: image of a perfectly captured boat*



*Figure 13: image of a false detection*



*Figure 14: image of a late captured boat*

A possible solution for these late captured boats, like the one shown in Figure 14, is to save the original frame where the detection happens on. The downside to this method is that the saved picture has a resolution of 640 px by 480 px instead of 4056 px by 3040 px.



## 2.6 YOLO and licensing

After the training and testing is done, the model is ready for deployment. However, there are licenses to consider when using YOLO.

The YOLO architecture is free to use but falls under the AGPL-3.0 license [20].

The AGPL-3.0 license, or Affero General Public License version 3.0, is a copyleft license which grants users the freedom to use, change, and distribute software, provided that any modifications or additions made to the software are also released under the same license terms. Additionally, the AGPL-3.0 requires that if the software is changed and used over a network, then the modified source code must be made available to the users who interact with the modified software over that network.


Under the terms of the AGPL-3.0 license, this project is subject to open-sourcing, meaning that all code modifications and additions made to the project must be shared with others under the same license terms. Readers interested in accessing the code base for this project can find it on the GitHub platform [17].

If the city wishes to keep the source code private, the developers of Ultralytics provide an Enterprise licensing option [20].

### 3 LoRaWAN

This project relies heavily on LoRaWAN technology for sending data to the dashboard. This protocol is based on the LoRa modulation technique. The main advantages LoRaWAN has over other wireless protocols are: low cost hardware, long distances that signals can travel, and low power consumption. The main disadvantage for LoRa is the limited data rate that can be sent in one message.

As shown in Figure 15, compared to other wireless technologies for Internet of Things (IoT) applications LoRa seems like the perfect solution for long range, low cost and low data rate applications [21]. This fits perfectly in the scenario of this project, where the placement is remote and only small messages must be sent. It considers future revisions of the project, where the module could be run off of batteries or renewable energy sources.

<u>Traditional Cellular</u>  Long Range High Data Rates Low Battery Life High Cost	<div> <div>LPWAN (3-5B in 2022)</div> <div>  <div> Long Range  Low Data Rates  Long Battery Life  Low Cost  High Capacity Potential </div> </div> </div>	<u>Cat-M1</u>  Long Range High Data Rates Low Battery Life Medium Cost
<u>Local Area Network</u> (Wi-Fi)  Short Range High Data Rates Low Battery Life Medium Cost	<u>Narrow-Band IoT</u> (NB-IoT)  Stationary Devices Short Range (indoor coverage) Low Data Rates Good Battery Life Low Cost	<u>Personal Area Network</u> (Bluetooth®)  Very Short Range Low data rates Good Battery Life Low Cost

*Figure 15: comparison of different wireless network technologies [21]*

However, it is important to consider that wrong usage of wireless protocols like LoRaWAN are potentially punishable by Traficom, the Finnish radio spectrum regulator [22]. That is why in this chapter, there will be focus on the specific regional parameters needed for LoRaWAN certification in its relevant frequency band for Finland.

#### 3.1 Terminology behind LoRa and LoRaWAN architecture

LoRa is a modulation technique used for wireless communication. The technology is proprietary and belongs to Semtech Corporation, a global leading supplier of high-performance semiconductors from the United States. The modulation technique is achieved by implementing Chirp Spread Spectrum (CSS) modulation. This means the data is encoded in continuous waveforms that change frequency over time. These waveforms are called chirps. It is extremely power efficient, as it only requires short bursts of waves at specific frequencies. These frequencies are chosen in such a way that they are spread out over the entire frequency band for the protocol.

LoRa can be used as a standalone technology to create point-to-point communication. It can also be configured as a star network, through a centralized gateway. The ideal application for LoRa is remote sensor monitoring. It is mostly used for a small number of end devices. A problem arises when the networks get bigger and there are multiple gateways needed. The solution to this problem is formed as LoRaWAN.

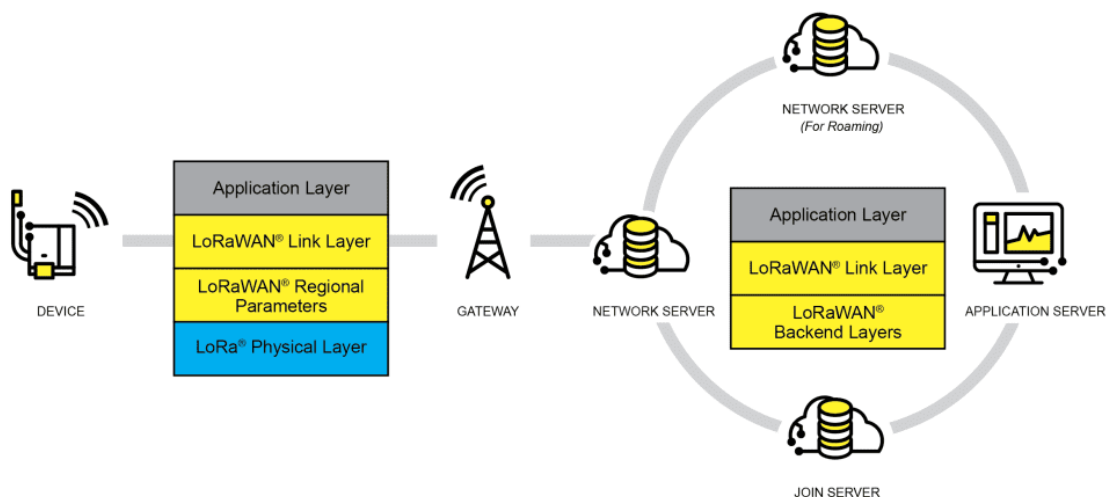
LoRaWAN is a protocol built on top of the LoRa modulation technique. It is a set of rules defined by the LoRa Alliance, an open and nonprofit association. It implements both the data link layer and the network

layer, also known as layer two and layer three respectively in the OSI network model. It combines both of these layers in a so-called Media Access Control (MAC) layer.

This protocol gives the opportunity to create exceptionally large networks of end devices and gateways. Because the end devices can use LoRa modulation technology, a Low Power Wide Area Network (LPWAN) is formed. This type of network is ideal for IoT solutions because it does not use up any capacity of the existing cellular networks, while staying relatively close to the same working principle and keeping power costs low. Without the LoRaWAN protocol, it would not be possible to create a network and scale it up.

It is interesting to note that LoRaWAN is an open standard and is a part of the Internet, Science and Medical (ISM) frequency band, a particularly important band for the world of IoT, because all the frequency ranges within this band are globally available for unlicensed use [23]. This makes LoRaWAN communication free to use for anybody who owns a LoRa radio transceiver. Other notable wireless communication protocols from within the ISM band include Wi-Fi and Bluetooth in the 2,4 GHz to 2,4835 GHz range [24].

A typical LoRaWAN architecture exists of at least five components, which are laid out in a star-of-stars topology, visually represented in Figure 16 [25] [26].



*Figure 16: overview of the LoRaWAN network [26]*

The end devices in the network are sensors or actuators. They can send data back and forth between themselves and gateways, using the LoRa modulation technique. The gateways receive messages from the end nodes and pass them on to the Network Server. These gateways use Ethernet, Wi-Fi or even cellular networks. The Network Server is a piece of software which is used for managing the entire network. A well-known example of a Network Server is the Things Stack [26]. The Join Server is also a piece of software, running on a server. It is only used to process join-request messages sent by end devices. The Application Server is a piece of software, responsible for securely processing application data. Typical examples of application servers include Message Queuing Telemetry Transport (MQTT) servers, HTTP web hooks or Amazon Web Services (AWS). Multiple application servers can be active at the same time.

### 3.2 Regional Parameters for EU863-870 MHz Band

The chirps that are generated by the LoRa modulation happen in specific frequency zones. These frequencies are chosen by the LoRa Alliance in such a way that they are spread out over the entire frequency band for the protocol. In Europe, this band is either 433 MHz (EU433) or 863 to 870 MHz

(EU863-870). Finland falls under the EU863-870 MHz band [27]. Some parameters around LoRaWAN are region bound. It is important to know about these parameters before implementing LoRaWAN, because if they are wrongly initialized it would be impossible to join a LoRaWAN network. Incorrect usage of LoRaWAN could create trouble for other wireless protocols in the ISM band and could possibly have legal consequences. In Finland, the radio spectrum is closely monitored by the Transport and Communication Agency Traficom [28].

### 3.2.1 Channels

This frequency band gets divided into smaller bands, called channels. For Finland and other countries in the EU863-870 band, the rules laid out by the LoRaWAN protocol state that there should be at least 3 default channels provided by the end device. The width for each of these channels is 125 kHz. This width is called the bandwidth of the channel. These channels are used specifically by the end devices to broadcast a Join-Request message to the Join Server [27]. There can be other channels with wider bandwidth implemented, which allow for the sending and receiving of higher data rates, at the cost of a lower signal-to-noise ratio (SNR). A lower SNR means that the signal is less robust against interference and noise.

### 3.2.2 Spreading Factor and Coding Rate

The data signal gets multiplied with a spreading code. The amount of spreading code is an indicator for the bit rate and range. A higher amount of spreading code means that the bit rate goes up, but the range goes down. The LoRa protocol expresses the different amounts of spreading code as a spreading factor (SF). In the current form of the technology, there are six different spreading factors defined. They are numbered by starting at SF7 and moving up until SF12. The higher the SF number, the more spreading code is added to the signal, thus the higher the bitrate is [27].

### 3.2.3 Data Rate

The data rate is the number of bits that are transmitted per unit of time. This number is dependent on the spreading factor, bandwidth of the channel and the coding rate. They are linked to each other in Figure 17.

Data Rate	Configuration	Indicative physical bit rate [bit/s]
0	LoRa: SF12 / 125 kHz	250
1	LoRa: SF11 / 125 kHz	440
2	LoRa: SF10 / 125 kHz	980
3	LoRa: SF9 / 125 kHz	1760
4	LoRa: SF8 / 125 kHz	3125
5	LoRa: SF7 / 125 kHz	5470
6	LoRa: SF7 / 250 kHz	11000
7	FSK: 50 kbps	50000
8	LR-FHSS <sup>13</sup> CR1/3: 137kHz OCW <sup>14</sup>	162
9	LR-FHSS CR2/3: 137kHz OCW	325
10	LR-FHSS CR1/3: 336kHz OCW	162
11	LR-FHSS CR2/3: 336kHz OCW	325
12..14	RFU	
15	Defined in [TS001] <sup>15</sup>	

*Figure 17: relationship between Data Rate, Spreading Factor and bit rate [27]*

According to the LoRa Alliance, not all of these Data Rates should be implemented, but to receive certification at least DR0 to DR5 should be supported by the end device [27].

### 3.2.4 Maximum Transmission Power

The maximum transmission power for LoRaWAN is expressed as the Effective Isotropic Radiated Power (EIRP), which is the total power radiated by an isotropic antenna in a single direction. For EU863-870, the maximum EIRP is +16 dBm [27].

### 3.2.5 Maximum Payload Size

The maximum payload size length (M) of a message is dependent on the Data Rate and is defined in Figure 18. It also lists the maximum payload size length if the optional control field FOpt is absent (N).

Data Rate	M	N
0	59	51
1	59	51
2	59	51
3	123	115
4	230	222
5	230	222
6	230	222
7	230	222
8	58	50
9	123	115
10	58	50
11	123	115
12-15	Not defined	

Figure 18: the Maximum Payload size, dependant on the Data Rate [27]

### 3.2.6 Duty Cycle

According to the LoRa Alliance, a device should only transmit 1% of the time at the maximum EIRP of +16 dBm [27]. The duty cycle is limited to ensure that the device cannot cause interference to other devices using the ISM band. To calculate how much Time on Air is allowed per day, following formula can be used:

$$\text{Time on Air} = \text{amount of seconds in a day} \times \text{duty cycle \%}$$

For the maximum duty cycle of 1% (at maximum EIRP), the Time on Air becomes:

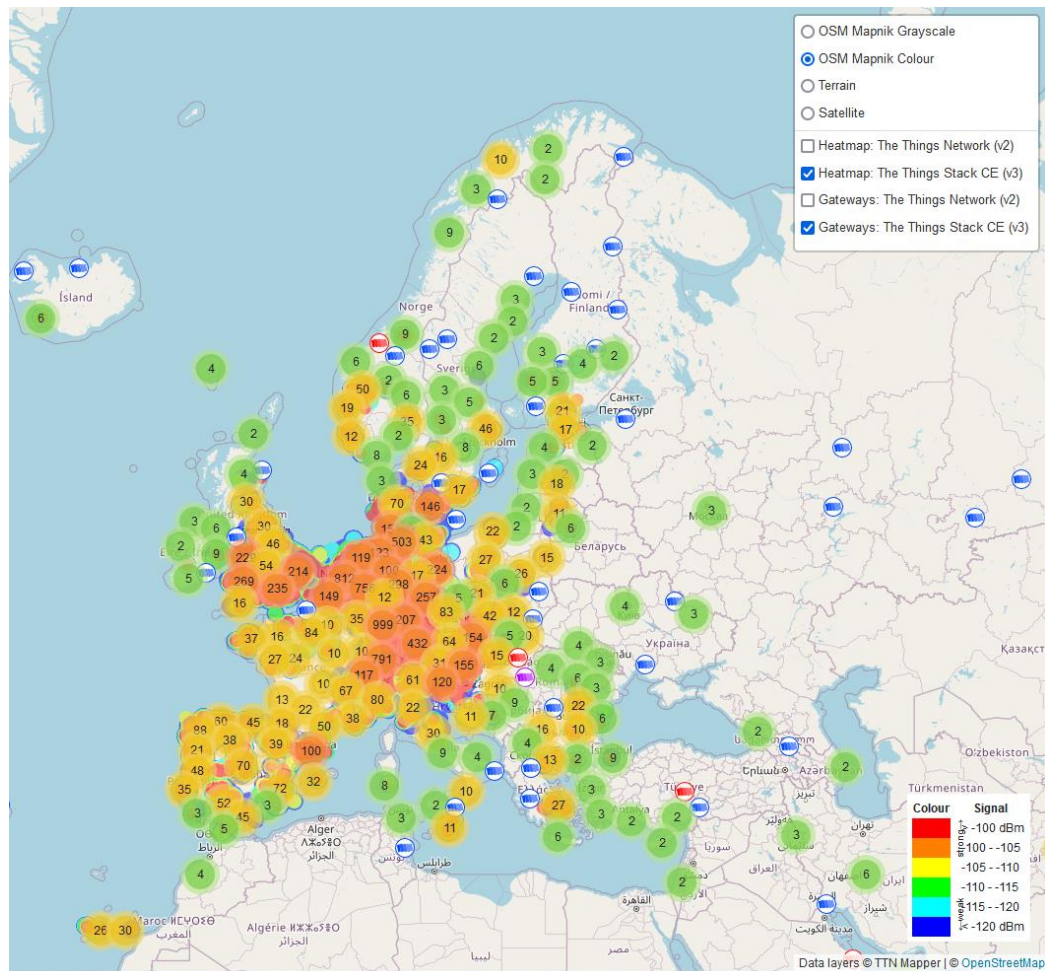
$$\text{Time on Air} = \left( 24 \frac{\text{hours}}{\text{day}} \times 60 \frac{\text{minutes}}{\text{hour}} \times 60 \frac{\text{seconds}}{\text{minute}} \right) \times \frac{1}{100} = 864 \text{ seconds}$$

This means that a LoRaWAN end device in Europe can transmit for up to 864 seconds per day at maximum EIRP. Assuming the module used in this project takes one second to transmit a message, it would be allowed to send 864 messages at maximal transmission power each day.

## 3.3 LoRaWAN in Finland

The LoRaWAN alliance looks over the correct implementation of the protocol and tries to promote it all over the world. Implementation of the protocol requires heavy investments. Not only the necessary hardware is needed to create the network, but a lot of software must be written as well. As explained in Chapter 3.2, not all places in the world can transmit LoRa signals in the same way. In Finland, the implementation of the LoRaWAN protocol is carried out by Digita Oy, a Finnish telecom company, specialized in broadcasting radio, television and telecom. In 2013, they were the ninth biggest audiovisual company of Finland, based on revenue [29].

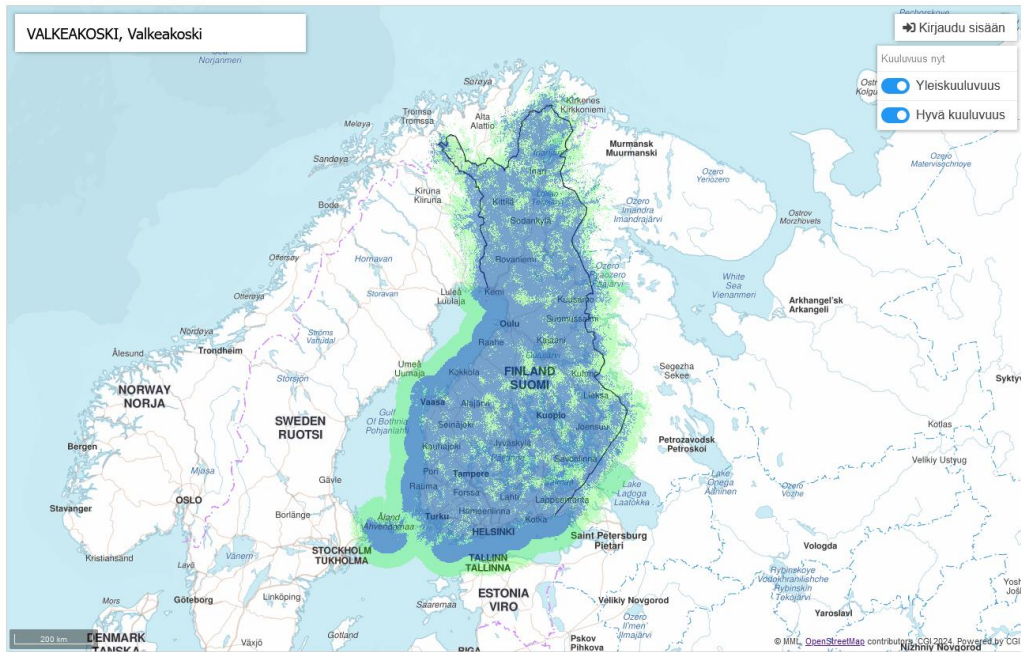
Digita Oy cooperates with the Things Industries, where the Things Industries provide the necessary software (called the Things Stack). In return, Digita Oy provides the hardware and monitoring. This collaboration brings great benefits to both parties. The Things Industries has immediate professional coverage of almost all of Finland's surface, while Digita Oy can provide an industrial grade IoT network all over Finland. Because Digita Oy also has roaming rights in the European Union, a company would be able to reach end devices all over Europe through this network [30]. Figure 19 shows the spread of the Things Stack on mainland Europe, while Figure 20 represents an estimated coverage map of the LoRaWAN network in Finland [31].



**Figure 19: spread of the Things Stack devices and gateways in Europe [31]**

The light blue colors show the theoretical coverage of LoRaWAN, the dark blue shows places with good coverage. According to Digita, the light blue boundaries of the network coverage area have been calculated based on a LoRa spreading factor (SF) 12 and a terminal location 1,5 m above ground level [32].





*Figure 20: LoRaWAN coverage in Finland [32]*

This project uses the LoRaWAN network from Digita Oy free of charge, because HAMK as an educational institution received free access to the necessary hardware and software. In return, the institution implements LoRaWAN in their curriculum.

### 3.4 Project Implementation: LoRaWAN Module

As explained earlier in this chapter, it is very important to take the regional parameters into account when working with LoRaWAN, in order to avoid legal persecution [28]. Since this project takes place in Finland, only a LoRaWAN module which can transmit and receive within the EU863-870 band can be used.



*Figure 21: the M5Stack LoRaWAN868 module used in this project*

The chosen module is the Unit LoRaWAN868 from M5Stack. It is a LoRaWAN compatible module with an operation frequency of 868 MHz. The LoRa radio inside the module is made by Semtech and is called the ASR6501. The LoRa modulated signal gets transmitted via an 11 cm long antenna with an SMA connector. Figure 21 shows the module with its antenna in an upright position.

The communication between the module and the LoRa radio chip inside happens through SPI. Communication between the module and an external microcontroller is possible through serial

communication (UART). The module itself can be programmed from a series of AT commands. The exact AT commands which the module understands and can use are written down into the documentation of the module [33]. This document copies most of the regional parameter settings from the official LoRaWAN documentation and translates them into configurable parameters as AT commands.

It is notable that these AT commands only support the LoRaWAN protocol. It is not possible to use this module to directly translate messages into a LoRa modulated signal. This makes it impossible to use this module for peer-to-peer (P2P) communication.



## 4 Raspberry Pi

This project makes use of two Raspberry Pi SBCs. One of them is used as the main computer for the machine vision part of the project. The other one is used as a webserver to host a Node-RED dashboard. Together they provide the computing power for this project.

The Raspberry Pi is a series of minicomputers, developed by the Raspberry Pi Foundation. The newest model in the series is called the Raspberry Pi 5. This chapter shows insight into the workings of the Raspberry Pi Foundation and explains the motivation for using a Raspberry Pi 4B in this project.

### 4.1 Raspberry Pi Foundation and Raspberry Pi Ltd.

The Raspberry Pi Foundation is a non-profit organization from the United Kingdom, founded in 2009. Their main goal is to promote the teaching of basic computer science and programming skills in schools and developing countries [34].

To achieve this, the Foundation developed the Raspberry Pi minicomputer in 2012. It has everything a real computer has, like a CPU, GPU, RAM, USB ports and even HDMI ports, but all of these in an incredibly small package. It measures 85 mm by 56 mm, comparable in size to a credit card [34].

Along with this minicomputer, the Foundation developed a Linux operating system called Raspbian (currently known as Raspberry Pi OS), specifically perfected for the Raspberry Pi.

The Foundation keeps developing new software for the Raspberry Pi and provides educational projects on their website [35].





The commercial aspects of the Raspberry Pi minicomputer, like manufacturing and distribution, are conducted by the Raspberry Pi Ltd. company. This company is a subsidiary of the Raspberry Pi Foundation and was founded at the launch of the first Raspberry Pi in 2012. They make sure that the price for the Raspberry Pi stays affordable, without wasting the resources of the Foundation.

### 4.2 Motivation for Raspberry Pi 4B

The minicomputers used in this project are both Raspberry Pi, model 4B. The reason for choosing the older model over the newer Raspberry Pi 5 is mainly because of power draw. While the Raspberry Pi 5 offers a significant performance upgrade, the module draws noticeably more power. According to the official documentation, a Raspberry Pi 5 needs a power supply of 5 V with a current rating of 5 A. The Raspberry Pi 4B only requires a power supply of 5 V with a current rating of 3 A [36] [37].

In the current form of the project, the computing power of the Raspberry Pi 4B suffices and the added power draw of the Pi 5 has more negative effects than the increase in performance would offer.

In future revisions, it could make more sense to upgrade to more specific hardware for machine learning or machine vision, to get more performance out of the project. A good example of a specific machine learning oriented SBC is the Nvidia Jetson Orin NX platform. These SBCs come with Nvidia Ampere GPUs, which have 1024 CUDA cores, specifically made for machine learning tasks [38]. They can be used to run at 10 W, which is comparable to the power draw of the Raspberry Pi 4B. The reason these boards are not used in this project is because of their significant cost. On the Farnell website, a developer kit for the Orin NX costs € 632,95 at the time of writing, while a Raspberry Pi 4B with 8 GB of RAM costs € 67,91 [39].

Compare Products: 2 Products Shown	
<div>Clear All</div> <div> <div>Manufacturer Part No</div> <div>RPi4-MODBP-8GB</div> <div></div> </div> <div> <div>Description / Manufacturer</div> <div>SBC, Raspberry Pi 4 B 8GB, BCM2711, ARM Cortex-A72, 8GB RAM, MicroSD, Linux, WiFi, 2x micro HDMI</div> <div>RASPBERRY-PI</div> </div> <div> <div>Quantity</div> <div>1</div> <div>Add</div> <div>Min: 1 / Mult: 1</div> </div> <div> <div>Availability</div> <div> 31,042 in stock</div> </div> <div> <div>Price For</div> <div>Each</div> </div> <div> <div>Price</div> <div>1+ € 67.910</div> </div>	<div> <div>110110144</div> <div></div> </div> <div> <div>Description / Manufacturer</div> <div>Developer Kit, NVIDIA Jetson Orin NX, ARM Cortex-A78AE CPU, Ampere, 8GB RAM, 128GB NVMe, HDMI</div> <div>SEED STUDIO</div> </div> <div> <div>Quantity</div> <div>1</div> <div>Add</div> <div>Min: 1 / Mult: 1</div> </div> <div> <div>Availability</div> <div> 8 in stock</div> </div> <div> <div>Price For</div> <div>Each</div> </div> <div> <div>Price</div> <div>1+ € 632.950</div> </div>

**Figure 22: Price comparison between Raspberry Pi 4B 8 GB and NVIDIA Orin NX [39]**

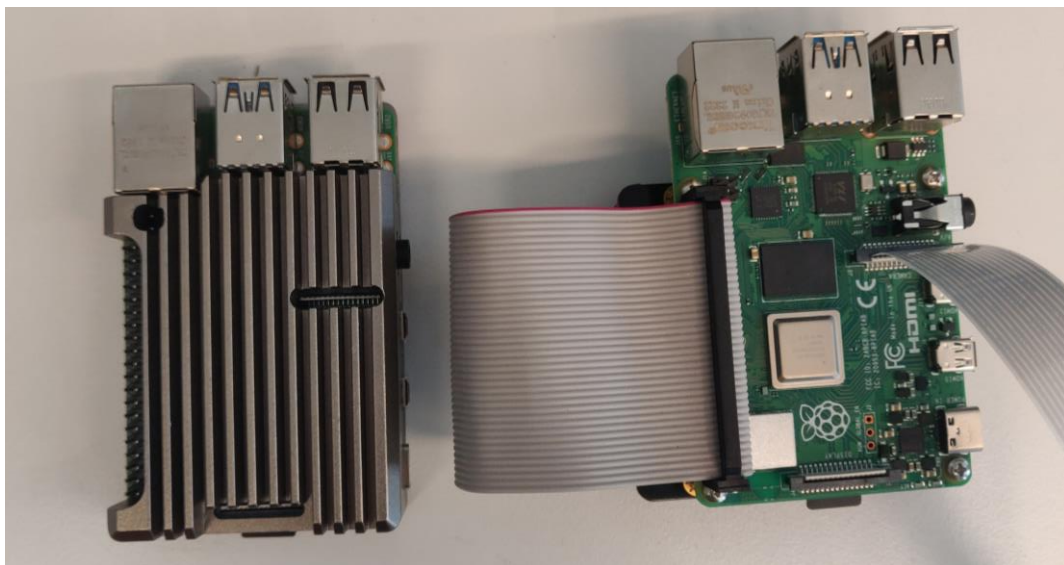
On top of the hefty price, these SBCs require more machine learning and programming knowledge to set up, compared to a Raspberry Pi.

To conclude, the Raspberry Pi 4B is chosen over the Raspberry Pi 5, because of power efficiency. If the project gets extended in the future, the usage of a more machine learning oriented SBC is recommended, like the Nvidia Jetson Orin NX platform. Their hefty cost could be worth the significant increase in performance, even if it requires an experienced machine learning developer to work with the module.

### 4.3 Project Implementation: Raspberry Pi Modules

In Figure 23, both Pi's used in this project are displayed. The Pi on the left is the Node-RED server. It is a Pi 4B with 4 GB of RAM and it is running Raspberry Pi Bullseye OS as its operating system.

The Pi on the right is a Pi 4B with 8 GB of RAM. This is the Pi used for the main project. It is running Raspberry Pi Bookworm OS as its operating system. This Pi is equipped with a CSI ribbon cable, which goes to the camera module. There is also another ribbon cable visible on the board. It leads to a DFRobot IO expansion HAT, which is used for connecting the M5Stack LoRaWAN module to the serial port of the Pi.



**Figure 23: the two Raspberry Pi's used in this project**

## 5 Camera module

This chapter focuses on the motivation and implementation of the camera module used in this project. Advice for selecting future cameras is given.

### 5.1 Project Implementation

The camera used in this project is the Raspberry Pi High Quality camera module, together with a 16 mm telephoto lens [40] [41]. The sensor of this module is an IMX477R from Sony. It features 12,3 megapixels with a pixel size of  $1,55\text{ }\mu\text{m}$  by  $1,55\text{ }\mu\text{m}$ . This translates to a resolution of 4056 by 3040 pixels, often referred to as 4K. The camera can capture images in RAW12 output format. This format uses 12 bytes of data to represent one captured pixel. The RAW file format does not compress any of the information of the pixels. This is ideal for the application in this project, where a high quality image is an absolute necessity after detection. However, there is a balance to be found between the image quality and the processing speed. By choosing the High Quality camera, the image resolution can be set to 640 px by 480 px (which is needed for the model to run properly), while also being capable of capturing a 4K resolution image of detected objects.

The High Quality camera module comes with a PCB with the camera sensor on, a flexible ribbon cable and either a C-CS-mount adapter or an M12 adapter for lenses. The module on its own does not come with a lens. This has the advantage that, depending on the application, a specific lens can be used. For this project, the High Quality camera module with the C-CS-mount adapter was chosen, together with a C-mount 16 mm telephoto lens. The reason for using a telephoto lens is because the position of the camera module is quite far from where the boats should pass through. An added advantage of using a telephoto lens is that it drastically narrows the field of view of the camera, making it easier to keep noise out of the area of interest. Figure 24 [42] shows the High Quality camera module with the C-CS-mount adapter.

One major disadvantage of this camera module is that the aperture and the focus are only manually configurable by turning the appropriate rings on the lens. This means that a good setting must be found through testing, but that the performance of the lens will fluctuate based on the weather conditions. The zoom lens is needed, as it enables the camera to observe boats at a similar distance to what the model is trained on. The zoom quality of the lens is so good that the flexibility of the aperture and focus can be given up.



*Figure 24: the Raspberry Pi High Quality camera module [42]*

### 5.2 Alternatives for Future Use

It is notable that Raspberry Pi also makes a global shutter camera module. The difference between the global shutter camera module and the High Quality camera module lies in the working of the shutters and thus in the way an image gets produced by the sensor. In a rolling shutter camera, like the one in the High Quality camera module, the sensor moves over the image from the top or bottom and reads the

pixels line by line. In a global shutter camera, all the pixels get exposed at the same time and the complete image gets read instantaneously [43].

The working of a global shutter is often preferred in fast moving machine vision applications, to make sure that a captured image does not have any distortions from the capturing process. However, to achieve the working of the global shutter and keep the price at the same level as the Raspberry Pi High Quality camera, tradeoffs were made. In terms of the Raspberry Pi Global Shutter camera module, the tradeoff comes in the image quality. The sensor features only 1,6 megapixels of image resolution. Global shutter cameras with higher image resolutions come at a significant price increase. For example, the company FLIR sells a global shutter camera called the "Blackfly S USB3", which can be configured to have similar image resolutions to the Raspberry Pi High Quality camera. This model with 12 megapixels resolution costs € 443,00 on their site [44].

Because this project focuses on recording images for future improvements to the model and the environment of the project does not deal with any exceptionally fast-moving objects, the decision is made to use the Raspberry Pi High Quality camera module.

In a future version of the project, if the need for recording images is no longer there, and there is sufficient budget available, a high-quality global shutter camera module like the FLIR Blackfly S USB3 could be considered.

## 6 Dashboard

For the visualization and logging of the data, the data must be made accessible through an alternative service than the LoRaWAN network. Digita Oy provides a dashboard for developers, to track the data packages of LoRaWAN modules, but since this project sends raw hexadecimal strings, an end user would be nothing with it. More insight into the exact LoRaWAN data can be found in Chapter 7.

Accessibility to the LoRaWAN data is provided through an MQTT broker. The data can be read by subscribing to the correct MQTT topic and is then further used to be displayed as a Node-RED dashboard. The Node-RED application does not only relay the data to a dashboard, but it also logs all data every twenty-four hours. The logged data is written to a comma-separated values (CSV) file and stored locally on a Raspberry Pi.

### 6.1 MQTT

MQTT works on the principle of publishing and subscribing to topics. Clients can choose to do both. If a client is subscribed to a topic, they will see every message published on that topic. The broker is the server which relays data correctly from the publishing client to any subscribed client over the internet [45].

HAMK hosts their own publicly available MQTT broker, accessible through the address "iot.research.hamk.fi." and port 1883. It is linked to the Digita Oy. LoRaWAN server and automatically relays messages to MQTT topics. This MQTT broker facilitates a topic for this project, reachable as "HAMK/boat/counter". Any interested party can subscribe to this topic and read out the raw data of the LoRaWAN messages. There is no concern about privacy or data leaks, as the only data sent through the LoRaWAN network are hexadecimal strings holding boat counter data. There is no exchange of sensitive or personal data.

A big advantage of using MQTT is the scalability of the protocol. There can be a virtually endless number of subscribers to the topic, as many as the broker can provide. Additionally, by combining it with Node-RED, the data is extremely easy to deploy, as the entire Node-RED flow can be exported in a single JSON file. If students are interested in doing summer projects, they have free, easy access to real-time boat counter data. The entire code base is available at the GitHub repository [17]

### 6.2 Node-RED

Node-RED is an open-source software tool for connecting programmed functions together in a browser-based editor [46]. Code can be written in separate blocks called nodes and then connected through "flows". This creates a visual overview of how code interconnects with each other. The function nodes can be programmed using the JavaScript language.

The backend of Node-RED is provided by the "Node.js" runtime, which is an event-driven runtime [47]. This means the software does not consume any computational resources when idle. Only when there is something to do (in this application: processing a new MQTT message), the software will use computing power from the device it is deployed on. This makes it very lightweight and portable, ideal for a Raspberry Pi [48].

Because Node-red is open source, users can add new features to the software by creating new nodes. One of these user-added nodes is used in this project. It is called "node-red-contrib-fs" and is written by Julian Knight [49].

Another benefit of using Node-RED as a software tool is the simplicity of creating web-based dashboards. Node-RED comes with a dashboard as one of the core nodes. It has multiple input and output fields, to show data. The site which displays the dashboard comes already formatted with CSS

and adjusts itself to the screen size of the device on which it is accessed. This way, the dashboard can be displayed on mobile devices and computers without having to add any extra CSS.

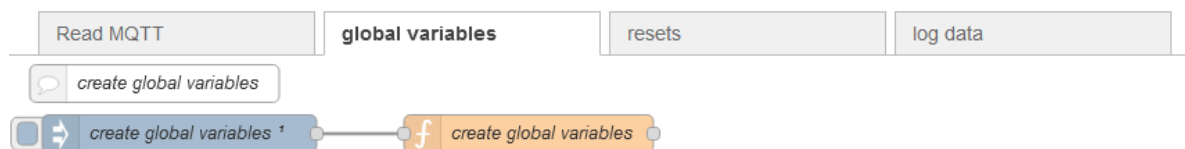
For this project, four flows were created: Read MQTT, global variables, resets and log data. They can be divided into two groups: Visualization and Data Logging.

To keep the document structured, the choice was made to add the JavaScript code from the function nodes as appendixes. To get a good understanding of how the code works, the reader is recommended to consult the appendixes while reading this chapter.

### 6.3 Project Implementation: Visualization

The visualization nodes are: Read MQTT, global variables and resets. Together, they show the daily and cumulative count of boats throughout the summer season.

When the flow is deployed, the global counter variables are created and initialized by the “global variables flow”. Figure 25 shows the flow in question.



*Figure 25: the global variables flow*

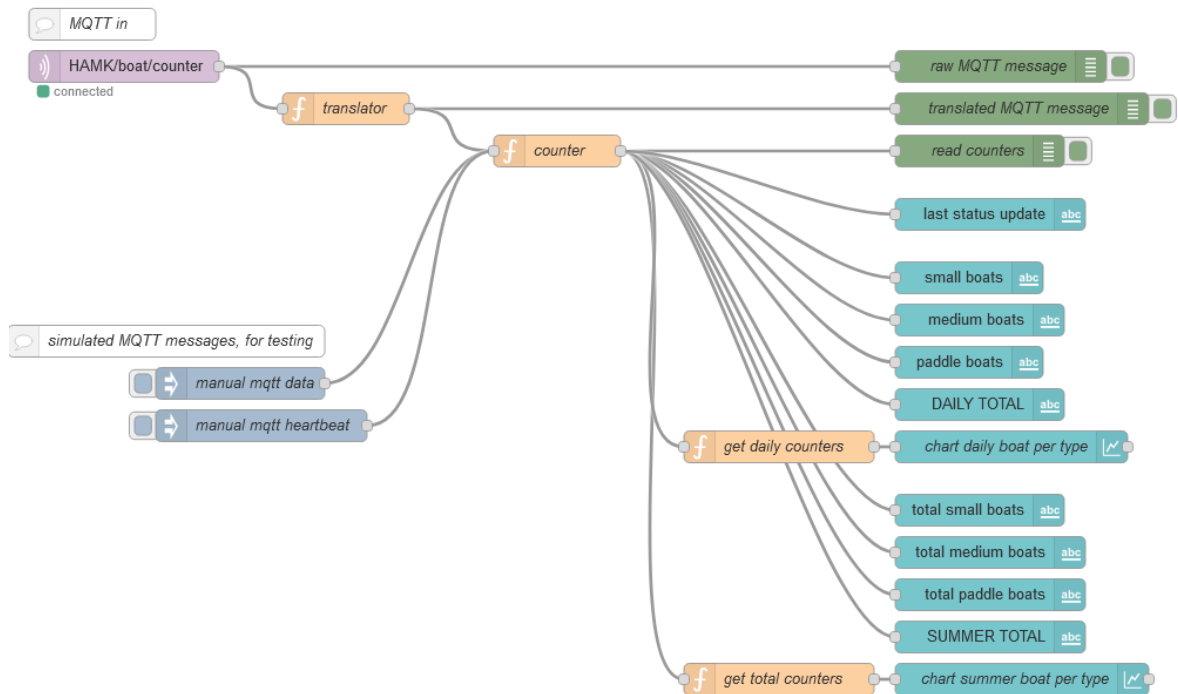
The inject node “create global variables” (blue on the figure) runs exactly once, right after the deployment of the node. This triggers the function node “create global variables” (orange on the figure), which has JavaScript code to initialize the global counter variables. In total, eight counter variables are created: counterS, counterM, counterP, counterDay, counterStot, counterMtot, counterPtot and counterTot. The first four of these variables are used to store daily counter data, while the last four are used to store the total counter data for the boat types. There is one more global variable initialized by this function node: heartbeat. This variable stores the timestamp of the last heartbeat signal. The heartbeat signal is further described in Chapter 7.7. Global variables are used to make sure that all variables are accessible in every node and that they are updated correctly.

Immediately after deployment, simultaneous with the “global variables” flow, the “Read MQTT” flow subscribes to the “HAMK/boat/counter” MQTT topic through the MQTT node (as shown in purple in Figure 26).

Each time there is new data published on the MQTT topic, the flow uses the “translator” node to convert the data from hexadecimal string to a string.

The “counter” node accesses this string, extracts the numbers from it and feeds them to the corresponding counter variables.

All the counter variables are displayed on the dashboard. The dashboard elements are noticeable by their light blue color in the flow, as displayed in Figure 26. There are green debug nodes visible in this flow, to aid users in understanding the working of each node. The dark blue inject nodes can be used to simulate new MQTT messages, for debugging or testing purposes. In the final deployment they should be left untouched.



**Figure 26: the Read MQTT flow**

The function nodes “get daily counters” and “get total counters” are used to format the counter variables in the correct way to be displayed as a bar graph on the dashboard. All the code is copied from an example in the Node-RED documentation around bar charts [50]. All code is well documented and can be accessed through the GitHub page of this project [17].

The dashboard, as shown in Figure 27, displays the total number of boats for the day and divides the total count in each separate boat class. A bar graph of the individual count of each class is added, to serve as a visual representation of the data. The display of the total count of boats over the entire summer follows the same pattern. A timestamp of when the last heartbeat message was received is also displayed on the bottom of the dashboard.

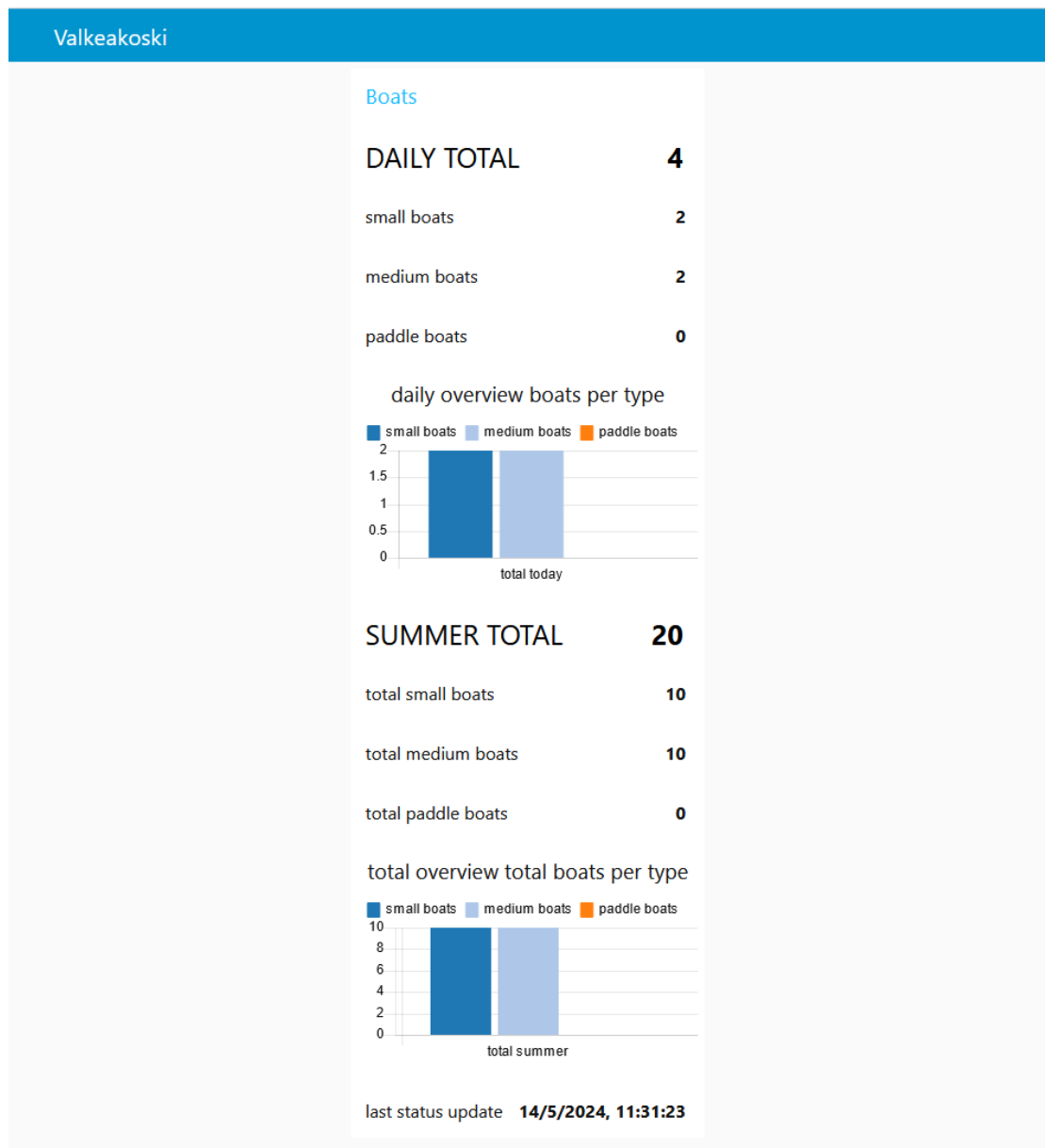


Figure 27: the dashboard, filled with example data

The “resets” flow manages the correct resets of the daily counter variables. Every 24 hours, at midnight, the daily counters get reset to zero. The looping is implemented by the “midnight loop trigger” inject node (dark blue in the figure). The resetting of the counters is done by the “reset daily counters” function node. Figure 28 shows a visual overview of the flow.



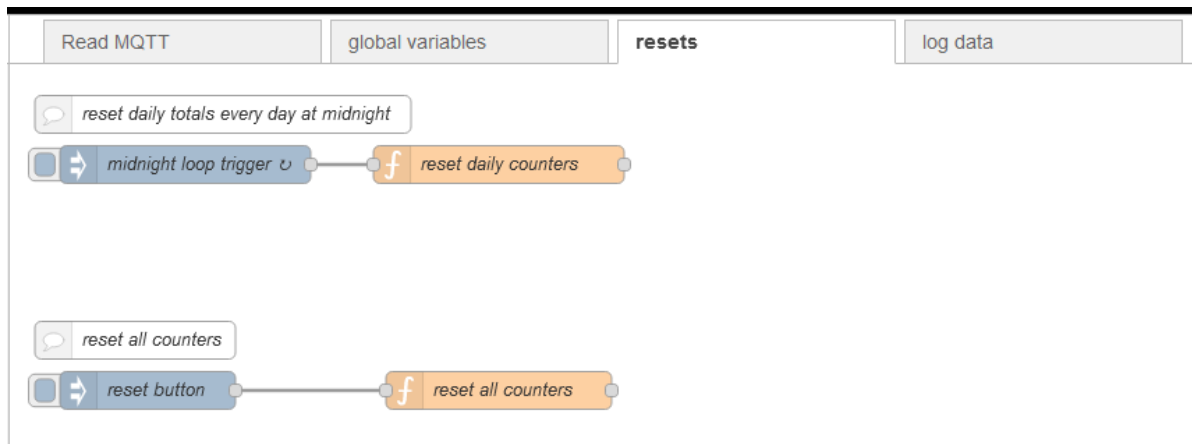


Figure 28: the resets flow

There is also a reset button present. It can be used during testing and debugging. However, it should never be used during the implementation of the project, as it will reset all counter objects. Triggering this button may lead to potential data loss, which can only be restored by consulting the logs. Up to one full day of data can be lost.

## 6.4 Project Implementation: Data Logging

The “log data” flow implements data logging, which is an indispensable part of the project. This step involves saving the boat counter data every 24 hours, precisely at 23:30 in the evening. This to ensure the data is logged before the daily counters are reset. The availability of a daily log enhances the overall resilience of the project. In case of malfunctions, no more than one full day of data is lost. The flow is fully copied from an online example, created by GitHub user nymga2004 [51]. To access the filesystem of the Raspberry Pi through Node-RED, a custom node must be installed. It is called “node-red-contrib-fs” and was developed by Julian Knight [49]. The custom node is called “File Lister” and is shown in purple in Figure 29.

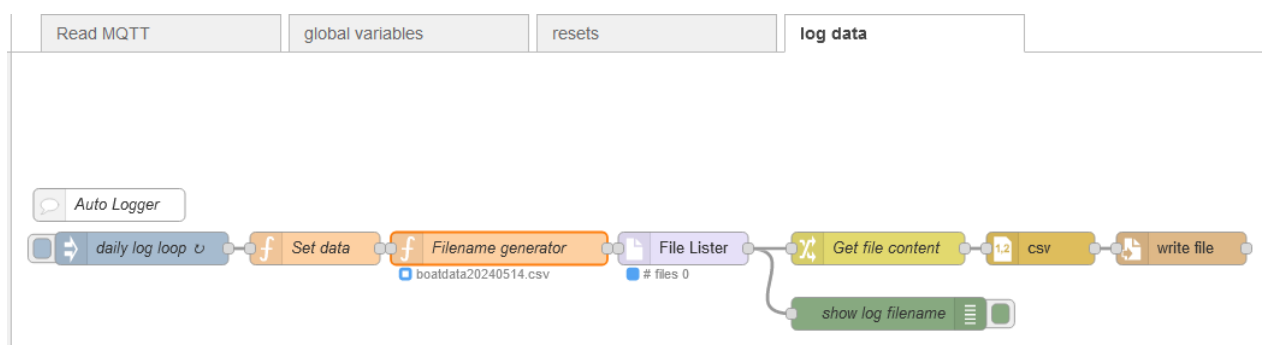


Figure 29: the log data flow

The inject node “daily log loop” (dark blue in Figure 29) triggers the flow to run every 24 hours. Next, the function node “Set data” reads the global counter variables and stores them as a payload of a msg object.

Following, the “Filename generator” function node creates a filename, based on the current date. The filename is stored under “msg.fname”. The full path where the file should be stored is stored under “msg.filecontent”. The “msg.payload” contains the “msg.fname” object but formats it so the File Lister node can use it to check whether the file already exists or not.

The File Lister node checks whether the file from “Filename generator” already exists on the given path. If it does not exist yet, it adds the path to the correct folder to the filename. The chosen folder for this project is “/home/admin/datalog”. The exact configuration of the node is displayed in Figure 30.

**Edit fs-file-lister node**

Delete Cancel Done

**Properties**

Start Folder: /home/admin/datalog/

File Pattern: \*\*

Folder Pattern: \*

Files/Folders: Files

☒ Include hidden files in output?

☒ Include full path in output?

☒ Output single message (array)?

☐ Show "Cannot access file" warnings?

Max. search depth: 0

☒ Return file details?

Name: Name

**Figure 30: configuration of the File Lister node**

After the “File Lister” node is done, the change node “Get file content” accesses the “msg.filecontent” object from the “Filename generator” function node and passes it on to the “csv” node. The configuration can be seen in Figure 31.

**Edit change node**

Delete Cancel Done

**Properties**

Name: Get file content

**Rules**

Set msg. payload

to the value msg. filecontent

☐ Deep copy value

**Figure 31: configuration of the change node**

When the “csv” node receives the data from “Get file content”, it converts the data to a correct csv file format. The headers of the csv file are manually filled in to match the strings stored in the msg.payload of the “Set data” function node. Figure 32 shows the configuration of this node.

**Edit csv node**

Delete Cancel Done

**Properties**

Columns: timestamp,daily\_small\_boat,daily\_medium\_boat,daily\_large\_boat

Separator: comma

Name:

**CSV to Object options**

Input: Skip first 0 lines

- ☐ first row contains column names
- ☒ parse numerical values
- ☐ include empty strings
- ☐ include null values

Output: a message per row

**Object to CSV options**

Output: always send column headers

Newline: Linux (\n)

**Figure 32: configuration of the csv node**

Finally, the “write file” node writes the data from the “csv” node to the given file and path created as “msg.filename” in the function node “File generator”, as shown in Figure 33.

**Edit write file node**

Delete Cancel Done

**Properties**

Filename: msg.filename

Action: append to file

- ☐ Add newline (\n) to each payload?
- ☒ Create directory if it doesn't exist?

Encoding: default

Name:

Tip: The filename should be an absolute path, otherwise it will be relative to the working directory of the Node-RED process.

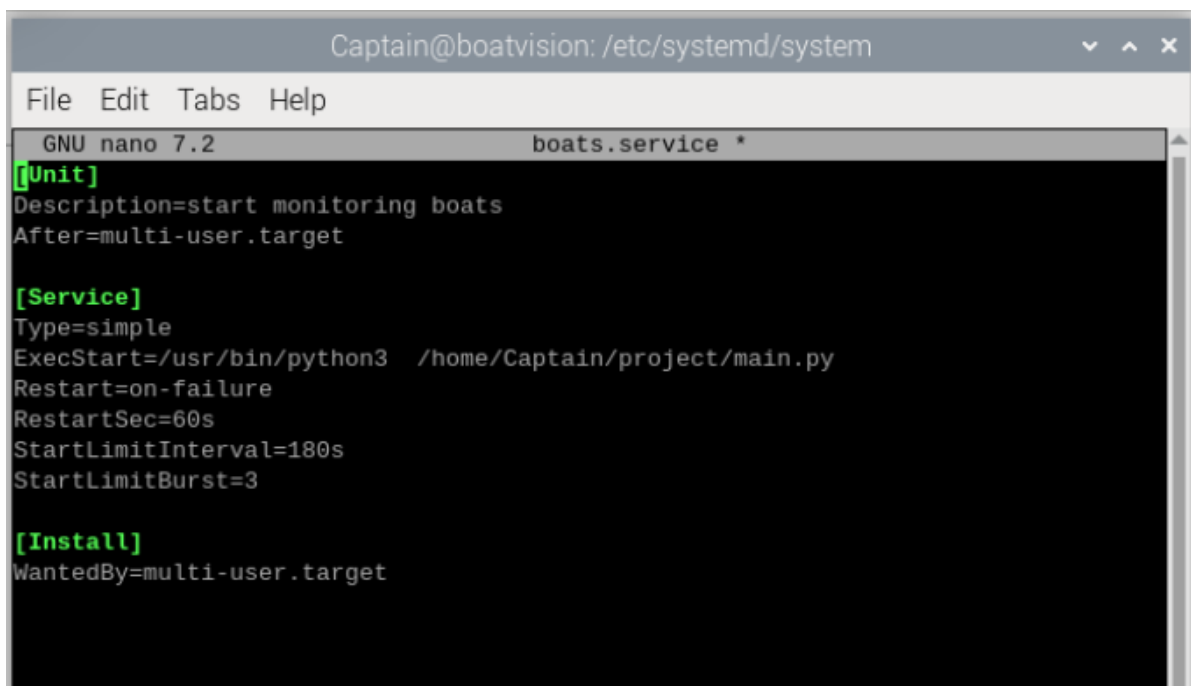
**Figure 33: configuration of the write file node**

## 7 Main Code Analysis and Explanation

The main code for the machine vision part of this project consists of five files. This chapter describes each file and shows how they affect each other. Furthermore, specific software settings around automating the code are described.

### 7.1 Systemd

The main code around the machine vision part of this project starts running automatically after boot. This is achieved by running the code as a systemd service. Systemd is a Linux specific software package, which controls several system settings and services [52]. Among these services is the boot sequence. If a Linux computer boots up, systemd runs a long checklist to ensure a proper boot. Users can add their own tasks, known as services, to the list. Figure 34 shows the service for the main code of this project.



```
Captain@boatvision: /etc/systemd/system
File Edit Tabs Help
GNU nano 7.2 boats.service *
[Unit]
Description=start monitoring boats
After=multi-user.target

[Service]
Type=simple
ExecStart=/usr/bin/python3 /home/Captain/project/main.py
Restart=on-failure
RestartSec=60s
StartLimitInterval=180s
StartLimitBurst=3

[Install]
WantedBy=multi-user.target
```

*Figure 34: the systemd service file which runs the main code automatically after boot*

The service file is called “boats.service”. By using `Restart=on-failure`, the service will rerun if it fails to start. The `RestartSec`, `StartLimitInterval` and `StartLimitBurst` parameters limit the amount of reruns to 3 tries, separated by 60 second intervals and for a maximum total duration of 180 seconds. This adds resilience to the execution of the code.

### 7.2 Crontab

Cron is a job scheduler for Linux/UNIX-like operating systems. As Raspberry Pi OS is Linux based, it comes with cron preinstalled [53]. Through cron it is really easy to make code run on fixed time intervals.

In Figure 35, the crontab overview is shown. The crontab is the file where automatic jobs, called cronjobs, can be listed. Note that there is a cronjob scheduled as `*/15 * * * *`. This means this code file will run every 15 minutes. The comments in blue explain thoroughly how the parameters work. For more information about the file “heartbeat.py”, see further in this chapter.

```

Captain@boatvision: ~
File Edit Tabs Help
GNU nano 7.2 /tmp/crontab.dGT4ZF/crontab
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow  command
*/15 * * * * python3 /home/Captain/project/heartbeat.py
^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute   ^C Location
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^_ Go To Line

```

*Figure 35: the crontab of the Raspberry Pi running the machine vision code*

### 7.3 Send\_data.py

The first code file is made to interface with the LoRaWAN module over AT commands, specifically for sending data, configuring and reading from the serial port. It defines four functions: open\_serial, wait\_response, send\_message and encode\_msg.

This file makes use of the Python serial and time libraries. It has the Python shebang, to make the script executable from a terminal.

```

#!/usr/bin/python3
import serial
import time

```

*Code Fragment 1: imported libraries for send\_data.py*

The open\_serial function takes no arguments. It relies on the Python serial library to configure the serial port of the Raspberry Pi (GPIO pins 14 and 15, respectively TXD and RXD). Once the port is correctly configured and initialized, it flushes their in- and output buffer, to ensure that every following data will go through exactly as intended. It returns the configured serial port as a variable "ser".

```

#function for opening serial port at correct baudrate
def open_serial():
    port = "/dev/ttyS0"
    baudrate = 115200
    ser = serial.Serial(port, baudrate, timeout=1) #keep timeout for sending
    low
    ser.flushInput()
    ser.flushOutput()
    return ser

```

*Code Fragment 2: open\_serial function of send\_data.py*

The `wait_response` function takes two arguments: `serial_port` and `time_out`.

The `serial_port` argument refers to the correct serial port to read from. In the context of this project, the `serial_port` variable will always be "ser" from the `open_serial` function.

The `time_out` argument expects an integer. The value of the integer is used as seconds to read from the serial port. This value is typically kept around one second, but during testing, it was elevated, to assure everything was properly read from the serial port.

The function creates a new internal variable called "response". The function uses a non-interrupting loop to read data from the serial port and paste it as a string onto the "response" variable. This loop is sustained by the variables "start", "current\_millis", and "time\_out". The loop relies upon the working of the `time()` function from the Python time library to read the elapsed amount of microseconds in the given program. These microseconds are multiplied by 1000 to facilitate working in seconds. To assure the message is properly translated to a string, the `decode()` function from the serial library is accessed.

If there is data available from the serial port, it can communicate this through the "in\_waiting" parameter from the Python serial library. The reading of the state of this parameter is used alongside the time restriction of the non-interruptive loop as added redundancy.

If the loop times out, and there is nothing left to read from the serial port (`in_waiting` is False), the loop breaks, and the string "response" is returned.

```
#function for reading out the serial port
def wait_response(serial_port, time_out):
    response=""
    start = time.time()*1000
    while True:
        current_millis = time.time()*1000
        if serial_port.in_waiting or (current_millis - start) < time_out:
            msg = serial_port.readline().decode()
            response += msg
        else:
            break
    return response
```

*Code Fragment 3: function wait\_response from send\_data.py*

The `send_message` function takes four arguments: `serial_port`, `confirm`, `nbtrials` and `message`. These arguments are chosen for seamless integration with how the M5Stack LoRaWAN module expects the AT command for sending data [41].

The `serial_port` argument is exactly the same as the one used in `wait_response` and translates to the "ser" object returned by the `open_serial` function.

The `confirm` argument corresponds to the "confirm" bit needed in the AT command for LoRaWAN messaging. If this bit is 0, the LoRaWAN gateway will not confirm the uplink of the message. If the bit is 1, the gateway will confirm the uplink. In this project, the confirm bit is always set to 0, but for future-proof purposes, the bit is settable through the function.

The `nbtrials` argument takes an integer in the range from 1 up to and including 15. This argument corresponds to the "nbtrials" argument required in the AT command for LoRaWAN messaging [33]. In the current application of the project, the `nbtrials` is kept to 1. From testing it has become apparent that messages always reach their destination. Keeping the `nbtrials` as argument allows for easy customisation in the future, if data transmission issues arise.

The `message` argument takes a string as input. This string gets encoded into hex by the `encode_msg` function, because LoRaWAN expects hexadecimal data to be sent.

The function takes the length of the hex encoded version of the message string. Again, this is done in accordance with the expectation of the LoRaWAN messaging documentation and the AT+DTRX command from the M5Stack module [27] [33].

Finally, the function writes the AT+DTRX command to the serial port.

```
#function for sending LoRaWAN message
def send_message(serial_port, confirm, nbtrials, message):
    time.sleep(0.1)
    hex_msg = encode_msg(message)
    hex_msg_length = len(hex_msg)
    serial_port.write(("AT+DTRX="+str(confirm) + "," + str(nbtrials) + "," +
str(hex_msg_length) + "," + str(hex_msg)+"\r\n").encode())
```

*Code Fragment 4: send\_message function from send\_data.py*

The function encode\_msg takes exactly one argument: message.

The message argument is a string. In the context of this code, it is the same message argument as in the send\_message function.

This function translates a string into the equivalent hexadecimal characters and returns them as a variable "hex\_string".

```
#function for converting data from String to Hex
def encode_msg(message):
    byte_array = bytearray(message, 'utf-8')
    hex_string = ''.join(format(byte, '02x') for byte in byte_array)
    return hex_string
```

*Code Fragment 5: encode\_msg function from send\_data.py*

## 7.4 Lora.py

The second code file is made to correctly set up the M5Stack module, according to the AT commands documentation, as well as the LoRaWAN regional parameters documentation [27] [33]. The underlying Python libraries used are again serial and time. For reading out the serial port, the wait\_response function from the send\_data file is reused. The file has the Python shebang, to make it executable from terminal.

```
#!/usr/bin/python3
import serial
import time
from send_data import *
```

*Code Fragment 6: imported libraries for lora.py*

There is only one function in this file: send\_command. It is similar to the send\_message function from the send\_data file, but it is deliberately kept simple, to allow for easy customization to the different setup commands.

```
def send_command(command):
    time.sleep(0.1)
    ser.write(("AT+" + command + "\r\n").encode())
```

*Code Fragment 7: send\_command function from lora.py*

The exact setup commands can be found on the GitHub repository [17]. They are well documented by comment blocks in the code. The setup commands are all derived from the regional parameters required for LoRaWAN communication, as previously described in Chapter 3.2.

## 7.5 Random\_folder\_generator.py

The third code file is used for creating a random directory onto the external SSD of the project. It defines two functions: `get_random_bytes` and `create_directory`.

This file uses the Python `os` library since the folder must be created at the drive's mounting point.

```
import os
```

*Code Fragment 8: imported library for random\_folder\_generator.py*

The first function, `get_random_bytes` takes one argument: `num_bytes`. It expects an integer to represent the amount of bytes which should be read from the `os.urandom` module. Based on the value of `num_bytes`, a string object gets created, called `random_bytes_str`. This string contains the hex representation of the random bytes generated by `os.urandom`. This string serves as the return value of the function.

```
#function for reading random bytes from urandom
def get_random_bytes(num_bytes):
    random_bytes_str = os.urandom(num_bytes).hex()
    return random_bytes_str
```

*Code Fragment 9: get\_random\_bytes function from random\_folder\_generator.py*

The second function, `create_directory`, takes two arguments: `mount_point` and `directory_name`.

The `mount_point` argument expects a string representation of the mounting point of the drive.

The `directory_name` expects a string representation for the name of the directory. The `os.makedirs` function is used to create a directory on the drive. If the creation of the random folder fails, an `OSError` will be thrown, and a fault message will be displayed. The function returns `False`. If the directory is successfully made, the function returns `True`.

```
#function for creating a directory on an external hard drive
def create_directory(mount_point, directory_name):
    try:
        directory_path = os.path.join(mount_point, directory_name)
        os.makedirs(directory_path)
        return True
    except OSError as e:
        print("something went wrong, random directory wasn't created")
        return False
```

*Code Fragment 10: create\_directory function of random\_folder\_generator.py*

## 7.6 Main.py

The main code file uses all previously mentioned files to implement the machine vision. For working with the camera, the experimental Python library `Picamera2` is used, as well as the `cv2` library. The `cv2` library is strictly speaking not needed for the final version of the project, as the Raspberry Pi will run completely headless without Graphical User Interface (GUI) environment. However, it is imported to facilitate GUI troubleshooting, which is necessary during testing. To implement the machine vision part, the Ultralytics



library is used. For efficient mathematical operations, the numpy library is needed as well. A Python shebang is added, to make the code executable from terminal.

```
#!/usr/bin/python3
from picamera2 import Picamera2, Preview
import cv2
from ultralytics import YOLO
import numpy as np
from send_data import *
from random_folder_generator import *
```

*Code Fragment 11: imported libraries for main.py*

The entire code can be found on the GitHub repository [17]. It is well documented with comment blocks. The emphasis of this section will be on a self-made counting function and the central code loop. The function `count_values` takes one argument: `arr`. This function expects `arr` to be an array of non-negative integers.

The function implements the Numpy `bincount` function on `arr`. This will store a histogram of the original array as the array `"counts"`, where the bin at index `"i"` of `counts` shows the occurrences of the value `"i"` in the input array `arr`.

After the count is completed, a formatted string is formed and returned as `form_str`. It shows the counts of `"1"`, `"2"` and `"3"` in the original array. These numbers translate to the counts of a small boat (1), medium boat (2) and paddle boat (3) in one frame.

```
#make a counter function from the results of the model
def count_values(arr):
    counts = np.bincount(arr, None, 4) #np bincount is the most efficient
    counter in Python
    form_str = "S{},M{},P{}".format(counts[1], counts[2], counts[3]) #string
    formatting for easy data extraction
    return form_str
```

*Code Fragment 12: count\_values function of main.py*

The main event loop of the file reads a frame of the camera as a Numpy array and feeds it to the model, to perform machine vision.

The model will only report if it detects classes 1, 2 or 3, which translates to the small boat, medium boat or paddle boat classes. Furthermore, it will only report a detection if the confidence score is higher than 50 %.

The results of the detections are saved as a PyTorch Tensor called `boxes`. In order to work with these results, the Tensor datatype needs to be translated to the Numpy ndarrays.

One of the arrays inside the `boxes` ndarray is accessible through `boxes.cls`. It is a one-dimensional array which stores the classes of the detected objects. For example, if there are three small boats detected in a frame, the `boxes.cls` array will look like this: `[1 1 1]`. However, the numerical values contained within this array are of the floating-point data type. To enable the application of Numpy's `bincount` function to this array, it is required to create a duplicate array named `class_ids`, where the values are cast into the integer data type. Subsequently, the `class_ids` array is used as the argument for the `count_values` function. Lastly, the result of `count_values`, `form_str`, is used as argument for the `send_message` function to transmit the data over the LoRaWAN network.

If the boxes.cls array is not empty, the program immediately saves the current frame. Then, the camera is switched from video mode to capture mode and takes a 4K image of the subject. The picture is stored on the random folder of the external drive, which was previously generated by the code. The name of the picture is given by a counter object. After saving the images, the counter gets incremented, to serve as the name of the subsequent image.

The whole code is paused for 10 seconds. In the context of this project, the Python time.sleep function is allowed, as freezing our entire code is wished. The continue keyword brings the code back to the beginning of the while loop.

```
#main event loop
while True:
    frame = camera.capture_array("main") #read camera as np array, use main
    configuration
    results = model.predict(source=frame, imgsz=640, conf=0.5, classes=[1,2,3])
    #use model on the captured frame
    class_ids=np.zeros(1, dtype=np.int64) #create an empty array to store
    detected class IDs
    for result in results:
        boxes = result.boxes.cpu().numpy() #translate Tensors to numpy arrays,
        make sure to use CPU
        print(boxes.cls) #debug boxes.cls
        print(boxes.cls.size) #debug boxes.cls
        if boxes.cls.size > 0: #check boxes.cls.size for detections
            #frame_ = results[0].plot() #load annotated picture (with bounding
            boxes, class names and confidence scores)
            #cv2.imshow('frame', frame_) #show annotated picture
            raw_frame =
            cv2.imwrite("/home/Captain/Pictures/{folder}/frame_{counter}.jpg".format(folde
            r=random_folder,counter = counter), frame) #if external drive is connected:
            /media/Captain/boat_exfat/{folder}/{counter}
            pic =
            camera.switch_mode_and_capture_file(pic_config,"/home/Captain/Pictures/{folder
           }/{counter}.jpg".format(folder=random_folder,counter = counter)) #if external
            drive is connected: /media/Captain/boat_exfat/{folder}/{counter}
            counter += 1
            class_ids = boxes.cls.astype(int) #copy the boxes.cls to the class_ids
            array
            print(count_values(class_ids)) #check message that will be sent over
            LoRaWAN
            ser = open_serial()
            send_message(ser, 0, 0, count_values(class_ids))
            print(wait_response(ser, 1))
            time.sleep(10) #pause the entire loop for 10s
            continue

    if cv2.waitKey(1) & 0xFF==ord('q'):
        break
```

*Code Fragment 13: the main event loop of main.py*

## 7.7 Heartbeat.py

The last code file in this project is used as a status monitor.

The code relies on the send\_message function, so send\_data is imported.

```
#!/usr/bin/python3
from send_data import *
```

*Code Fragment 14: imported library for heartbeat.py*

This code makes use of the `open_serial` function to open the serial port and the `send_message` function to send a hexadecimal "00" over the LoRaWAN network.

```
#this script repeats every 15 minutes as a cronjob
ser = open_serial() #open serial port
send_message(ser, 0, 0, "\x00") #send heartbeat as "00"
```

*Code Fragment 15: the working of heartbeat.py*

This code is repeated every 15 minutes as a cronjob on the Raspberry Pi.

## Conclusion

The primary goal of the project was to estimate the flow of water traffic during events held in the touristic season in the city of Valkeakoski. The basic need of recognizing and classifying boats has been met by creating a custom machine vision model.

Furthermore, by utilizing readily available components, the project minimizes costs without compromising functionality. The selection of Raspberry Pi SBCs and Raspberry Pi camera module keeps the project hardware easy to maintain by students, as there is a vast amount of well written documentation readily available.

The technical depth of the project is shown in the chapter about machine vision. The process from annotating a custom dataset, training it as a custom model, to validating and testing its performance in a real-world test is both time consuming and technically challenging.

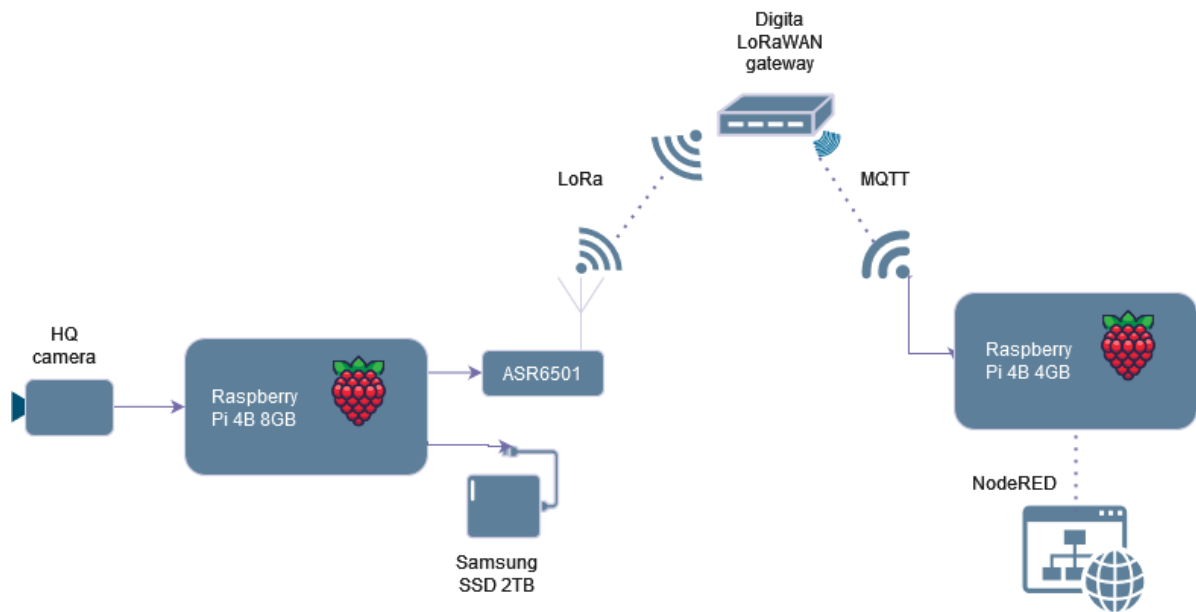
With the integration of LoRa modulation techniques and by utilizing the LoRaWAN network, the knowledge gathered through the Electronics-ICT degree in Telecommunications Engineering from Odisse University of Applied Sciences is highlighted.

Overall, this project not only meets its primary objectives but also lays a strong foundation for future advancements. By demonstrating the practical applications of machine vision and IoT technologies in a real-world setting, it contributes valuable insights to the fields of automation and smart city development.

Future work could focus on enhancing the model's accuracy and robustness, expanding its coverage, and integrating more advanced features like object tracking. For more advanced features, better performing hardware will be needed.

## Appendixes

### Appendix A: System Diagram



### Appendix B: Create Global Variables Function Node

```
global.set("counterS", 0);
global.set("counterM", 0);
global.set("counterP", 0);

global.set("counterStot", 0);
global.set("counterMtot", 0);
global.set("counterPtot", 0);

global.set("counterDay", 0);
global.set("counterTot", 0);

global.set("heartbeat", "");
return null;
```

### Appendix C: Translator Function Node

```
var hexString = msg.payload;
var asciiString = "";

for(var i = 0; i<hexString.length; i+=2){
    asciiString += String.fromCharCode(parseInt(hexString.substr(i,2),16));
}
msg.payload = asciiString;
return msg;
```

## Appendix D: Counter Function Node

```
//get global vars
var counterS = global.get("counterS");
var counterM = global.get("counterM");
var counterP = global.get("counterP");
var counterStot = global.get("counterStot");
var counterMtot = global.get("counterMtot");
var counterPtot = global.get("counterPtot");
var counterTot = global.get("counterTot");
var counterDay = global.get("counterDay");
var heartbeat = global.get("heartbeat");

//example input : "S5,M0,P0" or "00"
var data = msg.payload;
node.warn("my var data = " + data);
if(data.length > 2){ //ignore the heartbeat signal "00"
    var values = data.split(','); //split string on commas
    //example for values = ["S5", "M0", "P0"]

    //example for values[0] = "S5", substring(1) = "5"
    //parse "5" to Int and then add to counter variable
    counterS += parseInt(values[0].substring(1));
    counterStot += parseInt(values[0].substring(1));
    global.set("counterS", counterS); //update global var
    global.set("counterStot", counterStot); //update global var

    counterM += parseInt(values[1].substring(1));
    counterMtot += parseInt(values[1].substring(1));
    global.set("counterM", counterM); //update global var
    global.set("counterMtot", counterMtot); //update global var

    counterP += parseInt(values[2].substring(1));
    counterPtot += parseInt(values[2].substring(1));
    global.set("counterP", counterP); //update global var
    global.set("counterPtot", counterPtot); //update global var

    //update total counters
    var sum = parseInt(values[0].substring(1)) +
    parseInt(values[1].substring(1)) + parseInt(values[2].substring(1));
    counterDay += sum;
    counterTot += sum;
    global.set("counterTot", counterTot); //update global var
    global.set("counterDay", counterDay); //update global var
}
else{ //handle heartbeat "00"
    heartbeat = new Date().toLocaleString(); //get current timestamp
    global.set("heartbeat", heartbeat); //update global var
}
//update global variables
msg.payload = {
    counterS: counterS.toString(),
    counterM: counterM.toString(),
    counterP: counterP.toString(),
    counterStot: counterStot.toString(),
    counterMtot: counterMtot.toString(),
    counterPtot: counterPtot.toString(),
    counterDay: counterDay.toString(),
    counterTot: counterTot.toString(),
    heartbeat: heartbeat
}

return msg;
```

## Appendix E: Get Daily Counters Function Node

```
//get global vars
var counterS = global.get("counterS");
var counterM = global.get("counterM");
var counterP = global.get("counterP");

//use correct formatting for bar chart (from examples in docs)
var m={
  "series":["small boats","medium boats","paddle boats"],
  "data":[[counterS],[counterM],[counterP]],
  "labels":["total today"]
};
return {payload:[m]};
```

## Appendix F: Get Total Counters Function Node

```
//get global vars
var counterStot = global.get("counterStot");
var counterMtot = global.get("counterMtot");
var counterPtot = global.get("counterPtot");
//use correct formatting for bar chart (from example in docs)
var m={
  "series":["small boats","medium boats","paddle boats"],
  "data":[[counterStot],[counterMtot],[counterPtot]],
  "labels":["total summer"]
};
return {payload:[m]};
```

## Appendix G: Reset Daily Counters Function Node

```
global.set("counterS", 0);
global.set("counterM", 0);
global.set("counterP", 0);
global.set("counterDay", 0);
return null;
```

## Appendix H: Set Data Function Node

```
//get global vars
var counterS = global.get("counterS");
var counterM = global.get("counterM");
var counterP = global.get("counterP");
var counterStot = global.get("counterStot");
var counterMtot = global.get("counterMtot");
var counterPtot = global.get("counterPtot");
var counterTot = global.get("counterTot");
var counterDay = global.get("counterDay");

var now = new Date(); //get current timestamp
//give labels to each counter variable
//these strings will become the headers of the csv file
msg.payload = {
  "timestamp" : now.getTime(),
  "daily small_boat" : counterS,
  "daily medium_boat" : counterM,
  "daily paddle_boat" : counterP,
  "daily total" : counterDay,
  "total small_boat" : counterStot,
  "total medium_boat" : counterMtot,
  "total paddle_boat" : counterPtot,
  "total summer" : counterTot
}
return msg;
```

## Appendix I: Filename Generator Function Node

```
// Get the current time and convert it to text
var now = new Date();
var yyyy = now.getFullYear();
//perform zero padding on month, day, hours and minutes, for proper display
of dates (easier to filter later on)
var mm = now.getMonth() < 9 ? "0" + (now.getMonth() + 1) : (now.getMonth()
+ 1); // getMonth() is zero-based
var dd = now.getDate() < 10 ? "0" + now.getDate() : now.getDate();
var hh = now.getHours() < 10 ? "0" + now.getHours() : now.getHours();
var mmm = now.getMinutes() < 10 ? "0" + now.getMinutes() :
now.getMinutes();
var ss = now.getSeconds() < 10 ? "0" + now.getSeconds() :
now.getSeconds();

// Generate out file name pattern
msg.fname = "boatdata"+ yyyy + mm + dd + ".csv";
// Full filename with path for the file node later
msg.filename = "/home/admin/datalog/"+ msg.fname;

// We save the current payload into a different place on the msg object
msg.filecontent = msg.payload;

// We are passing the file name search pattern to fs node to tell us if the
file exists or not
msg.payload = {"pattern":msg.fname};

node.status({fill:"blue",shape:"ring",text:msg.fname});
return msg;
```



## List of References

- [1] "Luettelo Suomen sadasta suurimmasta järvestä," Wikipedia, 11 04 2024. [Online]. Available: [https://fi.wikipedia.org/wiki/Luettelo\\_Suomen\\_sadasta\\_suurimmasta\\_j%C3%A4rvest%C3%A4](https://fi.wikipedia.org/wiki/Luettelo_Suomen_sadasta_suurimmasta_j%C3%A4rvest%C3%A4). [Accessed 2024].
- [2] "Google Maps," Google, 2024. [Online]. Available: [https://www.google.com/maps/place/Linnosaari/@61.2702551,24.0374928,1192m/data=!3m1!1e3!4m6!3m5!1sox468ee56e4cc93763:ox4692915292b25ed!8m2!3d61.2697522!4d24.041968!16s%2Fg%2F1hc7xqtv\\_?entry=ttu](https://www.google.com/maps/place/Linnosaari/@61.2702551,24.0374928,1192m/data=!3m1!1e3!4m6!3m5!1sox468ee56e4cc93763:ox4692915292b25ed!8m2!3d61.2697522!4d24.041968!16s%2Fg%2F1hc7xqtv_?entry=ttu). [Accessed 2024].
- [3] "IP ratings," IEC, 2024. [Online]. Available: <https://www.iec.ch/ip-ratings>. [Accessed 2024].
- [4] Fibox Oy, "SOLID PC 1919 13 T," 26 07 2017. [Online]. Available: <https://www.farnell.com/datasheets/2360746.pdf>. [Accessed 2024].
- [5] Burhan-Q, glenn-jocher and RizwanMunawar, "Quickstart," Ultralytics, 08 05 2024. [Online]. Available: <https://docs.ultralytics.com/quickstart/#use-ultralytics-with-python>. [Accessed 2024].
- [6] J. Terven, . D.-M. Córdova-Esparza and . J.-A. Romero-González, "A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS," *machine learning & knowledge extraction*, no. Special, 2023.
- [7] "Convolutional Neural Networks," MathWorks, [Online]. Available: <https://nl.mathworks.com/discovery/convolutional-neural-network.html>. [Accessed 2024].
- [8] "Kaggle," [Online]. Available: <https://www.kaggle.com/datasets>. [Accessed 2024].
- [9] "Open Images Dataset V7 and Extensions," Google, [Online]. Available: <https://storage.googleapis.com/openimages/web/index.html>. [Accessed 2024].
- [10] Y. GmbH, "Yachtall," 2024. [Online]. Available: [https://www.yachtall.com/en/boat-builders-shipyards/finland?orderby=popul\\_desc](https://www.yachtall.com/en/boat-builders-shipyards/finland?orderby=popul_desc). [Accessed 2024].
- [11] S. Annika, "Sports and activities," City of Valkeakoski, 26 04 2024. [Online]. Available: <https://www.valkeakoski.fi/en/sports-and-activities/>. [Accessed 2024].
- [12] "Pirkka Lines," 2024. [Online]. Available: <https://www.pirkkalines.fi/itinerary-cruises>. [Accessed 2024].
- [13] "Majoitu," Velhovisio, [Online]. Available: <https://www.waltikka.fi/ajankohtaista/saunalautta/>. [Accessed 2024].
- [14] "When is the processing of personal data permitted?," office of the data protection ombudsman, [Online]. Available: <https://tietosuoja.fi/en/when-is-the-processing-of-personal-data-permitted>. [Accessed 2024].
- [15] Ultralytics Inc., "Modeltraining with Ultralytics YOLO," Ultralytics, 2024. [Online]. Available: <https://docs.ultralytics.com/modes/train/>. [Accessed 2024].
- [16] B.-Q. glenn-jocher, "K-Fold Cross Validation with Ultralytics," 03 12 2023. [Online]. Available: <https://docs.ultralytics.com/guides/kfold-cross-validation/>. [Accessed 2024].
- [17] Bonsa-BE, "boats," GitHub, 2024. [Online]. Available: <https://github.com/Bonsa-BE/boats>. [Accessed 2024].
- [18] I. L.-q. R. glenn-jocher, "Object Detection Datasets Overview," Ultralytics, 18 04 2024. [Online]. Available: <https://docs.ultralytics.com/datasets/detect/#ultralytics-yolo-format>. [Accessed 2024].
- [19] "Prohibited and restricted areas in waterways," Finnish Transport Infrastructure Agency, 21 03 2024. [Online]. Available: <https://vayla.fi/en/transport-network/waterways/prohibited-and-restricted-areas>. [Accessed 2024].

- [20] "Ultralytics Enterprise Licensing," Ultralytics, 2024. [Online]. Available: <https://www.ultralytics.com/license>. [Accessed 2024].
- [21] "LoRa developer portal," 2024. [Online]. Available: <https://lora-developers.semtech.com>. [Accessed 2024].
- [22] "Translations of Finnish acts and decrees: 2014," 2014. [Online]. Available: <https://www.finlex.fi/en/laki/kaannokset/2014/en20140917.pdf>. [Accessed 2024].
- [23] "Questions of a general nature," ITU, [Online]. Available: <https://www.itu.int/net/ITU-R/terrestrial/faq/#go13>. [Accessed 2024].
- [24] "THE EUROPEAN TABLE OF FREQUENCY ALLOCATIONS AND APPLICATIONS IN THE FREQUENCY RANGE 8.3 kHz to 3000 GHz (ECA TABLE)," 07 07 2023. [Online]. Available: <https://docdb.cept.org/download/4385>. [Accessed 2024].
- [25] "What is LoRaWAN specification," LoRa Alliance, 2024. [Online]. Available: <https://lora-alliance.org/about-lorawan/>. [Accessed 2024].
- [26] "LoRaWAN Architecture," The Things Network, 2024. [Online]. Available: <https://www.thethingsnetwork.org/docs/lorawan/architecture/>. [Accessed 2024].
- [27] LoRa Alliance Technical Committee Regional Parameters Workgroup, "RP002-1.0.2 LoRaWAN regional parameters," 08 10 2020. [Online]. Available: [https://lora-alliance.org/wp-content/uploads/2020/11/RP\\_2-1.0.2.pdf](https://lora-alliance.org/wp-content/uploads/2020/11/RP_2-1.0.2.pdf). [Accessed 2024].
- [28] "Radio frequency regulation 4 AE/2024M," 16 02 2024. [Online]. Available: [https://www.traficom.fi/sites/default/files/media/file/Radio\\_Frequency\\_Regulation\\_4AE2024M.pdf](https://www.traficom.fi/sites/default/files/media/file/Radio_Frequency_Regulation_4AE2024M.pdf). [Accessed 2024].
- [29] A. Lange and A. Schneeberger, "Brief note on the main broadcasting groups in Finland," European Audiovisual Observatory, 2014.
- [30] "The Things Industries partners with Digita Finland to expand international LoRaWAN® roaming services," Digita Oy, 24 02 2022. [Online]. Available: <https://www.digita.fi/en/news/the-things-industries-partners-with-digita-finland-to-expand-international-lorawan-roaming-services/>. [Accessed 2024].
- [31] "TTN Mapper," [Online]. Available: <https://ttnmapper.org/heatmap/>. [Accessed 2024].
- [32] "IOT LoRaWAN network coverage map," Digita Oy, [Online]. Available: <https://www.digita.fi/en/iot-lorawan-network-coverage-map/>. [Accessed 2024].
- [33] Q. Li, "ASR650X AT Command Introduction," 14 11 2018. [Online]. Available: <https://m5stack.oss-cn-shenzhen.aliyuncs.com/resource/docs/datasheet/unit/lorawan/ASR650X%20AT%20Command%20Introduction-20190605.pdf>. [Accessed 2024].
- [34] "RASPBERRY PI FOUNDATION," [Online]. Available: <https://register-of-charities.charitycommission.gov.uk/charity-search/-/charity-details/5002372>. [Accessed 2024].
- [35] Raspberry Pi Foundation, "Learn to code," Raspberry Pi Foundation, [Online]. Available: <https://www.raspberrypi.org/learn/>. [Accessed 2024].
- [36] Raspberry Pi Ltd., "Raspberry Pi 4 model B," 12 2023. [Online]. Available: <https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-product-brief.pdf>. [Accessed 2024].
- [37] Raspberry Pi Ltd., "Raspberry Pi 5," 10 2023. [Online]. Available: <https://datasheets.raspberrypi.com/rpi5/raspberry-pi-5-product-brief.pdf>. [Accessed 2024].
- [38] NVIDIA Corporation, "NVIDIA Jetson Orin NX Series Modules," 12 2023. [Online]. Available: [https://developer.download.nvidia.com/assets/embedded/secure/jetson/orin\\_nx/docs/Jetson-Orin-NX-Series-Modules-Data-Sheet\\_DS-10712-001\\_v1.1.pdf?CLLiM6tKF\\_yBYyDUOLG35-oV-NZjCYCICloTmmgttMRA1FXxT\\_skr4ovAhqeNN3OQKENWazdJ8DkC4n4IQoomBEirqkMImMZ9j7jDFinYVqeoas1u](https://developer.download.nvidia.com/assets/embedded/secure/jetson/orin_nx/docs/Jetson-Orin-NX-Series-Modules-Data-Sheet_DS-10712-001_v1.1.pdf?CLLiM6tKF_yBYyDUOLG35-oV-NZjCYCICloTmmgttMRA1FXxT_skr4ovAhqeNN3OQKENWazdJ8DkC4n4IQoomBEirqkMImMZ9j7jDFinYVqeoas1u). [Accessed 2024].

- [39] "Compare Products: 2 Products Shown," Oy Farnell (Finland) AB, [Online]. Available: <https://fi.farnell.com/en-FI/catalog/compare?catalogId=15001&langId=80&storeId=10159&catentryId=257283512,343484287&pageType=PDP>. [Accessed 2024].
- [40] Raspberry Pi Ltd, "Raspberry Pi High Quality Camera," 01 2023. [Online]. Available: <https://datasheets.raspberrypi.com/hq-camera/hq-camera-product-brief.pdf>. [Accessed 2024].
- [41] "镜头参数 (Parameter of Lens) ," [Online]. Available: <https://www.farnell.com/datasheets/2938677.pdf>. [Accessed 2024].
- [42] "RASPBerry-PI RPI-HQ-CAMERA," Oy Farnell (Finland) AB, [Online]. Available: <https://fi.farnell.com/en-FI/raspberry-pi/rpi-hq-camera/rpi-high-quality-camera-12-3-mp/dp/3381605>. [Accessed 2024].
- [43] "Global & Rolling Shutters," RED Digital Cinema, LLC., [Online]. Available: <https://www.red.com/red-101/global-rolling-shutter>. [Accessed 2024].
- [44] "Blackfly S USB3," Teledyne FLIR LLC, 2023. [Online]. Available: <https://www.flir.eu/products/blackfly-s-usb3/?model=BFS-U3-12oS4C-CS&vertical=machine+vision&segment=iis>. [Accessed 2024].
- [45] "FAQ," MQTT.org, 2022. [Online]. Available: <https://mqtt.org/faq/>. [Accessed 2024].
- [46] "Node-RED," OpenJS Foundation, [Online]. Available: <https://nodered.org/#features>. [Accessed 2024].
- [47] "Introduction to Node.js," OpenJS Foundation, [Online]. Available: <https://nodejs.org/en/learn/getting-started/introduction-to-nodejs>. [Accessed 2024].
- [48] "Running on Raspberry Pi," OpenJS Foundation, [Online]. Available: <https://nodered.org/docs/getting-started/raspberrypi>. [Accessed 2024].
- [49] T. (. Knight), "node-red-contrib-fs," GitHub, 29 08 2020. [Online]. Available: <https://github.com/TotallyInformation/node-red-contrib-fs>. [Accessed 2024].
- [50] " node-red-dashboard/Charts.md," 22 11 2020. [Online]. Available: <https://github.com/node-red/node-red-dashboard/blob/master/Charts.md#example>. [Accessed 2024].
- [51] nygma2004, "Auto Logger: log data in daily CSV files," 10 09 2020. [Online]. Available: <https://flows.nodered.org/flow/4fo653905120f422cc4f37cd5512ee14>. [Accessed 2024].
- [52] M. Kerrisk, "systemd(1) — Linux manual page," 22 12 2023. [Online]. Available: <https://man7.org/linux/man-pages/man1/systemd.1.html>. [Accessed 2024].
- [53] M. Kerrisk, "crontab(5) — Linux manual page," 22 12 2023. [Online]. Available: <https://man7.org/linux/man-pages/man5/crontab.5.html>. [Accessed 2024].