University of South-Eastern Norway

www.usn.no

TS3000 Bacheloroppgave IRI

Image Processing on the Edge

Group 6 - Aerial Edge

Faculty of Technology, Natural Sciences and Maritime Sciences

Campus Kongsberg

University of South-Eastern Norway

www.usn.no

Course: *TS3000 Bacheloroppgave IRI*
Title: *Image Processing on the Edge*

This report forms part of the basis for assessing the student's performance on the course.

Project group: *Group 6 - Aerial Edge*

Group participants: *SINDRE NES,*
EVEN JØRGENSEN,
ABDUL MAJEED ALIZAI,
ÅDNE KVÅLE,
MARTIN BØRTE LIESTØL,
JON JAHREN

Supervisor: *JAN DYRE BJERKNES*
Project partner: *Kongsberg Defence & Aerospace*

Summary:
Temporary placeholder

1 Acknowledgements

2 Abstract

Kongsberg, 15th May 2023

Contents

1 Acknowledgements **4**

2 Abstract **5**

Contents **7**
List of Figures 8

3 Introduction **9**
3.1 Introduction 9
3.2 Problem Domain 9
3.3 Research Perspective 10

4 Process **12**
4.1 Process 12
4.2 Project Tools 13
4.2.1 Taiga 13
4.2.2 Github 13
4.2.3 Overleaf 13
4.2.4 ChatGPT 14
4.2.5 Microsoft Office 14
4.2.6 Microsoft Teams 14
4.3 Risk Analysis 15

5 Method **18**
5.1 Initial Proposals 19
5.2 Operating System and Software Platform 20
5.3 Selected Hardware Configurations 21
5.3.1 nVidia Jetson Nano 22
5.3.2 Coral Edge TPU 22
5.3.3 Raspberry Pi 4B & Zero 2 23
5.4 Selected Cameras 23
5.4.1 Pi Camera module v2 23
5.4.2 Pi Camera module v3 24
5.5 Selected Software Configuration 24
5.5.1 Operating System 24

5.5.2 Container . . . 24
5.5.3 Robot Operating System (ROS) . . . 24
5.6 Architectures . . . 24
5.6.1 Configuration 1, Jetson Nano . . . 25
5.6.2 Configuration 2, Pi 4 w/ Coral Edge TPU . . . 30
5.6.3 Configuration 3, Pi Zero w/ Coral Edge TPU . . . 31
5.6.4 Configuration 4, Pi 4 . . . 31
5.7 Technical overview . . . 32
5.7.1 Test bench for distributed architectures . . . 33

6 Measurements **34**

References **37**

A Operating System Architecture **42**
A.1 Operating System Selection Process . . . 42

B Algorithms **47**
B.1 Distance and Color detection . . . 47

C Team **49**

List of Figures

3.1 Hermenutic circle [3] 11

4.1 Taiga Interface 13
4.2 Risk table 16
4.3 Risk Matrix [7] 17

5.1 Initial architecture design 19
5.2 Proposed Operating System Configuration 21
5.3 Jetson nano [8] 22
5.4 Coral USB accelerator [9] 22
5.5 Hardware architecture, config 1 25
5.6 Deepstream app config file, source0 27
5.7 config_infer_primary_YoloV5.txt, custom model 27
5.8 Hardware architecture, config 2 30
5.9 Software, config 2 30
5.10 Hardware architecture, config 3 31
5.11 Hardware architecture, config 4 31
5.12 Four distributed and one centralized architecture 32
5.13 Dataflow during test scenario 33

A.1 Decision Process OS 45
A.2 Decision Process Distribution 46

Symbol Explanation

3 Introduction

3.1 Introduction

We are a team of six computer engineering students from the University of South-Eastern Norway, Campus Kongsberg. Our bachelor assignment was given to us by Kongsberg Defence & Aerospace (KDA), a Norwegian technology company headquartered in Kongsberg. KDA specializes in manufacturing equipment for defense, space exploration, and aviation.[1]

Our client conducts a student-centered initiative known as 'Local Hawk', which is operational during the summer. The main focus of this initiative is to investigate a variety of methods for fostering the development of autonomous drones. The client has expressed an interest in our project with the aim of garnering insightful data that could be applied to future deliberations concerning the architectural design of these unmanned aerial vehicles.

The drone systems traditionally used in the Local Hawk project has limited computing power and restrictions on weight. KDA expressed an interest in doing a research project for our assignment, where we would examine any potential performance gains by moving the processing closer to the sensor hardware, meaning we will be using dedicated hardware for image processing.

3.2 Problem Domain

Today, the drones being used in the Local Hawk project, uses what we call Single Board Computers (SBC). These devices are usually created in a small form factor, and due to their limited size, they are also limited in processing capabilities. This means that it has challenges performing several tasks simultaneously, e.g., controlling the drone motors and executing object detection at the same time.

In earlier iterations of Local Hawk they have attempted to run object detection while flying the drone at the same time on a single SBC, this resulted in very low framerate which in turn meant that it could not be used for any meaningful purpose. The reason for this is because processing camera images can be computationally expensive. For instance, if we need to process a 24-bit color image with a 600x600 resolution pixel-by-pixel, we would have to handle 360,000 pixels, each with 3 color channels, resulting in an input data size of 1,080,000 bytes per image. This poses a challenge for the limited hardware available for a lightweight drone.

This study aims to explore multiple software and hardware architectures for a small UAV with object detection capabilities. The architectures will be compared in terms of performance, cost, complexity and weight. Our results will provide recommendations on which technology to use when building lightweight UAVs for various use cases which require object detection.

3.3 Research Perspective

In our initial discussions with the customer, it was clearly communicated that they desired a research report containing actionable information for decision-making, provided our findings indicate the potential for a successful endeavor.

This presented challenges for us, as we lacked prior experience with research-focused projects. Consequently, we needed to adapt our approach in order to comprehend how best to organize and execute our work, ensuring optimal delivery of results.

During the initial stages of the project, we deliberately refrained from immediately engaging with specific technologies or embarking on the development of a product in a domain where our knowledge was limited. Instead, we adopted a hermeneutic process, which is not commonly employed in engineering or systems engineering. However, this methodology proves valuable when dealing exclusively with knowledge and information, rather than specific implementations.[2]

The diagram of the hermeneutic circle presented below illustrates the practical application of this concept. As our understanding of the problem domain deepens, we gain the ability to formulate more refined inquiries regarding the information that is relevant to us and the customer. This enhanced understanding guides us in approaching the problem domain with the aim of achieving the desired results.

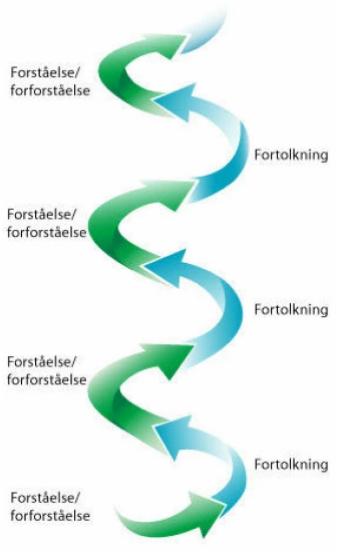


Figure 3.1: Hermenutic circle [3]

4 Process

4.1 Process

After our initial phase where we spent a lot of time trying to adjust our perspective in order to understand how to work with a research assignment, we started to create an implementation of an actual working project process with this in mind.

We decided that our project demands were best met using an agile methodology, this made it possible for us to rapidly change course should the need arise, and it made it easy to follow up closely with each individual task in case someone in the group found a task difficult or impossible to complete requiring us to course correct swiftly.

Following the agile methodology, we decided to have daily stand-up meetings where we would update the group on our individual progress. We organized our sprints to last one week at a time, where we had an initial meeting on Monday to set up our tasks for the week, and a meeting at the end of the week where we updated our client on where we were in terms of progress. In the end of week meetings the client was invited to give feedback on which tasks they wanted us to prioritize going forward into the next sprint.

In practice, we implemented a system where we organized our team using a web framework called Taiga, which enabled us to create sprints and create user stories within those sprints, giving us the opportunity to track our work. In addition we linke our Taiga software to github, making it possible to track code changes against specific tasks in a sprint, in case that was needed.

4.2 Project Tools

4.2.1 Taiga

In order to organize our agile workflow we have elected to use Taiga. It is a web based tool which allows us to track sprints, tasks within sprints, and assign aforementioned tasks to specific members of the group and additionally allow members to follow updates on a specific task. In addition to this, Taiga has integration with GitHub, allowing us to modify tasks from GitHub whenever we commit code in our repository there.

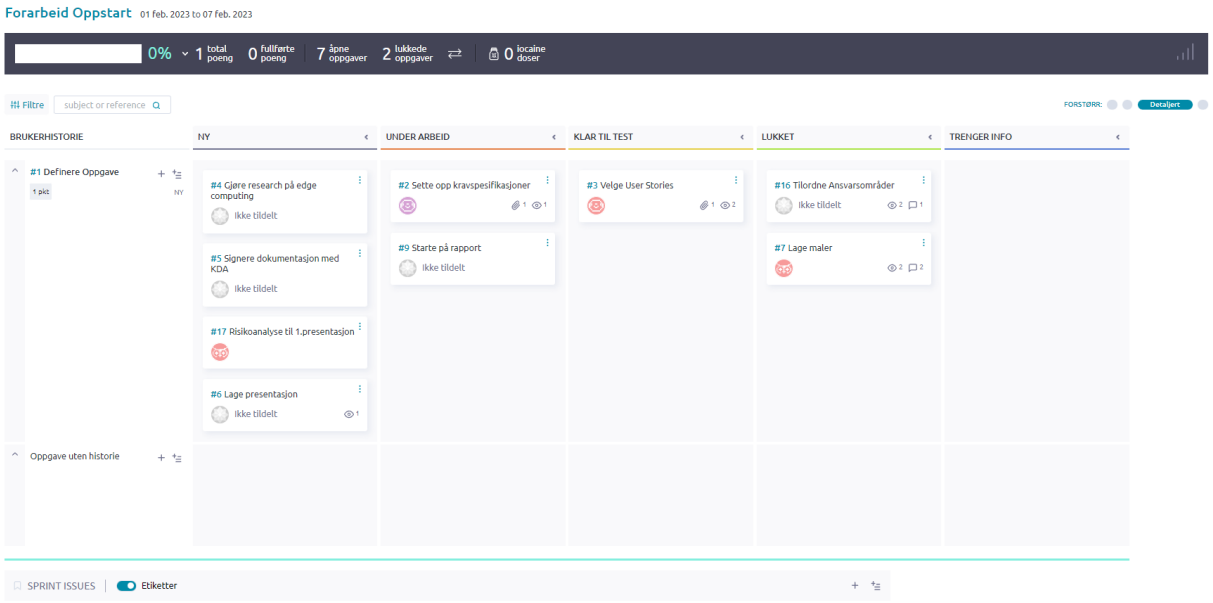


Figure 4.1: Taiga Interface

4.2.2 Github

Our code and information will be kept in a GitHub repository where we are able to organize our code, docker configuration and accompanying documentaion.

4.2.3 Overleaf

Our report is written in LaTeX, it is kept on Overleaf, allowing us to collaborate and write simultaneously in one document. It also keeps record of everyone who makes changes to the document, and allows us GitHub integration to do version control.

4.2.4 ChatGPT

ChatGPT is an artificial intelligence language model, released to the public recently. It is designed to understand natural language and generate human-like responses to a wide range of prompts and questions.

ChatGPT is most useful for tasks that require natural language processing, such as language translation, sentiment analysis, text summarization, and conversational interfaces. It can also be used for a variety of other applications, such as content generation, language modeling, and knowledge extraction.

We have on occasion used ChatGPT for cleaning up and helping us formulate language in a more academic and formal fashion.

4.2.5 Microsoft Office

Documents that was impractical to keep or create in LaTeX, e.g., spreadsheets, were created and maintained in Microsoft Office.

4.2.6 Microsoft Teams

Whenever we were unable to meet physically or when we needed to have an easily accessible place to store documents we used where using git would be suboptimal, we decided to organize our group through Microsoft Teams.

4.3 Risk Analysis

Risk analysis is an ongoing process that continuously evaluates risk throughout the project's lifecycle. It is the responsibility of the risk manager to ensure that this process takes place regularly and consistently during the project. This is crucial because it raises awareness among both us and our customers about potential risks and vulnerabilities, encourages necessary improvements, and facilitates necessary changes. Such analysis can help the risk manager identify new risks and changes that require attention along the way. [4]

After identifying risks, it is essential to prioritize them based on their probability and severity. Moreover, measures should be put in place to manage them effectively if they occur. While everyone on the team should participate in assessing the project's risks, the risk manager will be primarily responsible for ensuring quality assurance. [5]

A risk analysis was conducted for our project, wherein we identified both internal and external risks. Internal risks are linked with factors that are under our control, whereas external risks are associated with factors that lie beyond our control.[6]

After identifying the risks at this stage, we evaluate their potential consequences and determine the appropriate measures that can be taken if they occur. To gain a better overview, we record the risks in a table that includes a unique code, a description of the risk event, recommended measures, probability (P), consequence (C), and priority. Below is a comprehensive overview of both internal and external risks:

Code	Risk event	Consequence	Measures	P	C	Priority
R1	If multiple gets sick at the same time?	Delay and loss of work. More work for those who are available. High stress level.	Minimize loss of work by saving often. Work a few extra hours during that period and actively use Taiga to distribute tasks to stay on track. Lower stress levels by having good communication and planning within the group.	Unlikely	Critical	10
R2	If someone suddenly wants to quit?	Delay, poor solutions, and lower trust within the group.	Being honest, follow-up in stand-up meetings, and actively using out dashboard.	Rare	Critical	5
R3	Various problems with group work.	Delays, poor communication, attendance.	Good work method, communication, and status meetings. Social gathering.	Moderate	Major	12
R4	The group is not able to achieve an optimal solution as the client envisioned.	Bad results (grade / further understanding of the problem). Bad conscience / self-confidence.	Maintain good communication with the client. Have a good working process so that any errors can be detected early.	Unlikely	Critical	10
R5	New nationwide pandemic.	Lockdown so that we do not meet physically. The chance of more people getting infected and becoming sick.	Work remotely, use Teams and other tools available actively. Have good hygiene habits.	Unlikely	Major	8
R6	Global component shortages.	Delays in work/test and results.	Ask the client about components / alternative components. Find alternative solutions.	Rare	Insignificant	1
R7	The war in Ukraine.	Increased fuel prices causing hesitation to drive to school.	Driving together or using public transportation.	Unlikely	Minor	4
R8	Issues related to software drivers and versions	We may be unable to answer the customer's	Use older versions	Likely	Minor	8

Figure 4.2: Risk table

The prioritization of these risks was determined by evaluating their consequence and probability using the risk matrix:

		Consequence				
	Risk	Insignificant	Minor	Significant	Major	Critical
	Almost certain	5	10	15	20	25
	Likely	4	8	12	16	20
Probability	Moderate	3	6	9	12	15
	Unlikely	2	4	6	8	10
	Rare	1	2	3	4	5

Risk criteria	
Level	**Measure**
High	Unacceptable, and action must be taken immediately.
Medium	We need to implement measures to reduce risk.
Low	Acceptable, but measures should be taken where feasible

Figure 4.3: Risk Matrix [7]

5 Method

In this chapter we will shed a light on the path we took, the decisions we made, and the progression of our project, with a primary focus on the architectures that underpinned our work.

Our first step was to present a set of potential architectures to our client. These architectures, which form the initial part of this chapter, will be given a cursory overview. The purpose of this part is to give a glimpse of the breadth of options we considered before narrowing down our focus.

After thorough discussions and iterations with our client, we arrived at a consensus on the architectures that held the most potential for our client's interests. This agreement marked a significant turning point in our project, as it enabled us to channel our efforts in a focused direction.

After deciding which architectures we were going to proceed with, we will outline and detail the decisions that went into each architectural design. This will give a clear picture on how and why we selected the hardware and software that comprises our system.

5.1 Initial Proposals

Before we could start implementing our specific designs we needed to get the go-ahead from our client. We made a design sketch with five different configurations, and presented these in order to reach an agreement on which areas we should focus on.

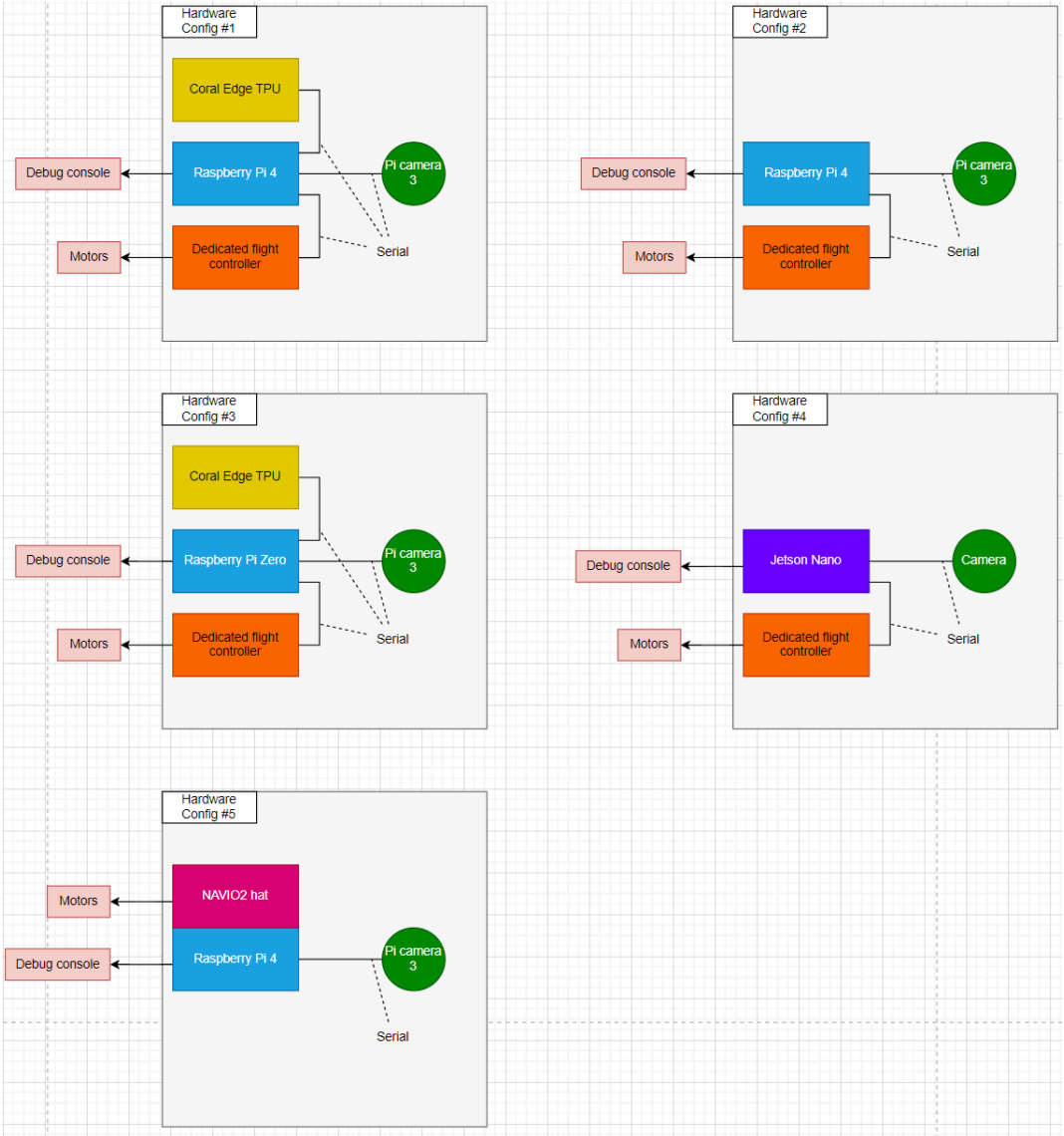


Figure 5.1: Initial architecture design

From the figure 5.1 in this figure, you can see our five proposals which we presented to the client. These proposals were based on a variety of hardware configurations we had briefly discussed during meetings, and the ones that we thought were the most likely to achieve our clients goals and deliver results that would show that this idea was viable.

5.2 Operating System and Software Platform

In the pursuit of rigorous research and robust information, the significance of a controlled environment and reproducible results cannot be overstated. Absent these, the applicability and value of our findings risk being limited, particularly under rigorous scrutiny. A crucial component of establishing this controlled environment involves ensuring uniformity in our software across all configurations to the greatest feasible extent. This is to prevent the introduction of performance artifacts that could significantly impact performance, such as inconsistencies in software libraries or versions, including the kernel or the C-library version.

Our objective was to arrive at an informed decision, ensuring we did not default to the standard software without assessing potential alternatives. With regard to the operating system functioning on the hardware, preliminary testing was conducted using several Linux distributions. This was to determine the most suitable system for meeting our stringent requirements. The rationale underpinning this strategy was to ascertain uniformity across all hardware configurations in terms of the system operating on the bare metal. The decision to opt for Linux was informed by several factors. Firstly, it is the system included with the Raspberry Pi Single Board Computers (SBCs) that we have employed for drone control. Additionally, its rapid development model enhances the probability of our required software being supported or easily installable. Lastly, it guarantees hardware support.

The Linux distributions we examined to assess their suitability for our project included Fedora, OpenSUSE, and Ubuntu Linux, these were all examined in addition to the vendor-supplied Raspberry Pi OS, which is based on Debian 11. Following thorough testing and troubleshooting, we elected to continue with the default operating system provided with the Raspberry Pi, a Debian-based distribution specifically crafted for the Raspberry Pi.

Upon settling on the Raspberry Pi OS, we resolved to run our software in a container to maintain identical and unaltered software configurations across different settings. This approach further strengthens our commitment to ensuring an environment conducive to

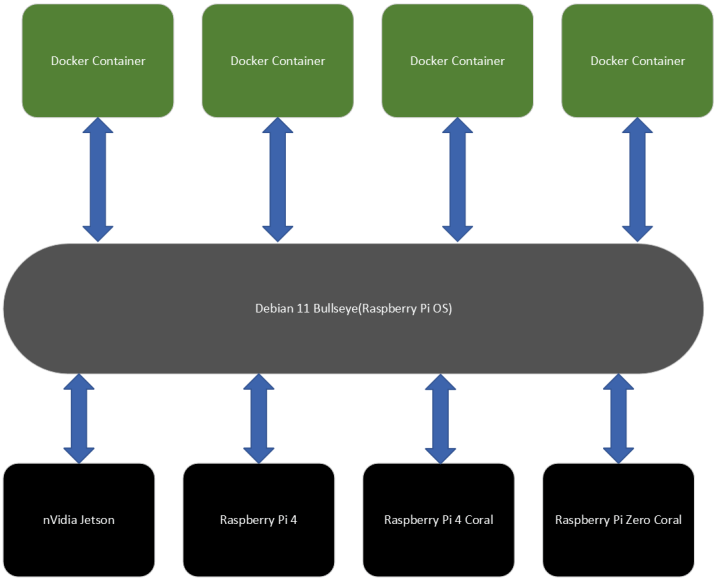


Figure 5.2: Proposed Operating System Configuration

consistent and reliable research outcomes.

Our proposed configuration can be seen in figure 5.2, showing that we wanted a uniform platform across all our proposed configurations.

5.3 Selected Hardware Configurations

After presenting the five configurations in the figure 5.1, the client initially selected three different versions, with slight modifications. As their wish was for us to look closer at doing sensor readings on the edge, we were instructed to create a more decentralized approach, where we separate functionality between the edge configuration, and we keep one central configuration similar in all different versions.

Previous iterations of the Local Hawk UAV can navigate autonomously using GPS, IMU and a barometer. They achieved this by using a NAVIO2 hat for the Raspberry Pi 4, which supplies all the sensors. Ardupilot, which is a UAV flight control software suite was running directly on the Pi 4. Introducing additional heavy computations to the Pi 4's CPU by running object detection as well was something our client wanted to avoid. We were asked to focus on distributed solutions where the image processing is performed on dedicated hardware. Early on in the project we selected various hardware components for our UAV architectures

5.3.1 nVidia Jetson Nano

The Jetson Nano is a single board computer with a built-in NVIDIA GPU with 128 CUDA cores. We chose the Jetson because the GPU can be leveraged for high performance deep learning based object detection(VEDLEGG/REF SOM FORKLARER HVORFOR DET ER NICE). Performing image processing on a GPU has the added benefit of freeing up the CPU for other tasks.

Figure 5.3: Jetson nano [8]

5.3.2 Coral Edge TPU

Google's Coral Edge TPU is an ASIC, or an "application specific integrated circuit". The application in question is deep learning inference. Much like the Jetson Nano's built in GPU this piece of hardware is thought to perform deep learning based object detection faster than any single board computer's CPU. The Coral Edge TPU is a USB device which requires a computer to operate.

Figure 5.4: Coral USB accelerator [9]

5.3.3 Raspberry Pi 4B & Zero 2

	RPi 4B	RPi Zero2
CPU	Cortex-A72 @ 1.5GHz	Cortex-A53 @ 1GHz
GPU	VideoCore IV @ 500MHz	VideoCore IV @ 400MHz
Memory	1 GB - 8 GB	512 MB
Video decoding	H.264/H.265 (4Kp60)	H.264 (1080p30)
Video encoding	H.264/H.265 (1080p30)	H.264 (1080p30)
Connectivity	USB 3.0 x 2 USB 2.0 x 2 UART x 1 SPI x 2 I2C x 1	USB 2.0 x 1 UART x 1 SPI x 2 I2C x 1
Form factor	85mm x 56mm	65mm x 30mm
Weight	46g	11g
MSRP	$ 35 - $ 75	$ 15

5.4 Selected Cameras

5.4.1 Pi Camera module v2

The Raspberry Pi Camera Module v2 was introduced in April 2016 to replace the original Camera Module. The v2 Camera Module is equipped with a Sony IMX219 8-megapixel sensor, which represents a significant upgrade from the 5-megapixel OmniVision OV5647 sensor found in the original camera. The Camera Module is capable of capturing high-definition video and still photographs. It is user-friendly for beginners, but also offers advanced features for users seeking to expand their knowledge. Online examples showcase the camera's versatility for time-lapse, slow-motion, and other video applications, while libraries provided with the camera facilitate the creation of special effects.

Further details about the IMX219 and the Exmor R back-illuminated sensor architecture are available on Sony's website, underscoring that the camera's improved resolution represents a leap forward in image quality, color fidelity, and low-light performance. The Camera Module supports 1080p30, 720p60, and VGA90 video modes, in addition to still capture. It attaches via a 15cm ribbon cable to the Camera Serial Interface (CSI) port on the Raspberry Pi, and is compatible with all models of Raspberry Pi 1, 2, and 3. The camera can be accessed through the Multi-Media Abstraction Layer (MMAL) and Video for Linux (V4L) APIs, and numerous third-party libraries are available, such as the Picamera Python library. A "Getting Started with Picamera" resource is available for users seeking guidance on its use. The Camera Module is widely employed in home

security applications, as well as in wildlife camera traps.[10][11]

5.4.2 Pi Camera module v3

The Raspberry Pi Camera Module 3 is a compact camera designed by Raspberry Pi, equipped with a 12-megapixel sensor featuring high dynamic range (HDR) and phase detection autofocus. The camera is available in standard and wide-angle variants, both with or without an infrared cut filter.

Capable of capturing full HD video and still photographs, the Camera Module 3 includes an HDR mode for up to 3 megapixels. Its operation is fully supported by the libcamera library, and its rapid autofocus feature makes it accessible to beginners while providing ample functionality for advanced users. The Camera Module 3 is compatible with all Raspberry Pi computers, and features the same printed circuit board (PCB) size and mounting holes as its predecessor, the Camera Module 2. The only difference in dimension is the height, with the improved optics making the Camera Module 3 several millimetres taller than its predecessor.[12]

5.5 Selected Software Configuration

5.5.1 Operating System

5.5.2 Container

5.5.3 Robot Operating System (ROS)

5.6 Architectures

5.6.1 Configuration 1, Jetson Nano

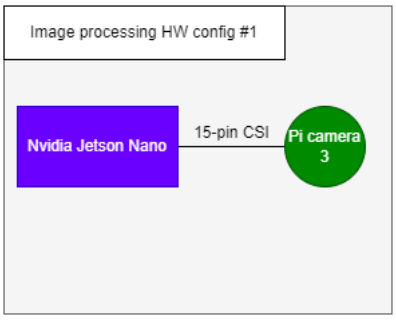


Figure 5.5: Hardware architecture, config 1

Software

The NVIDIA Jetson Nano platform employs a software development kit (SDK) image known as JetPack, specifically version 4.6.1, which is the newest compatible version for the Jetson Nano device. JetPack 4.6.1 is built upon Ubuntu 18.04 (Bionic Beaver) and utilizes Python 3.6.9. The image is preconfigured with several essential developer tools, including TensorRT, cuDNN, and CUDA.

To flash the image onto the Jetson Nano, one can either use a terminal or the NVIDIA SDK Manager, which can be accessed at [13]. It is strongly recommended to employ the NVIDIA SDK Manager for this task, as it significantly simplifies the installation process for the image and any additional SDKs. However, this method necessitates that the computer used for flashing possesses the same operating system as the target image. In the present case, the computer required reformatting to Ubuntu 18.04 to ensure compatibility.

Alternatively, the SD card can be directly flashed using the terminal. The necessary image can be obtained from the NVIDIA developer site [14]. Subsequent instructions specific to your operating system can be found at [15].

Following this, the NVIDIA SDK Manager can be used to install all additional SDKs via Secure Shell (SSH). This approach upholds the user-friendliness and efficient process inherent to the SDK Manager.

Yolo with deepstream

optional way of running yolo.

The process of deploying YOLOv5 on the Jetson Nano platform proved to be more complex compared to the implementation of MobileNet-SSD. The YOLOv5 documentation provides a dedicated page for its application on a Jetson Nano using a Software Development Kit (SDK) from Nvidia named Deepstream. However, the tutorial [16], last updated 18 November 2022) is not as straightforward as it might appear. For the purpose of this study, the deployment of YOLOv5 on the Jetson Nano was accomplished through two distinct approaches, both of which will be thoroughly explored in the following sections.

The initial step requires the use of a PC running Ubuntu 18.04 to operate the Nvidia SDK Manager in conjunction with the Jetson Nano. Following this, the installation of several additional packages, one of which is Deepstream, is necessary. This part of the procedure is fairly linear and should already be done with the initial setup of the Jetson.

The challenge arises when attempting to run YOLO, since YOLOv5 necessitates Python 3.7, while the Jetson Nano only supports Python 3.6.9, this is because the installation was done on a Jetson After cloning the repository, all entries in the requirements.txt file were commented out. Subsequent to numerous trials and errors, the tested system runs the following versions.

Table 5.1: pip packages used for deepstream

Python package	Version number
gitpython	3.1.20
matplotlib	3.3.4
numpy	1.19.5
opencv-python	4.1.1
Pillow	7.1.2
psutil	
PyYAML	6.0
requests	2.18.4
scipy	1.5.4
thop	0.1.1
tqdm	4.64.1
seaborn	0.11.0
setuptools	59.6.0

Continuing from this point, the subsequent steps as outlined on the GitHub repository should be followed until the DeepStream Configuration for YOLOv5 ”Step 4. Generate the cfg and wts files”. This can be accomplished with the following Python3 commands:

python3 gen_wts_yoloV5.py -w yolov5s.pt
python3 gen_wts_yoloV5.py -w custom.pt

Here, the user has the flexibility to either convert the pretrained yolov5s model or replace it with any model that has been subjected to transfer learning, as demonstrated above.

After completing the remaining steps, the user should be able to run inference on the included video. Additionally, to run inference on a Camera Serial Interface (CSI) camera, as employed in this study, the user must modify the source0 parameter in the deepstream_app_config.txt file as follows:

```
[source0]
enable=1
type=5
# uri=file:///opt/nvidia/deepstream/deepstream/samples/streams/sample_1080p_h264.mp4
camera-width=640
camera-height=480
camera-fps-n=30
camera-fps-d=1
camera-csi-sensor-id=0
```

Figure 5.6: Deepstream app config file, source0

DeepStream offers built-in support for CSI cameras, with type=5 indicating the usage of a CSI camera. The camera's width and height can be adjusted according to the user's needs. It should be noted, however, that the model is optimized for an input resolution of 640x640 pixels, so providing an input close to this resolution would potentially enhance performance.

It is also important to modify the config-file under primary-gie, set this to config_infer_primary_yoloV5.

To use a custom model, the configuration file must be modified in line with the particular version of YOLO in use, hence necessitating amendments to the 'config_infer_primary_yoloV5.txt' file. The modifications include changes to the following parameters:

```
[property]
gpu-id=0
net-scale-factor=0.0039215697906911373
model-color-format=0
custom-network-config=yoloV5_custom.cfg
model-file=yoloV5_custom.wts
```

Figure 5.7: config_infer_primary_YoloV5.txt, custom model

In this context, the weight (wts) and configuration (cfg) files have been renamed as 'yoloV5_custom.cfg' and 'yoloV5_custom.wts', respectively, indicating their customized nature and their association with the YOLOv5 model.

Now to run inference, simply use the command: deepstream-app -c deepstream_app_config.txt

As seen on the ultralytics page, it is possible to achieve 30 fps with a Jetson Xavier NX with FP32 and yoloV5s model. The jetson nano is weaker than the Xavier when it comes to hardware, and we were able to achieve 10 FPS with the yolov5n (nano) model.

We did however meet a problem 400 ms latency on the inference stream, after monitoring the system resources and being unable to solve the latency problem, we concluded that this method was to hard on the jetson nanos hardware, and wasnt suited to being used in a drone which require real-time detection.

The Deepstream method seems to work fine and has a wide range of customizability already built with ease of access trough the configuration files. If we had one of the newer jetson modules, maybe the outcome would have been different.

Mobilenet-SSD

Jetson Inference, a GitHub repository developed by Dustin Franklin from Nvidia, provides a well-documented tutorial complete with video walkthroughs, which is particularly beneficial for beginners in the field of computer vision working on the Jetson device.

The pretrained model demonstrated satisfactory performance at shorter distances but struggled with object detection when the objects were small or located at greater distances. To address this, transfer learning was applied to the model using our dataset, both on the Jetson device and on a separate computer equipped with a dedicated GPU. Despite training the model for over 1000 epochs, the inference failed to detect the object.

A survey of online forums confirmed that the model encountered difficulties with small objects. An attempt was made to upgrade the model to accommodate an input resolution of 512x512 instead of the standard 300x300. This was tried, and while this modification slightly improved the model's performance, it did not reach the level of the YOLO model trained earlier in the study. As a result, a decision was made to revisit YOLOv5.

5.6.2 Configuration 2, Pi 4 w/ Coral Edge TPU

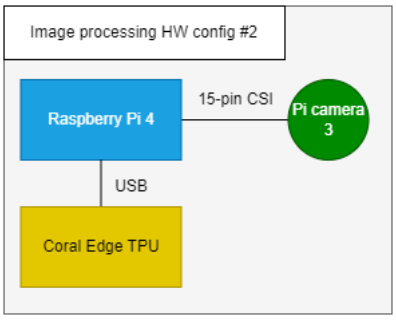


Figure 5.8: Hardware architecture, config 2

Software

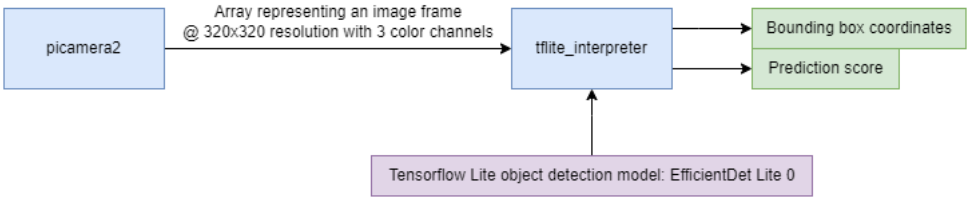


Figure 5.9: Software, config 2

The picamera2 python library provides an interface to the Raspberry Pi camera drivers and allows us to capture image frames as multidimentional arrays. The tflite_interpreter class provides methods for setting up and running inference with our trained object detection model. The model takes the image array as input, and outputs the coordinates of bounding boxes around any detected object as well as a prediction score per detected object. The score describes how likely it is that the detection is a true positive.

ELLER:
Camera interface: picamera2 python library
Object detector type: Deep neural network
Machine learning framework: Tensorflow Lite
Model type: INT8 Tensorflow Lite model (.tflite) compiled for Coral Edge TPU
Model input: 320x320x3 array
Model output: Bounding boxes, prediction scores, class names, number of predictions
Model size: 4MB

5.6.3 Configuration 3, Pi Zero w/ Coral Edge TPU

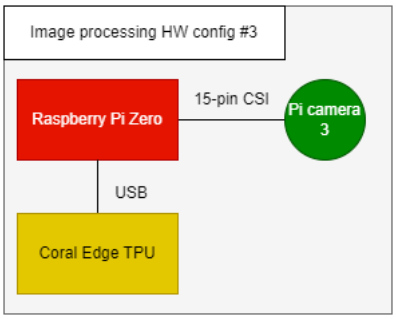


Figure 5.10: Hardware architecture, config 3

5.6.4 Configuration 4, Pi 4

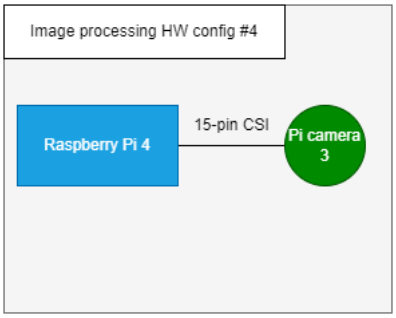


Figure 5.11: Hardware architecture, config 4

Software

When considering the absence of acceleration hardware, we needed to carefully select an appropriate image-processing algorithm for our research project. In this section, we will talk about why we chose a certain algorithm and how it helps with object detection on our hardware. We will also go over the difficulties we came across and the steps we took to progress despite them.

The algorithm we chose for this configuration is called blob detection. It is also known as blob analysis or blob tracking and is an image processing technique that focuses on identifying and analyzing distinct regions of interest in an image. For our project, the blob detection algorithm is tasked with detecting balls of different colors, relying on their HSV color values and contour properties for distinction.

Our choice of the blob detection algorithm was motivated by its versatility and adaptability. The algorithm's capability to be finely tuned to detect specific objects, its high level of modifiability to suit different computational requirements, and its suitability for the limited processing power of the Raspberry Pi 4 were important factors in our decision. These benefits made it a suitable choice for our project, as we aimed to balance detection accuracy and computational efficiency.

5.7 Technical overview

Our study aims to evaluate and compare four distinct distributed drone architectures, along with one centralized architecture. The distributed architectures solely vary in their approach to image processing. The leftmost diagram in figure 5.12 illustrates a high-level overview of the four distributed architectures, wherein each box represents hardware with processing capabilities. The rightmost box represents the centralized architecture, where all the processing tasks are performed on a single hardware unit.

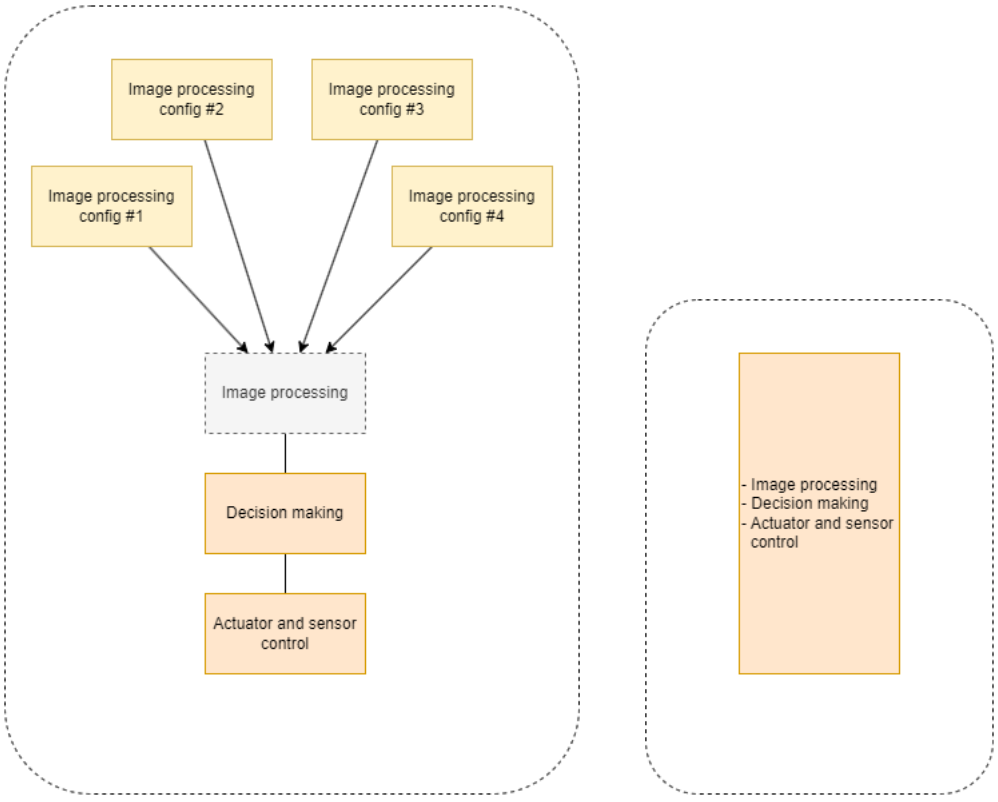


Figure 5.12: Four distributed and one centralized architecture

5.7.1 Test bench for distributed architectures

This section will cover the commonalities between the distributed architectures.

Figure 5.13 describes the test setup with internal interfaces. This setup contains all the components needed for a drone with autonomous capabilities, apart from the chassis and actuators which are not needed to measure data processing performance. As described in figure 5.13 all four image processing configurations have the same output format. They publish distance and position data to ROS2 topics which the decision making module subscribes to. This common interface enables testing of the different image processing configurations without making any changes to the remaining modules that make up the drone.

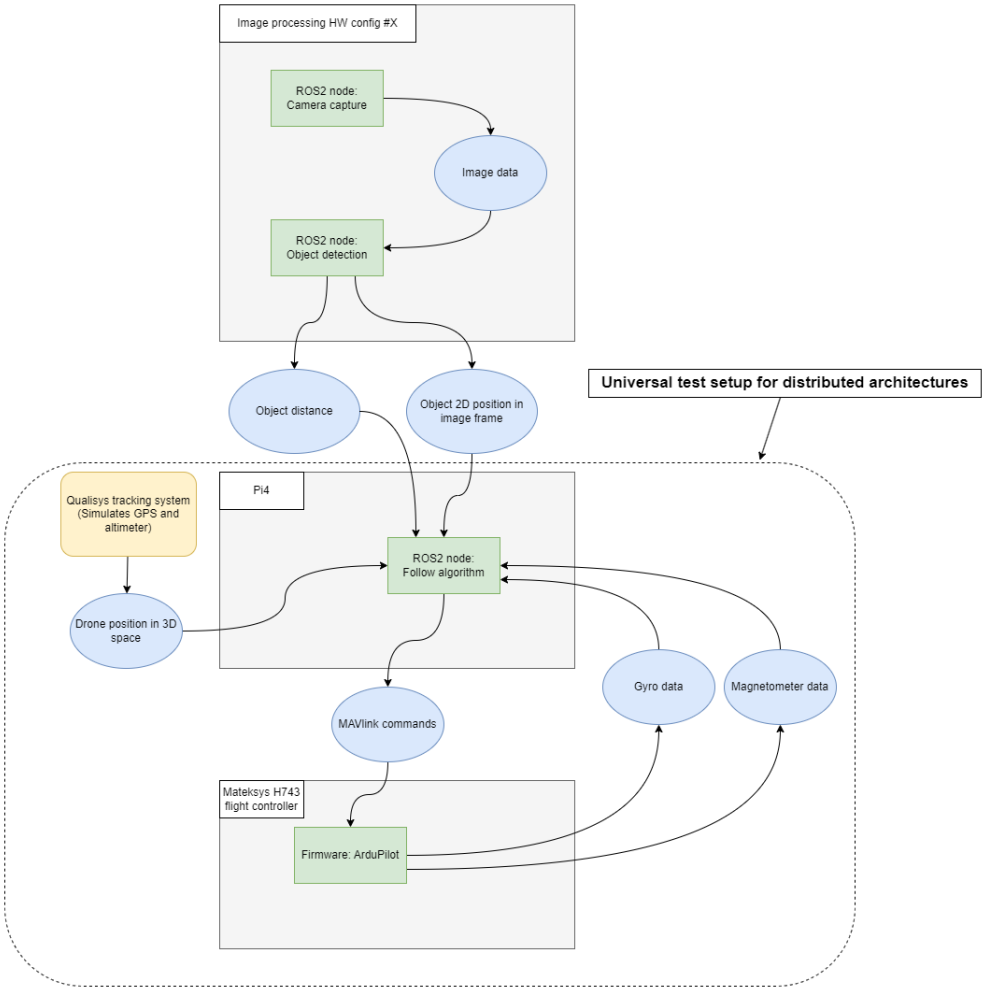


Figure 5.13: Dataflow during test scenario

6 Measurements

Performance Measurement Results

The purpose of this section is to evaluate and compare the performance of the four different hardware configurations. This assessment will help us determine the most suitable configuration for achieving the desired balance between processing power, accuracy, and energy efficiency.

Detection accuracy (precision, recall, F1-score)

- Precision: Precision is a measure of how many of the detected objects are actually relevant. It is calculated as the number of true positives (TP) divided by the sum of true positives and false positives (FP). A high precision indicates that the object detection system is good at identifying relevant objects while avoiding false detections. Precision = TP / (TP + FP)
- Recall: Recall is a measure of how many of the relevant objects are detected by the system. It is calculated as the number of true positives (TP) divided by the sum of true positives and false negatives (FN). A high recall indicates that the object detection system is good at finding all the relevant objects in the scene. Recall = TP / (TP + FN)
- F1-score: The F1-score is the harmonic mean of precision and recall, providing a single metric that balances both precision and recall. This is useful when you want to compare the performance of different object detection systems, especially when there's a trade-off between precision and recall. An F1-score closer to 1 indicates a better-performing object detection system. F1-score = 2 * (Precision * Recall) / (Precision + Recall)

Processing time

- Processing time refers to the duration it takes for the system to process a single frame or image for object detection. It is measured in milliseconds (ms) and represents the time elapsed from when the frame is received by the edge device until the object detection results are produced.
- Lower processing times indicate better performance, as the system can analyze more frames in a given period. This can be crucial for a drone application, where real-time or near-real-time object detection is often necessary for smooth operation and quick decision-making.

Power consumption

- Power consumption refers to the amount of electrical power used by the edge configurations while performing object detection tasks. It is measured in watts (W) and is obtained by monitoring the current and voltage supplied to the device during operation.
- Lower power consumption is generally more desirable, as it indicates higher energy efficiency and can result in longer flight times for the drone. Comparing the power consumption of the four configurations can help determine which option is better suited for a lightweight drone, where battery life and weight are critical factors.

Table 6.1: Configuration 1

Configuration	Jetson Nano with Yolov5
Avg Processing time (ms)	100
Frames per second (FPS)	10
Power Consumption (W)	?

Table 6.2: Detection Accuracy

Configuration	Precision	Recall	F1-score
Jetson Nano with Yolov5			
Blob Detection	0,624	0,252	0,353

References

[1] KongsbergGruppen. 'Om kongsberg gruppen.' (), [Online]. Available: https://www.annual-report.kongsberg.com/no/om-kongsberg-gruppen/dette-er-kongsberg-gruppen/. (accessed: 05.02.23).

[2] S. N. Leksikon. 'Dialektikk.' (), [Online]. Available: https://snl.no/dialektikk. (accessed: 08.02.23).

[3] M. Friis-Mikkelsen. 'Hva er hermeneutikk.' (), [Online]. Available: https://forskning.no/filosofiske-fag/hva-er-hermeneutikk/722732. (accessed: 05.02.23).

[4] L. Rosencrance. 'What is risk analysis?' (), [Online]. Available: https://www.techtarget.com/searchsecurity/definition/risk-analysis. (accessed: 20.03.23).

[5] V. Solutions. 'Risk matrix calculations – severity, probability, and risk assessment.' (), [Online]. Available: https://www.vectorsolutions.com/resources/blogs/risk-matrix-calculations-severity-probability-risk-assessment/. (accessed: 20.03.23).

[6] H. Underwood. 'Outline risk identifying risks to your project change managing change for your project.' (), [Online]. Available: https://slideplayer.com/slide/11367849/. (accessed: 20.03.23).

[7] P. Guevara. 'A guide to understanding 5x5 risk matrix.' (), [Online]. Available: https://safetyculture.com/topics/risk-assessment/5x5-risk-matrix/. (accessed: 20.03.23).

[8] Nvidia. 'Jetson nano developer kit.' (), [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-developer-kit. (accessed: 18.03.23).

[9] Coral.ai. 'Usb accelerator.' (), [Online]. Available: https://coral.ai/products/accelerator. (accessed: 18.03.23).

[10] LCSC. 'Camera module v2.' (), [Online]. Available: https://datasheet.lcsc.com/lcsc/1810010717_Raspberry-Pi-RPICAMERABOARD_C110649.pdf. (accessed: 20.03.23).

[11] R. P. Ltd. 'About the camera modules.' (), [Online]. Available: https://www.raspberrypi.com/documentation/accessories/camera.html. (accessed: 20.03.23).

[12] R. P. Ltd. 'Raspberry pi camera module 3.' (), [Online]. Available: https://datasheets.raspberrypi.com/camera/camera-module-3-product-brief.pdf. (accessed: 20.03.23).

[13] Nvidia. 'Nvidia sdk manager.' (), [Online]. Available: https://developer.nvidia.com/sdk-manager. (accessed: 12.05.23).

[14] Nvidia. 'Jetpack sdk 4.6.1.' (), [Online]. Available: https://developer.nvidia.com/embedded/jetpack-sdk-461. (accessed: 18.03.23).

[15] Nvidia. 'Write image to the microsd card.' (), [Online]. Available: https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit#write. (accessed: 12.05.23).

[16] G. Jocher. 'Nvidia jetson nano deployment.' (), [Online]. Available: https://docs.ultralytics.com/yolov5/tutorials/running_on_jetson_nano/. (accessed: 12.05.23).

[17] S. N. Leksikon. 'Forståelse.' (), [Online]. Available: https://snl.no/forst%C3%A5else. (accessed: 05.02.23).

[18] J. Redmon, S. Divvala, R. Girshick and A. Farhadi. 'You only look once: Unified, real-time object detection.' (), [Online]. Available: https://arxiv.org/abs/1506.02640. (accessed: 16.03.23).

[19] Ultralytics. 'Ultralytics yolov8.' (), [Online]. Available: https://docs.ultralytics.com/#ultralytics-yolov8. (accessed: 16.03.23).

[20] Nvidia. 'Nvidia tensorrt.' (), [Online]. Available: https://developer.nvidia.com/tensorrt. (accessed: 16.03.23).

[21] Qualisys. 'Qualisys tracking.' (), [Online]. Available: https://www.qualisys.com/engineering/robotics-and-uav/. (accessed: 16.03.23).

[22] Ros.org. 'Ros2 about.' (), [Online]. Available: https://wiki.ros.org/ROS/Introduction. (accessed: 16.03.23).

[23] Coral.ai. 'Coral about.' (), [Online]. Available: https://coral.ai/docs/accelerator/datasheet/. (accessed: 16.03.23).

[24] tensorflow.org. 'Tfl about.' (), [Online]. Available: https://www.tensorflow.org/lite/guide. (accessed: 16.03.23).

[25] wikipedia.org. 'Single-board computer.' (), [Online]. Available: https://en.wikipedia.org/wiki/Single-board_computer. (accessed: 16.03.23).

[26] OpenCV. 'Opencv documentation.' (), [Online]. Available: https://docs.opencv.org/. (accessed: 16.03.23).

[27] CVzone. 'Cvzone documentation.' (), [Online]. Available: https://github.com/cvzone/cvzone. (accessed: 16.03.23).

[28] R. P. PhD. 'Contour, shape and color detection using opencv-python.' (), [Online]. Available: https://www.researchgate.net/publication/325195384_Contour_Shape_Color_Detection_using_OpenCV-Python. (accessed: 16.03.23).

[29] goodday451999 and modalaashwin41. 'Multiple color detection in real-time using python-opencv.' (), [Online]. Available: https://www.geeksforgeeks.org/multiple-color-detection-in-real-time-using-python-opencv/. (accessed: 16.03.23).

[30] Ros.org. 'Ros2 dds implementation.' (), [Online]. Available: https://docs.ros.org/en/humble/Installation/DDS-Implementations.html. (accessed: 16.03.23).

[31] K. Gruppen. 'Intercom dds datasheet.' (), [Online]. Available: https://www.kongsberg.com/globalassets/kongsberg-defence--aerospace/2.1.-products/defence-and-security/c4isr/intercom-dds/intercom-dds_datasheet.pdf. (accessed: 16.03.23).

[32] eProsima. 'Eprosima fast dds documentation.' (), [Online]. Available: https://fast-dds.docs.eprosima.com/en/latest/. (accessed: 16.03.23).

[33] ros.org. 'Developer overview.' (), [Online]. Available: https://docs.ros2.org/beta2/developer_overview.html. (accessed: 16.03.23).

[34] raspberrypi.com. 'Datasheet for raspberry pi zero 2.' (), [Online]. Available: https://datasheets.raspberrypi.com/rpizero2/raspberry-pi-zero-2-w-product-brief.pdf. (accessed: 20.03.23).

[35] raspberrypi.com. 'Datasheet for raspberry pi 4b.' (), [Online]. Available: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf. (accessed: 20.03.23).

[36] nvidia.com. 'Datasheet for nvidia jetson nano.' (), [Online]. Available: https://developer.nvidia.com/embedded/dlc/jetson-nano-system-module-datasheet. (accessed: 20.03.23).

[37] OscarLiang.com. 'F1, f3, f4, g4, f7 and h7 flight controller processors explained.' (), [Online]. Available: https://oscarliang.com/f1-f3-f4-flight-controller/. (accessed: 20.03.23).

[38] mateksys.com. 'Flight controller h743-slim v3.' (), [Online]. Available: http://www.mateksys.com/?portfolio=h743-slim. (accessed: 20.03.23).

[39] OscarLiang.com. 'Firmware for fpv drone flight controller overview.' (), [Online]. Available: https://oscarliang.com/fc-firmware/. (accessed: 20.03.23).

[40] ArduPilot.org. 'Ardupilot documentation.' (), [Online]. Available: https://ardupilot.org/ardupilot/. (accessed: 20.03.23).

[41] iNavFlight. 'Ardupilot documentation.' (), [Online]. Available: https://github.com/iNavFlight/inav/wiki. (accessed: 20.03.23).

[42] MAVLink.io. 'Mavlink developer guide.' (), [Online]. Available: https://mavlink.io/en/. (accessed: 20.03.23).

[43] MAVLink.io. 'Using mavlink libraries.' (), [Online]. Available: https://mavlink.io/en/getting_started/use_libraries.html. (accessed: 20.03.23).

[44] mavlink.io. 'Manual control (joystick) protocol.' (), [Online]. Available: https://mavlink.io/en/services/manual_control.html. (accessed: 20.03.23).

[45] ArduPilot.org. 'Mavlink interface.' (), [Online]. Available: https://ardupilot.org/dev/docs/mavlink-commands.html. (accessed: 20.03.23).

[46] ArduPilot.org. 'Sitl simulator (software in the loop).' (), [Online]. Available: https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html. (accessed: 20.03.23).

[47] ros.org. 'Mavros.' (), [Online]. Available: http://wiki.ros.org/mavros. (accessed: 20.03.23).

[48] wikipedia.org. 'Pid controller.' (), [Online]. Available: https://en.wikipedia.org/wiki/PID_controller. (accessed: 20.03.23).

[49] ardupilot.org. 'Simulation.' (), [Online]. Available: https://ardupilot.org/dev/docs/simulation-2.html. (accessed: 20.03.23).

[50] raspberrypi.com. 'Camera hardware specification.' (), [Online]. Available: https://www.raspberrypi.com/documentation/accessories/camera.html#hardware-specification. (accessed: 20.03.23).

[51] libcamera.org. 'Documentation - libcamera.' (), [Online]. Available: https://libcamera.org/docs.html. (accessed: 20.03.23).

[52] raspberrypi. 'Github - raspicam.' (), [Online]. Available: https://github.com/raspberrypi/userland/tree/master/host_applications/linux/apps/raspicam. (accessed: 20.03.23).

[53] B. B. Rad, H. J. Bhatti and M. Ahmadi, 'An introduction to docker and analysis of its performance,' *IJCSNS International Journal of Computer Science and Network Security*, vol. 17, no. 3, pp. 228–235, 2017. [Online]. Available: http://paper.ijcsns.org/07_book/201703/20170327.pdf, (accessed 20.03.23).

[54] IBM. (), [Online]. Available: https://www.ibm.com/topics/computer-vision. (accessed: 20.03.23).

[55] R. Kulhary. 'Opencv overview.' (), [Online]. Available: https://www.geeksforgeeks.org/opencv-overview/. (accessed: 20.03.23).

[56] A. Stefanuk. 'Hire opencv developers.' (), [Online]. Available: https://mobilunity.com/blog/hire-opencv-developers/. (accessed: 20.03.23).

[57] 'Blob detection using opencv (python, c++).' (), [Online]. Available: https://learnopencv.com/blob-detection-using-opencv-python-c/. (accessed: 20.03.23).

[58] 'Contour detection using opencv (python/c++).' (), [Online]. Available: https://learnopencv.com/contour-detection-using-opencv-python-c/. (accessed: 20.03.23).

[59] wikipedia. 'Blob detection.' (), [Online]. Available: https://en.wikipedia.org/wiki/Blob_detection. (accessed: 20.03.23).

[60] MathWorks. 'What is object detection.' (), [Online]. Available: https://www.mathworks.com/discovery/object-detection.html. (accessed: 20.03.23).

[61] T. A. Yuya Maruyama Shinpei Kato. 'Exploring the performance of ros2.' (), [Online]. Available: https://ieeexplore.ieee.org/stampPDF/getPDF.jsp?tp=&arnumber=7743223&tag=1. (accessed: 15.05.23).

Appendix A

Operating System Architecture

A.1 Operating System Selection Process

We wanted to ensure that we had the same operating system configuration across all configurations, and to that end we started testing out different distributions to figure out which one would be the best for our purposes.

We wanted to make sure we made an informed decision when it came to our platform, and as such we did a number of tests on other operating systems in order to ensure that we made the correct choice.

Initially we decided to eliminate and narrow down our choices, as the amount of Linux distributions available would make this process in itself take up most our time. So in order to limit the time spent on this task, we decided to select the most common and mainstream Linux distributions, which effectively narrowed it down to Fedora, Ubuntu and OpenSUSE in addition to the vendor-supplied Raspberry Pi OS.

Then we decided to look at what our architecture should look like with the software running on the different hardware configurations and how we would ensure a uniform configuration across all our running systems. A decision was made to run everything within containers, this would ensure that libraries and versions of software running would be identical across all systems.

As a container system, we opted to go for Docker, as this is the most used and the most common configuration out there, and it is well supported on all the platforms we evaluated.

For the software for communicating between the different parts of our system, we opted to use Robot Operating System version 2 (ROS2). This is a ready to run turnkey solution

for a setting up robots or drones, it makes it easy to communicate between different parts of our system, and it is network aware as well as facilitating communication between ROS2 instances running within a Docker environment.

After deciding which configurations to explore, we started by systematically installing the different distributions on physical hardware, and examine what funcationality worked and what did not.

From figure A.1 you can see the different results and what worked what did not.

As indicated in the figure, there were no systems that met all the requirements we had for our software platform, this caused a need to do further research in order to be able to decide which platform we would use going forward.

We further narrowed down our choices to Ubuntu or Debian, as these platforms only had one missing dependency, however, we needed to determine which path would yield the best results in contrast to the work needed to fix each system.

Debian had the disadvantage of having problems getting ROS2 to work, which has a fairly complicated and extensive build-system, where the documentation can be outdated due to how fast the ROS2 project moves at this point in time. On the other hand, the challenge posed by the Ubuntu system was the absence of driver software for the camera module version 3.

As we had limited experience with the ROS2 build system, and documentation on how to manually build and install the software was sparce, we decided to attempt to install the existing driver from Debian in a Ubuntu environment, in order to run ROS2 with camera version 3.

We started by cloning the git repository from the Raspberry Pi foundation, where we knew there was a working driver, as our initial tests with Debian showed this to be the case. After cloning the repository we cross-compiled the kernel configuration along with the necessary libraries for installation on our Ubuntu system.

After finishing the installation and booting our new kernel, the hardware was detected, but there was no video output. At this point, it was our opinion that we could still get the hardware to function under Ubuntu, as that would be our best course of action at this moment in time.

We continued to perform different tests with different configurations, as well as compiling different versions of the Linux kernel, including a vanilla upstream version. When none of these approaches yielded the results we needed, we were forced to re-evaluate our attack vector for this problem.

We revisited the drawing board and had a look at running ROS2 on our Debian system, which would provide us access to the camera module 3 driver, and would simplify deployment of our software.

After deciding to attempt a ROS2 compile for Debian, we dove into the documentation for compiling the software on our system, however, this proved less than straightforward and we had numerous challenges attempting to get the software to compile. As an intermediary solution we found someone who had compiled a Debian package for us on github, but this was from an older build and we could not verify it for our uses, leaving us with the necessary steps to continue to deploy ROS2 while we had an insecure installation with ROS2 for other group members so we could work in parallell and speed up development.

After a long period of non-stop work we were able to compile ROS2 under a Debian system, allowing us access to the camera module 3 and ROS2 on the same computer. After this step was done, we needed to deploy this to a Docker image, allowing us to control the environment and secure a cohesive and uniform platform for our ROS2 source code.

However, an unfortunate side-effect of compiling software is that it grows rather large, and after building our Docker system, it had increased in size to 14GB, which was unwieldly for deployment and it posed a challenge to run on the Raspberry Pi Zero. This was sadly nothing we were able to rectify in time before delivering our project, and while it does work in the Raspberry Pi Zero, deployment is slow and prone to create problems if the Zero runs out of memory while pulling the Docker image.

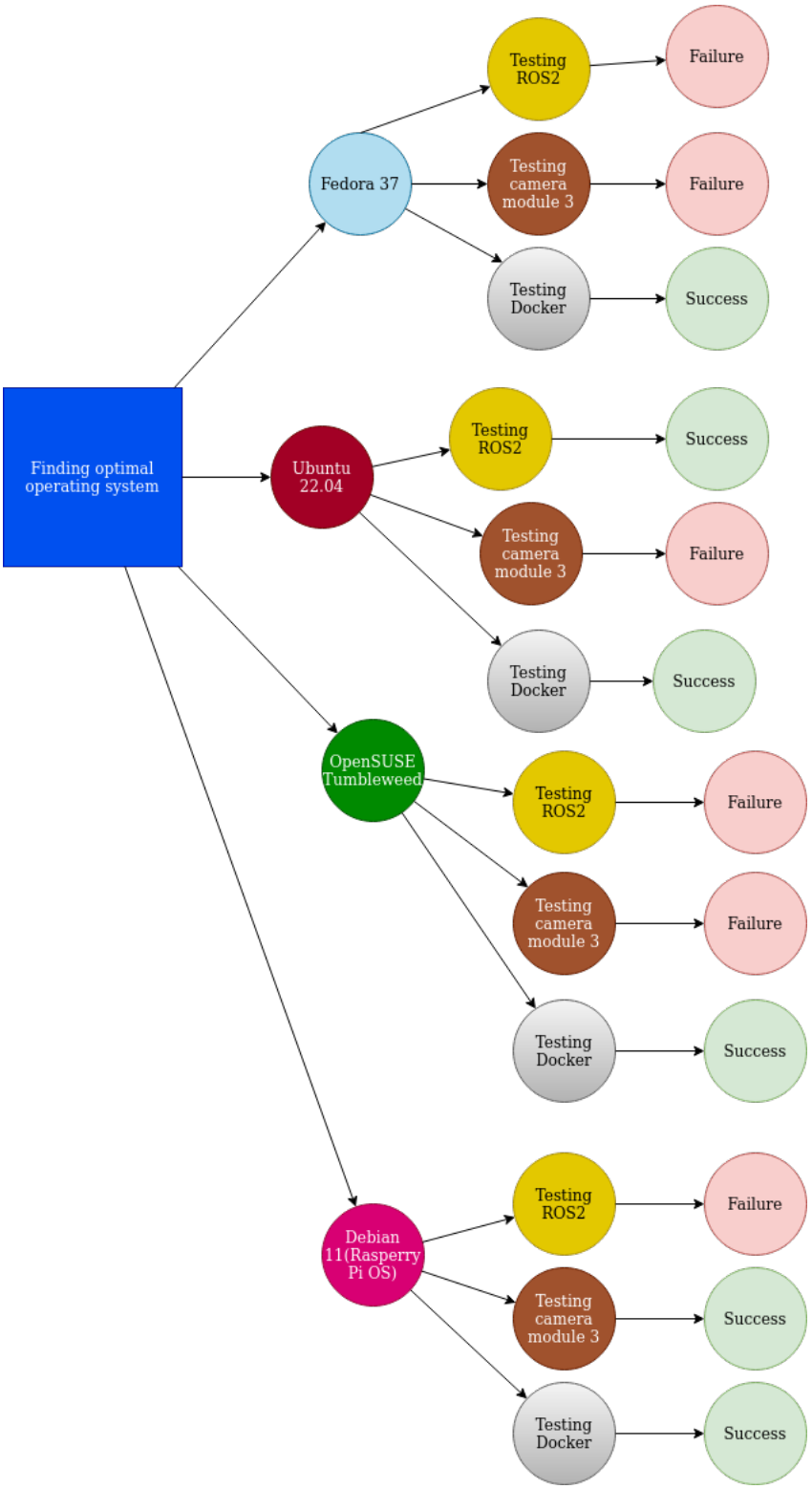


Figure A.1: Decision Process OS

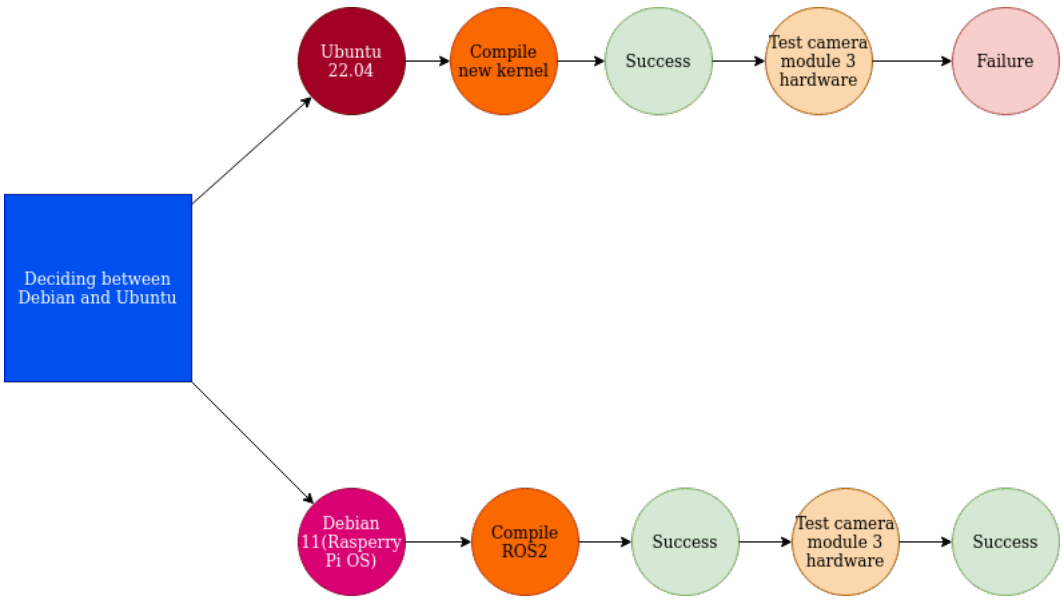


Figure A.2: Decision Process Distribution

Appendix B

Algorithms

B.1 Distance and Color detection

Since we use the Raspberry Pi Camera Module v3 and not a depth camera, we need to find a way to find the distance to a known object. We ended up using OpenCV and cvzone. Our approach involves detecting a specific color in the video frames, identifying the contours of the detected object, and then calculating the distance based on the object's apparent size in the image.

First, we import the necessary packages for image processing, numerical operations, and color detection:

```
1 import cv2
2 import cvzone
3 from cvzone.ColorModule import ColorFinder
4 import numpy as np
```

We create a VideoCapture object (OpenCV) to read video frames from the camera [26], and set the capture dimensions to 640x480 pixels [26]:

```
1 cap = cv2.VideoCapture(0)
2 cap.set(3, 640)
3 cap.set(4, 480)
```

Next, we create a ColorFinder object (cvzone) with automatic color range update turned off [27], and define the color range for detection (using predetermined HSV values):

```
1 myColorFinder = ColorFinder(False)
2 hsvVals = {'hmin': 97, 'smin': 21, 'vmin': 23, 'hmax': 125, '
    smax': 255, 'vmax': 193}
```

In a continuous loop, we read the video frames from the camera (OpenCV) [26], detect the specified color range in the image using the ColorFinder object (cvzone) [27], and find the contours in the binary mask (cvzone) [27]:

```
while True:
    success, img = cap.read()
    imgColor, mask = myColorFinder.update(img, hsvVals)
    imgContour, contours = cvzone.findContours(img, mask)
```

When contours are detected, we extract data from the first contour (assumed to be the object of interest) and calculate the distance to the object based on its apparent size in the image. We use the real-world dimensions of the object (in this case, a tennis ball with a width of 6.5 cm) and the camera's focal length to compute the distance:

```
if contours:
    data = contours[0]['center'][0], h - contours[0]['center'
        ][1], int(contours[0]['area'])

    f = 535  # focal length of the camera
    W = 6.5  # real-world width of the tennis ball

    w = np.sqrt(contours[0]['area']/np.pi) * 2  # width of the
        tennis ball in the image
    d = (W * f) / w  # calculate the distance

    print(d)
```

Finally, we display the distance in centimeters on the image (cvzone) [27] and stack the original image, detected color image, binary mask, and contour image for visualization (cvzone) [27] this is done for testing purpose:

```
cvzone.putTextRect(img, f'depth: {int(d)} cm', (contours[0]['
    center'][0] - 75, contours[0]['center'][1] - 50), scale= 2)
imgStack = cvzone.stackImages([img, imgColor, mask, imgContour
    ], 2, 0.5)
cv2.imshow("Image", imgStack)
cv2.waitKey(1)
```

This approach is suitable for objects with known dimensions and relatively uniform color distribution. This testing was used with a simple color detection. We can use this method to use the contours of trained models to find the distance aswell.

Appendix C

Team

Gruppe 6-2022 – Aerial Edge

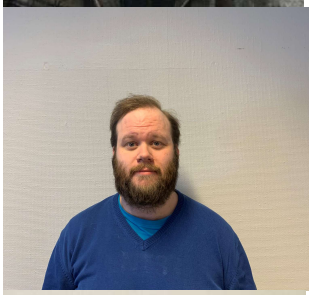
Our thesis is to research the possibilities edge computing gives in an embedded system.

Even Jørgensen
Even1993@hotmail.com
Computer engineer

Main responsibility
Group leader

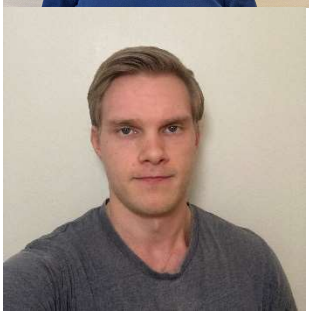
Martin Børte Liestøl
Martin.liest@gmail.com
Computer engineer

Main responsibility
Programming



Sindre Nes
Sin_pin@hotmail.com
Computer engineer

Main responsibility
Report



Ådne kvåle
237113@usn.no
Tlf: 47242193
Computer engineer

Main responsibility
Hardware

Abdul Majeed Alizai
Majeed.alizai4@gmail.com
Computer engineer

Main responsibility
Documentation

Jon Jahren
jon.jahren@gmail.com
Computer engineer

Main responsibility
Project model/Agile

Internal supervisor: **Henning Gundersen**
Epost: **Henning.gundersen@usn.no**

External sensor: **Tord Fauskanger**
Epost: **tord.fauskanger@gmail.com**

External supervisor: **Jan Dyre Bjerknes**
Epost: **Jan.dyre.bjerknes@kongsberg.com**