

Antti Hahka

PHYSICAL SAFETY IN WIRELESS EDGE OFFLOADING OF CONTROL DECISIONS

A case study using a nano-drone

Master's Thesis in Technology
Faculty of Engineering and Natural Sciences
Examiners: University Instructor Mikko Salmenperä
Associate Professor Pekka Jääskeläinen
November 2024

ABSTRACT

Antti Hahka: Physical Safety in Wireless Edge Offloading of Control Decisions
Master's Thesis in Technology
Tampere University
Master's Degree Programme in Automation Engineering
November 2024

Multi-access edge computing offers small edge devices the possibility to run more demanding workloads on a remote edge server instead of running them locally by using edge offloading. Edge offloading brings new use cases for these small edge devices, for instance, the capability to offload heavy AI object detection task as part of the application logic. However, edge offloading introduces increased latency and the possibility of completely losing the connection to an edge server, which poses a risk in physical safety if the offloaded computation is part of the device's control decisions.

The questions and issues of physical safety due to edge offloading are the main interests in this thesis. To understand the physical safety more thoroughly, a test scenario is created where a small scale nano-drone called Crazyflie edge offloads an AI object detection task to a remote edge server. The results of the AI object detection are then utilized in the Crazyflie to fly and follow a target object.

The questions of physical safety are considered by making use of the European Union's drone regulations 2019/947 and 2019/945 and automotive standards ISO 26262 and 19237. The EU drone regulations directly affect what should be taken into account when operating a drone, whereas the automotive standards are applied in improving and measuring safety in the test scenario.

Out of the standards and regulations, ISO 26262 Part 3 is utilized the most. This standard is used to define the concept of the drone system which leads to creation of an ISO 26262 item called "pedestrian detection system" (PDS). This item is the main point of interest in terms of safety as this item includes the edge offloading functionality in the test scenario. By following the standard, the safety requirements of this item are recognized. After this, ISO 19237 is applied to determine the physical safety test criteria in the PDS.

The actions of improving safety are validating the output, implementing a local fallback and measuring the latency requirements for the Crazyflie. Output validation is implemented in the application code level as a simple boundary check. The local fallback implementation is done to combat the situation where the connection to an edge server is lost. Finally, the latency between Crazyflie and an edge server is measured to determine safe flying speeds on different latency scenarios when edge offloading is used.

Keywords: edge offloading, physical safety, drone, control decision, ISO 26262

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

TIIVISTELMÄ

Antti Hahka: Turvallisuuden parantaminen langattomien ohjauspäätösten siirrossa
Diplomityö
Tampereen yliopisto
Automaatiotekniikan DI-ohjelma
Marraskuu 2024

Reunalaskenta tarjoaa pienille reunalaitteille mahdollisuuden hyödyntää reunapalvelinten suurta laskentatehoa raskaan laskennan suorittamisessa sen sijasta, että nämä pienet laitteet ajaisivat raskaat laskut itse. Tämä laskennan siirto mahdollistaa esimerkiksi tekoälymallien ajamista osana reunalaitteen sovelluslogiikkaa. Ongelmana laskennan siirrossa on kuitenkin se, että se vaatii asiakas-palvelin-yhteyden, joka aiheuttaa latenssia ja mahdollistaa yhteyden katkeamisen. Mikäli laskennan siirtoa on hyödynnetty osana reunalaitteen ohjauspäätöksiä, herättää tämä kysymyksiä reunalaitteen turvallisuudesta esimerkiksi latenssin puitteissa.

Tämän opinnäytetyön tarkoituksena on tutkia reunalaitteen fyysistä turvallisuutta osana laskennan siirtoa. Tätä tarkoitusta varten tässä työssä hyödynnetään testijärjestelyä, jossa pieni-kokoinen Crazyflie-drooni siirtää tekoälyn avulla suoritettavan olioiden tunnistustehtävän reunapalvelimelle. Crazyflie-drooni yrittää tämän tekoälylaskennan tulosten perusteella seurata tiettyä kohdeoliota.

Turvallisuuskysymyksiä tutkitaan hyödyntämällä sekä Euroopan Unionin drooniasetuksia 2019/947 ja 2019/945 että autoteollisuuden standardeja ISO 26262 ja ISO 19237. EU:n droonisäännökset kertovat, mitä pitää huomioida drooneja käytettäessä, kun taas autostandardeja sovelletaan turvallisuuden parantamiseksi testiskenaariossa.

Näistä säännöksistä ja standardeista ISO 26262:n 3:ttä osiota käytetään eniten. Tämän osion avulla droonijärjestelmä pilkotaan eri osiin, joista yksi otetaan tarkkaan käsittelyyn tässä työssä. Tämän tutkittavan osan nimi on "ihmisen tunnistamisjärjestelmä", joka sisältää edellä mainitun tekoälyn laskennan siirron reunapalvelimelle. Seuraamalla tätä standardia luodaan käsitys ihmisen tunnistamisjärjestelmän turvallisuusvaatimuksista. Standardin ISO 26262 lisäksi sovelletaan standardia ISO 19237 selvittämään ihmisen tunnistamisjärjestelmän suorituskyvyn vähimmäisvaatimukset.

Toimenpiteet turvallisuuden parantamiseksi ovat ihmisen tunnistamisjärjestelmän ulostulon validointi, varajärjestelmän toteuttaminen sekä latenssivaatimusten muodostaminen Crazyflie-droonilla. Ihmisen tunnistamisjärjestelmän ulostulo validoidaan rajatarkastuksella. Varajärjestelmä toteutetaan asiakas-palvelin-yhteyden katkeamista varten. Lopuksi Crazyflie-droonin ja reunapalvelimen välinen latenssi mitataan, mitä käytetään määrittämään droonin turvalliset lentonopeudet kun laskennan siirtoa käytetään.

Avainsanat: laskennan siirto, fyysinen turvallisuus, drooni, ohjauspäätös, ISO 26262

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck -ohjelmalla.

USE OF ARTIFICIAL INTELLIGENCE IN THIS WORK

Artificial intelligence (AI) has been used in generating this work:

- No
- Yes

Acknowledgement of risks

I hereby acknowledge, that as the author of this work, I am fully responsible for the contents presented in this thesis. This includes the parts that were generated by an AI, in part or in their entirety. I therefore also acknowledge my responsibility in the case, where use of AI has resulted in ethical guidelines being breached.

PREFACE

This thesis was written as a part of Customized Parallel Computing group's research project in the Information Technology and Communications faculty of Tampere University. The research project included demonstrating *PoCL-R* edge computing software stack capabilities in a computationally limited edge device. This thesis was done under the supervision of associate professor Pekka Jääskeläinen from Tampere University. University instructor Mikko Salmenperä worked as the other examiner.

First, I would like to thank Pekka for his guidance throughout the whole writing process, but also for giving me the opportunity to make my master's thesis out of this topic. I would also like to thank Mikko for both his guidance in this thesis but also the work he has done in automation technology courses over the years.

Great thanks are also appointed to the CPC members that I have worked with during the past year. Special mentions go to Jan Solanti, Robin Bijl, Yashvardhan Agarwal and Jakub Žádník.

Finally, I am grateful to my friends in Tampere and Rovaniemi who have made my study years as memorable as they are.

Tampere, 6th November 2024

Antti Hahka

CONTENTS

1.	Introduction	1
2.	Physical safety	4
2.1	Automotive standards	4
2.1.1	ISO 26262.	4
2.1.2	ISO 19237.	7
2.2	Safety regulations on unmanned aircraft system operations	8
2.3	Known drone hazards by Traficom	9
3.	Multi-access edge computing	11
4.	Test environment.	14
4.1	Example scenario	14
4.2	PoCL-R	15
4.3	Crazyflie 2.1 nano-drone.	16
4.3.1	Baseboard	16
4.3.2	AI-deck	18
4.3.3	Flow deck	19
4.3.4	Capabilities of actuators	20
4.4	Software development tools	26
4.5	Remote server.	26
4.6	Communication overview	27
4.7	Test scenario	28
4.8	Standards and test environment	29
4.8.1	Categorizing Crazyflie	30
4.8.2	Applying ISO 26262	32
4.8.3	Applying ISO 19237	33
5.	Adapting ISO 26262-3	35
5.1	Defining an item	35
5.1.1	Functional behavior	36
5.1.2	Quality, performance and availability of functionality	37
5.1.3	Constraints, dependencies and operating environment	39
5.1.4	Consequences of known hazards	39
5.1.5	Impact of actuators	40
5.2	Item interactions	41
5.2.1	Item requirements from other items	41
5.2.2	Assumptions on effects of item behavior	42

5.2.3	Requirements from the item	43
5.2.4	Item functionality in different operational scenarios	44
5.3	Hazard analysis and risk assessment.	46
5.3.1	Situation analysis and hazard identification	46
5.3.2	Classification of hazardous events	48
5.3.3	ASIL determination	50
5.3.4	Safety goals	51
6.	Actions to satisfy safety goals.	52
6.1	Output validation	52
6.2	Local fallback	53
6.3	Remote execution time budget	55
6.3.1	Latency measurements	55
6.3.2	Maximum initial speeds	58
6.3.3	Reflecting on results	60
7.	Conclusion	61
	References	63
	Appendix A: Python code	67

LIST OF SYMBOLS AND ABBREVIATIONS

AEG	Automatic Exposure and Gain. A feature found in AI-deck camera that independently tries to optimize the image quality in the given environment.
CPX	Crazyflie Packet eXchange. A communication protocol developed by Bitcraze enabling communication between different MCUs on Crazyflie ecosystem.
CRTP	Crazyflie RealTime Protocol. A real-time control focused packet protocol developed by Bitcraze for Crazyflie ecosystem.
E/E	Electrical and/or Electronic.
ETSI	European Telecommunications Standard Institute. European standards body governing electric telecommunications, networks and services.
FCS	Flight Control System. An ISO-26262 item recognized in the drone test environment. Receives AI-inference object results as an input and provides the drone control system a general input of where the drone should fly.
I2C	Inter-Integrated Circuit. A synchronous two-way master/slave serial communication bus protocol in intra-board communication.
IoT	Internet of Things. Concept of connecting multiple devices together to allow data sharing between network devices but also controlling these devices via internet.
ISA	Instruction Set Architecture. A set of instructions that define what instructions a processor implementation must be able to execute.
ISG	Industry Specification Groups. These groups operate under ETSI and each group works in different technological area alongside traditional standards committees to provide technical specifications. For instance, MEC ISG focuses on standardisation of MEC.
ISO	International Organization for Standardization. A worldwide federation of national standards bodies. The international standards are prepared in ISO technical committees where each member body has a right to be represented.

LiDAR	Light Detection And Ranging. A sensor type that sends light pulses to its surroundings and measures the travel time when the reflection hits back on the sensor. The travel time can be used to accurately measure distances to objects in the surroundings.
MCU	Microcontroller Unit. A miniature "computer" on a chip containing a processor, memory and input-output system.
MEC	Multi-Access Edge Computing. Network architecture model created by ETSI MEC ISG with the focus on bringing cloud computing capabilities closer to radio base stations to serve end users in proximity.
MMU	Memory Management Unit. A piece of hardware on a computer chip that translates virtual memory addresses into physical memory addresses.
MTOM	Maximum Take-Off Mass. Maximum operation mass of unmanned aircraft including its fuel and payload.
OpenCL	Open Computing Language. Open cross-platform standard for heterogeneous parallel computing.
PDCMS	Pedestrian Detection and Collision Mitigation System. A system in vehicles that reduces the severity of unavoidable pedestrian collisions.
PDS	Pedestrian Detection System. An ISO-26262 item under the scope of this thesis. PDS includes running an AI-inference of given input image to provide its results to FCS for controlling the drone.
PID	Proportional-Integral-Derivative. A type of process controller utilizing aforementioned three parameters to calculate the amount of control needed to guide system output towards a setpoint value.
PoCL	Portable Computing Language. Open-source implementation of OpenCL with the aim of easy portability and interoperability between multiple devices.
PoCL-R	PoCL-Remote. A remote driver in PoCL that allows execution of OpenCL commands on remote servers.
RAN	Radio Access Network. Mobile telecommunication network utilizing some physical radio communication technologies, e.g. Wi-Fi and 5G.
SPI	Serial Peripheral Interface. A synchronous intra-board master/slave serial communication standard.

SRAM	Static Random-Access Memory. Volatile memory that keeps its state without the need of refreshing until it is powered off.
ToF	Time-of-Flight. A sensor type that utilizes round trip time of light pulse to measure distances.
UART	Universal Asynchronous Receiver Transmitter. An integrated circuit able to transfer parallel data (bytes) to serial data (bits) and send them over a serial port and vice versa.
UAS	Unmanned Aircraft System. The actual aircraft (UAV) but also the equipment to control it remotely.
UAV	Unmanned Aerial Vehicle. The actual aircraft flying.
UML	Unified Modeling Language. A standardized visual modeling language for visualization of system design.
VLOS	Visual Line Of Sight operation. UAS operation in which the remote pilot maintains visual contact with the unmanned aircraft.

1. INTRODUCTION

Modern society is all about connections whether it be connecting people or devices. IoT (Internet of Things) has taken over and more and more devices are connected together in ever-rising pace to form complex information systems such as wearable smart devices, autonomous vehicles and smart home applications like heating and security. As more devices are connected, not only the amount of connected devices has increased but also the amount processable data has grown substantially. To tackle this task, one of the emerged connection solutions is called *edge computing*, in which multiple different types of *edge devices* can communicate to physically near *edge servers* quickly.

Edge servers are capable of orchestrating a network of devices but they also come with a useful functionality of allowing edge devices to *offload* heavy computational workload to them. As edge devices get smaller and more lightweight, their computational capabilities will be heavily constrained compared to what is required from them. For this, edge computing offers a solution: edge devices may offload heavy workloads to be computed away from small devices but still utilize the computed results in these small devices.

An example use-case using an AI object detection is presented in this thesis. A small edge device wants to utilize an AI model for an object detection task to be able to follow an object. However, due to the computational requirements present in running the AI object detection, the edge device cannot run the AI locally on the device in a timely manner to feasibly follow the object. In order to fulfill this follow task, the small edge device offloads the AI calculations on a remote edge server. However, this edge offloading introduces new problems such as latency and the server connection being lost during the follow task.

The problems of latency and server connection are relevant especially when there exists an actual moving edge device like an autonomous car or a small flying drone that wants to use an AI object detection as a part of its movement choices. A new challenge appears in terms of physical safety of an edge device. Is the latency low enough for this kind of follow task to be feasible? What happens on the edge device when the server connection is lost? These are the most essential questions regarding physical edge device safety in this thesis.

To consider the physical safety of an edge device in an edge offloading scenario, a case-

study is performed in this thesis. A small *Crazyflie* nano-drone manufactured by Bitcraze is used as an edge device and it offloads an AI object detection part of its application logic to be run on a remote server instead of running it locally. This edge offloading is done with an example edge computing software stack called *PoCL-R* (Portable Computing Language Remote). Alongside the test scenario, safety is also looked more in detail by utilizing two automotive safety standards, the European Union's regulations on flying drones and the national rules set by Traficom (the Finnish Transport and Communications Agency).

The first automotive standard is ISO 26262 which is an automotive safety standard that governs functional safety for automotive systems. The standard gives guidelines on how to consider all parts of electric and electronic parts of an automotive system with safety in mind. In this thesis, utilizing the standard leads into defining an ISO 26262 *item* called "pedestrian detection system" (PDS). The PDS is a system that consists of the edge offloading part of AI object detection in the case-study.

The second automotive standard is ISO 19237, an intelligent transport systems standard, which helps vehicle manufacturers to test their "pedestrian detection and collision mitigation systems" (PDCMS) performance-wise. PDCMS is a system that tracks e.g. pedestrians and their relative movement to the vehicle, which fits the definition of the PDS in the case-study. The performance metrics in the standard are used to give insight into the measurable safety parameters of the PDS.

In addition to vehicles, also the physical safety of flying drones is considered. This is done by inspecting the EU drone regulations 2019/947 and 2019/945, and national drone rules set by Traficom. The EU regulations cover different safety measures to be taken when flying drones of different categories. The regulations help properly categorize the drones used in the EU area and thus use the relevant safety measures. Additionally, Traficom has set rules regarding drone flights in the Finnish aerospace.

Research questions in this thesis relate to the safety of edge offloading of control decisions from edge devices and are confined as follows:

1. Can time-critical movement related control decisions be edge offloaded safely from an edge device?
2. What kinds of safety regulations and standards exist for controlling real-time devices?

Research question one answers to feasibility of edge offloading from real-time devices as this introduces higher latency. This provides a challenge in real-time movement control especially in the case of time-critical actions. Time-critical actions could be for example an autonomous car braking before hitting an object. Too big of a latency in receiving the braking command will make the offloading scenario unfeasible. This question is an-

swered by the case-study. Question two explores standards and regulations of real-life real-time devices. The question is answered by taking a look into the EU and Finnish safety regulations of flying drones and existing ISO standards of autonomous vehicles.

The structure of this thesis is as follows. Chapter 2 introduces the automotive standards ISO 26262 and 19237 and the EU drone regulations 2019/947 and 2019/945. Chapter 3 introduces multi-access edge computing basic principles and features. Chapter 4 describes the test environment: used equipment and tools, the example scenario where the test scenario is made of and how the standards are used especially with regards to the test equipment. Chapter 5 focuses on adapting ISO 26262 Part 3 to define a "pedestrian detection system" which includes the edge offloading functionality. This leads to defining safety goals that should be met on the test scenario. Chapter 6 describes the actions taken to satisfy the safety goals. Finally, Chapter 7 wraps up this thesis by summarizing what was done.

2. PHYSICAL SAFETY

This chapter introduces the key standards and rules related to *physical safety* in this thesis. Physical safety is the most important concern in any moving autonomous systems. It is seen as the protection of humans from injuries when any moving objects are present in the vicinity of humans. For instance, an autonomous car can cause serious injuries to humans if it cannot break in time to avoid pedestrians crossing a street. As latency is increased in an edge offloading scenario, physical safety becomes even more serious of a concern.

This chapter is divided into three sections. Section 2.1 introduces the automotive safety standards ISO 26262 and ISO 19237. Section 2.2 expands the safety aspect by including the European Union's regulations impacting unmanned aircraft systems (UAS) such as the implementing regulation 2019/947 [1] and delegated regulation 2019/945 [2]. Section 2.3 introduces some known potential risks that the Finnish Transport and Communications Agency Traficom has recognized in drone uses.

2.1 Automotive standards

This Section introduces automotive standards ISO 26262 in Section 2.1.1 and ISO 19237 in Section 2.1.2. It is worth noting that these standards are designed for automotive industry. The following two standards were chosen for adaptation due to recent introduction of computer vision based self-driving capabilities in modern road vehicles, which is similar situation as in the test case of moving a drone based on AI object detection.

2.1.1 ISO 26262

ISO 26262 is a series of standards adopting IEC 61508 set of standards in the sector of electrical and/or electronic (E/E) systems within road vehicles. IEC 61508 is a generic basis for *functional safety* in electrical/electronic/programmable electronic systems, whereas ISO 26262 is solely dedicated to functional safety within automotive industry. [3, Introd., Ch. 4.1] ISO 26262 consists of 12 different parts which include, for example, vocabulary and definitions, management level operations, different development phases like hardware and software development and supporting processes like validation and verification.

Adopting ISO 26262 supports all stages in the vehicle's lifecycle to improve functional safety in the E/E systems found in the vehicle. [3, Introd.]

According to ISO 26262, functional safety means the "absence of unreasonable risk due to hazards caused by malfunctioning behavior of E/E systems" [4]. In other words, functional safety aims to reduce the impact on physical health of humans when some part of an electrical system fails. ISO 26262 aims for functional safety with the following methods [3, Introd.]:

- provide a reference framework for a safety lifecycle in all different lifecycle phases,
- provide a risk-based approach to determine Automotive Safety Integrity Levels (ASILs),
- use ASILs to specify applicable ISO 26262 requirements to avoid risks,
- provide requirements for functional safety in all parts of the development process,
- provide requirements for customer-supplier relations.

ASIL (Automotive Safety Integrity Level) specifies an *item's* necessary ISO-26262 requirements and safety measures to apply in order to avoid an *unreasonable risk*. An unreasonable risk is a risk deemed unacceptable under certain context. [4]

Starting off with the item definition, an item represents a system or combination of systems that implements a function within a vehicle. [5, Cl. 5] An *element* is a *system, component* (hardware or software), *hardware part* or *software unit*: [4]

1. System: set of components that relates sensors, actuators and controllers with each other.
2. Component: non-system level element, which consists of one or more hardware parts or software units (e.g. a microcontroller).
3. Hardware part: a portion of *hardware component* (e.g. a CPU in a microcontroller).
4. Software unit: a portion of *software component* which can be stand-alone tested.

Figures 2.1 and 2.2 show how ISO 26262 sees these terms and their relationships with other terms [3, Ch. 4.2]. Figure 2.1 shows the general composition of different terms whereas Figure 2.2 uses UML (Unified Modeling Language) to represent relationships between some of the terms. In UML, the arrows bear different meanings depending on how they look. For example, a function is implemented by one or more items (represented by a dotted line and a triangle as the end point), or a system has one or more components (represented by a solid line with a diamond as the end point).

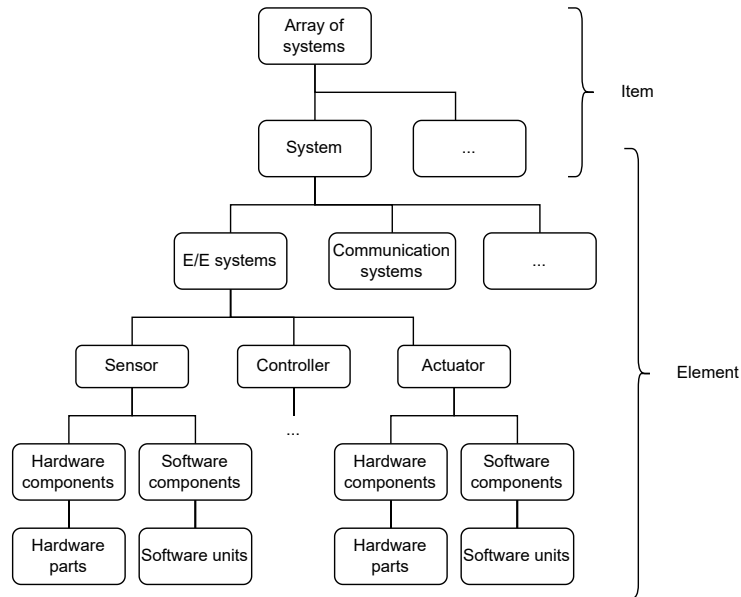


Figure 2.1. Definitions of "item" and "element" in ISO 26262 [3, Figure 4].



Figure 2.2. Relationships between different terms in ISO 26262 [3, Figure 3].

Coming back to ASIL, ASIL consists of 4 levels, A, B, C and D, which represent the required safety measures to avoid risks with A being the lowest safety integrity level and D the highest. ASIL A requires the least strict safety measures whereas D requires the most strict ones. [5, Clause 6.4.3] ASIL also has a fifth class called QM (Quality Management) which denotes no requirement to comply with ISO-26262. Classification QM indicates that quality processes in place are enough to manage the identified risk. Table 2.1 shows all different combinations of severity, exposure and controllability that are used to determine ASILs.

Table 2.1. ASIL determination using S, E and C classifications.

Severity class	Exposure class	Controllability class		
		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

In terms of safety, ISO 26262 uses the following safety notions [3, Ch. 4.1]:

- A *hazard* or *hazardous event* is an event that needs to be prevented, mitigated or controlled. These come as a result of hazard analysis and risk assessment (HARA).
- Each hazard is associated with at least one *safety goal*. Safety goals are given an ASIL level.
- *Functional safety* concept describes the functionality on how to achieve a set safety goal.
- *Technical safety* tells how the functionality is actually implemented on hardware and software level.
- *Software* and *hardware safety* state different safety requirements that are to be implemented in software and hardware design, respectively.

2.1.2 ISO 19237

ISO 19237 is an intelligent transport systems standard describing pedestrian detection and collision mitigation systems (PDCMS) performance requirements and test procedures for light duty passenger vehicles (at most 8 passenger seats plus the driver seat). The standard specifies the operation concept, minimum functionality, system requirements, system interfaces and test procedures for PDCMS. It also defines the test criteria to verify that a PDCMS implementation meets the requirements of the standard as the implementation choices are left to the vehicle designers. The standard can be applied to stand-

alone PDCMS or on the integration of PDCMS functionalities into other driver vehicle assistance and support systems. [6]

The goal of PDCMS is to reduce the severity of unavoidable pedestrian collisions by reducing the likelihood of fatality rate and severity of injuries. Collision avoidance is not in the scope of this standard and neither are other countermeasures than emergency braking such as evasive steering. [6, Ch. 1]

The standard specifies requirements of minimum enabling capabilities of a PDCMS [6, Ch. 5.1]:

- detect the presence of pedestrians,
- determine the direction of the pedestrian relative to vehicle,
- determine the distance and relative velocity between the vehicle and the pedestrian,
- determine the vehicle velocity,
- initiate PDCMS countermeasures,
- provide the driver information about potential collision with collision warning system,
- activate vehicle brakes,
- control brake lights,
- enhance vehicle yaw stability through ESC (electronic stability control) during braking,
- reduce speed according to defined test pass criteria (avoid a collision or hitting a pedestrian with the speed of less than 10 km/h),
- allow the driver to increase the deceleration when emergency braking up to maximum possible vehicle deceleration,
- allow the driver to override commands at any time, and
- provide system availability information to the driver.

In other words, the PDCMS require information about range to pedestrians, motion of pedestrians, motion of the subject vehicle (vehicle equipped with PDCMS) and driver commands and actions. With these information the PDCMS determines whether or not the collision is imminent and starts emergency braking to mitigate collision severity. [6]

2.2 Safety regulations on unmanned aircraft system operations

The European Union has posed safety regulations on unmanned aircraft system (UAS) operations. The applicable regulations are Commission Implementing Regulation (EU) 2019/947 and Commission Delegated Regulation (EU) 2019/945. Regulation 2019/947

considers the rules for operating a UAS and the personnel included in the operation while Regulation 2019/945 focuses on the specifications of the UAS itself.

Regulation 2019/947 divides UAS operations into three categories based on the unmanned aircraft in use: *open*, *special* and *certified* [1, Art. 3]. These categories define different requirements which are to be enforced when flying.

UAS operations in the *open* category do not need any prior operational permissions before the operation. Neither does the UAS operator need to declare the operation in advance. [1, Art. 3]

Operations in the *special* category require an operational authorization by a competent authority. In Finland, this authority is Traficom [7]. A declaration made by a UAS operator is also sufficient to perform the operation in specified cases in this category. An example of this kind of case is flying an unmanned aircraft with a dimension up to three meters in a controlled airspace below 120 meters. [1, Art. 3 & 5, UAS.SPEC.020]

The *certified* category operations require a certification of the UAS according to Regulation 2019/945. The drone operator must be certified and in some cases the remote pilot must have a license. [1, Art. 3] For instance, certified operations are drone flights over assemblies of people and if the drone carries either people or dangerous goods that pose a high risk for humans in case of an accident [1, Art. 6].

Regulation 2019/945 establishes the requirements for the design, manufacture and required certifications of a UAS operating under the three different categories listed in Regulation 2019/947. This Regulation also specifies rules on drones intended to be used in the "open" category within the EU market. [2, Art. 1]

2.3 Known drone hazards by Traficom

Traficom is the Finnish Transport and Communications Agency. They are the local aviation authority in Finland [7] and oversee the drone operations and compliance of drone regulations in Finnish airspace.

Traficom has established guides and potential risks when operating drones in their website. As known hazards are part of considering physical safety in ISO 26262, a collective list of known hazards and potential risks is presented here. These hazards and risks were collected from Traficom's self-study material [8].

- Decline of performance in humans – intoxicants, senses, colorblindness,
- Weather conditions affecting flying – temperature (air pressure) , wind, rain, fog,
- Weather conditions affecting drone – battery level, potential ice formation on wings or motors, rain affecting drone,

- Weather conditions affecting human operators – flying in dark vs. light,
- Other airspace users – other aircraft, birds,
- Taking off/landing – lack of VLOS, surface material and how flat it is,
- Drone payload – weight, displacement of payload including battery (affects center of gravity), air friction increase (different response to flight inputs),
- Motor failure – more severe when flying a drone with 4 motors than a drone with 6 motors,
- Radio signal interference – congested operating channel frequency, loss of communication,
- Battery charge level,
- Sudden turbulence.

3. MULTI-ACCESS EDGE COMPUTING

Edge computing is a distributed network architecture model in which a *cloud* provider offers its cloud computing capabilities, storage and other IT services closer to end users at the network *edge*. A cloud can be understood as a vast network of servers that offer e.g. storage or computing power and is accessed through the internet. As a conceptual model edge can be imagined as the boundary between end users and a cloud, which lies in close proximity to where data is produced.

Multi-access edge computing (MEC) is a network architecture concept that is being developed by the European Telecommunications Standard Institute (ETSI) Industry Specification Group (ISG) starting from 2014. Back then the concept was known as "mobile edge computing" as the initial work revolved around using radio access network (RAN) to provide cloud computing capabilities to nearby mobile users via base stations. In 2016 MEC ISG expanded their concept to include heterogeneous networks (e.g. network including 5G RAN & Wi-Fi LAN) and thus rebranded the term as "multi-access edge computing". [9]

Figure 3.1 shows an example of MEC network structure. MEC servers are located right on the edge of a network instead of further away in some centralized server or data center. This placement allows the servers to be located in close proximity to end devices on the edge, which allows low latency communication between them. Multiple types of different end devices may communicate with the MEC servers in different networks using e.g. Wi-Fi or 5G, which increases the mobility aspect in this kind of edge architecture. The following list introduces different features found in MEC architecture [9, Chapter 2]:

- Low latency. Offloading to MEC server results in smaller latency than offloading to cloud. Latency to MEC server can be about 1 ms while latency to cloud can vary between 30–100 ms [9, Table 1].
- High bandwidth. Instead of all end devices pushing their data to cloud and creating congestion, end devices push their data to MEC servers in smaller networks.
- Real-time radio network statistics. MEC servers can receive network information from end devices.
- Location awareness. MEC server can estimate end device locations within a network based on received information.

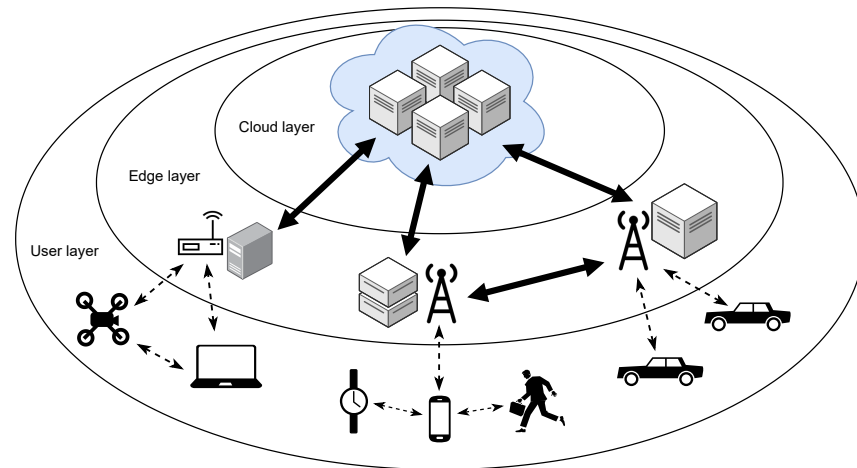


Figure 3.1. Multi-access edge computing. Adapted from [9, Figures 1-6].

- **Mobility.** As end devices tend to move, tasks can be offloaded to different MEC servers via collaboration between servers.
- **Security and privacy.** End devices share their data with MEC servers instead of cloud.
- **Local energy saving.** Edge offloading can increase local IoT device's battery life by 20%–30% [10].

MEC architecture provides a platform for innovation as base stations offer a great infrastructure for evolving edge computing applications for both current operators but also for new third-party operators. As research is being conducted on different use cases, the following list depicts some of the main use cases of MEC [9, Chapter 3]:

- **Computation offloading.** A resource constrained device can offload computationally intensive tasks fully or partially to edge server, increasing performance and battery life.
- **Distributed content delivery.** Content delivery, particularly in the case of video content, can be improved by pushing content delivery network services towards the edge and by utilizing caching schemes on the edge. These reduce backhauling (traffic between edge and cloud) and core network usage.
- **Web performance gain.** Faster access to web content can be achieved by dynamically fetching and caching content based on network usage.
- **IoT and big data.** Data sources can push their data to edge servers where the data is processed first instead of pushing directly to cloud, which reduces latency and the amount of data sent to cloud.
- **Smart city.** MEC enables many "smart city" functions like video surveillance, car-to-car communication and location-based services (advertising, emergency services).

- Application awareness and content optimization. As an example, video encoding may be altered during a video call as a result of network analysis. Another example is caching video content in different MEC servers based on the amount of users in a given area.

4. TEST ENVIRONMENT

This chapter aims to introduce an example MEC computation offloading scenario and how this scenario is applied in a test environment. The example edge offloading scenario utilizes a small nano-drone to offload AI object detection to a remote server by using an edge computing software stack.

This chapter is divided as follows. Section 4.1 depicts a real-life case of an edge offloading need. Based on this case, a test scenario is created in Section 4.7. Before explaining the test scenario, the used test equipment and software stack are introduced. Section 4.2 introduces the used software stack, Section 4.3 the test nano-drone, Section 4.4 the used programming languages and tools and Section 4.5 the server side hardware. The communication flow between an edge device and remote server using the software stack is presented in Section 4.6, after which the test scenario is explained in the test environment terms. Finally, Section 4.8 describes how the standards and regulations from Chapter 2 are applied in the test scenario context.

4.1 Example scenario

Imagine a scenario where some small edge device like a phone or drone wants to use some AI scheme to detect objects in its surroundings. The devices could try to run some AI object detection algorithms themselves but this could be either slow or consume a lot of battery charge. Maybe this wouldn't even be possible with the hardware found in the edge devices. Thus, the small edge device needs to find some other way to do it for them. This is where computation offloading from MECs comes into play.

The small edge devices can offload the heavy workload of an AI object detection to some remote server which runs the AI inference for them. The server returns the results back to edge devices which then can utilize the results. In the case of a drone, the results could be used for flying purposes, for example "follow this person" or "move further away from buildings". In a scenario like this where a drone is controlled by the AI object recognition results coming from the remote server side, the application logic could be something like shown in Figure 4.1.

In Figure 4.1, the drone may do some initialization steps like powering on, initializing

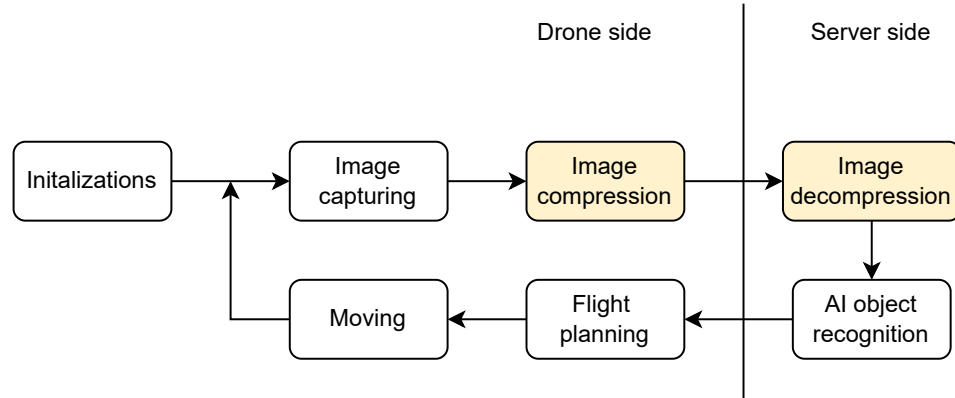


Figure 4.1. Application logic with remote offloading. Image compression/decompression is optional.

server connection and taking off before it proceeds to utilizing the remote server. In the application logic, first an image is captured. This image is then sent to the server side for AI inference in either compressed or uncompressed form. On the server side, the image is decompressed (if needed) and the AI object detection task is performed on the image. The results of AI inference are then sent back to the drone side. The drone may, for instance, plan a trajectory to reach a detected object, after which it actually starts moving towards the object. If some kind of follow task is required, this kind of application loop can be repeated as long as is required.

This is one example scenario where an edge device capable of moving wants to utilize a remote edge server as part of its control decision logic as the small edge device benefits from having increased computational power from the remote server. However, this kind of scenario also presents challenges in terms of physical safety. For example, if a drone wants to follow an object during its flight, how much can the latency be in order to have enough room to avoid collisions? What if the AI object detection takes too long? What happens if/when the server connection ceases to exist? In this thesis, improving physical safety is done in a test scenario following this same example scenario presented here.

4.2 PoCL-R

The test scenario is made possible by utilizing *PoCL* (Portable Computing Language) as the MEC software stack. *PoCL* comes with a remote extension called *PoCL-R* (PoCL-Remote) which allows the offloading of computationally demanding AI object detection from the test drone to a remote server. *PoCL* was chosen as the MEC software stack due to it already being implemented in the test drone platform [11].

PoCL is an open source implementation of the *OpenCL* (Open Computing Language) API with the focus on portability and inter-operability between OpenCL capable devices [12]. OpenCL is a standard for cross-platform heterogeneous parallel computing developed by

Khronos Group [13].

PoCL-R is an OpenCL driver introduced in *PoCL*, capable of forwarding OpenCL commands in a client-server manner. The aim of *PoCL-R* is to provide a low latency and scalable solution for edge offloading. [14]

The *PoCL-R* runtime is implemented as a remote driver (client) and a server daemon. The client driver forwards OpenCL commands to a remote server running *PoCL-R* server daemon over a network, giving access of the server's computational resources to the client. The driver exposes usable remote server's devices similarly to local devices through OpenCL platform API, and using these remote devices is identical to using local devices for the client. A command sent to server daemon gets dispatched to server's OpenCL device driver handling all the server side OpenCL devices. Additionally, the server daemon supports scheduling commands independently and migrating them between multiple interconnected servers (clusters) using peer-to-peer communication. [14]

The test drone utilizes the client driver of *PoCL-R* to offload OpenCL commands to the server daemon running on test setup laptop (see Section 4.5) in order to expand its computational resources using the laptop's dedicated GPU for the AI-inference. The results of this AI-inference are then used in the test drone for flight control purposes.

4.3 Crazyflie 2.1 nano-drone

The test drone in use is called *Crazyflie 2.1*, a small-scale quadcopter manufactured by Bitcraze AB. Crazyflie, shown in Figure 4.2, was chosen to be the test edge device due to its minimalistic nature. It acts as a good example IoT device due to its limited computational capability and ability to move on its own. In the test scenario Crazyflie has an expansion deck attached to it called *AI-deck* which has an onboard camera and is able to run the *PoCL-R* software stack [11]. This makes the Crazyflie a suitable test platform for running the edge offloading scenario on.

Crazyflie has an open-source development platform [15] and supports multiple different expansion decks which can be attached below or above the baseboard. The drone weighs 27 g without any expansion decks and its size (WxHxD) is 92x92x29 mm (motor-to-motor including motor mounts). Example flying time is ≈ 7 minutes and maximum recommended payload weight is 15 g. Battery is recharged through a micro-USB interface.

4.3.1 Baseboard

The baseboard is the part of the drone where motors, expansion decks and a battery are attached. Figure 4.3 shows both sides of the baseboard. Expansion decks are attached



Figure 4.2. Crazyflie 2.1 with an AI-deck expansion deck on top. Also a pencil for scale.

to it by metal pins. The battery is held in place on top of the baseboard with the help of either an expansion deck or battery holder. In Figure 4.2 a tip of the battery can be seen between the AI-deck and the baseboard.

The baseboard has two different MCUs (microcontroller units): STM32 for main board logic and nRF51 for radio and power management:

1. STM32F405 – ARM Cortex-M4, 168 MHz, 192 kB SRAM, 1 Mb flash memory,
2. nRF51822 – ARM Cortex-M0, 32 MHz, 16 kB SRAM, 128 kB flash memory.

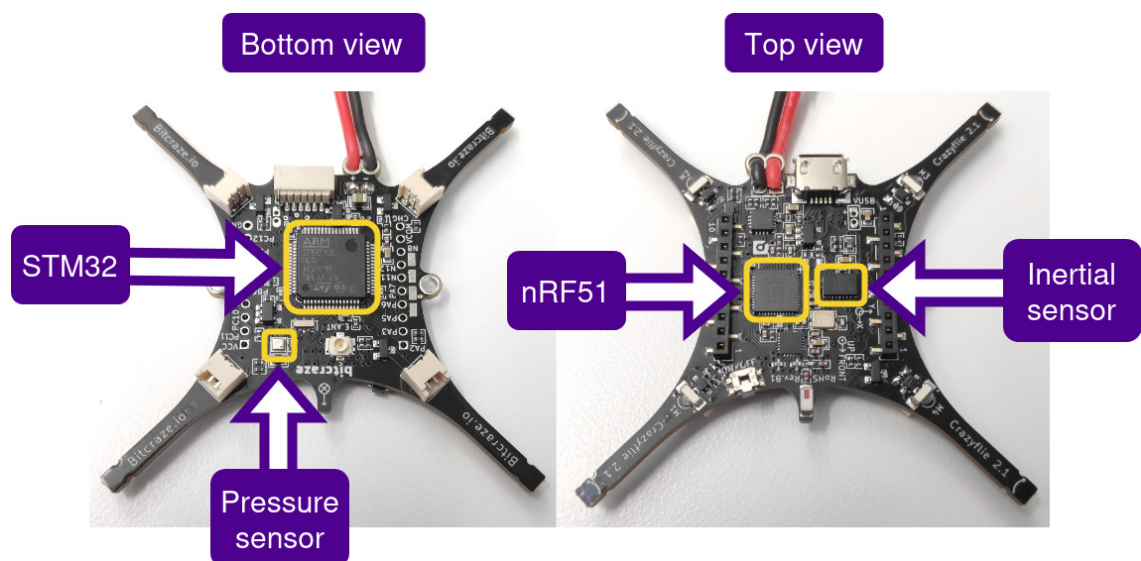


Figure 4.3. Crazyflie 2.1 baseboard from both bottom and top views.

The drone supports flying with low-latency long-range radio communication with Crazyradio PA (USB-dongle) on Windows, Mac and Linux systems and Bluetooth LE (low energy) on iOS and Android systems. The baseboard is also equipped with Bosch's BMI088 6-axis inertial sensor (3-axis accelerometer and 3-axis gyroscope) and BMP388 pressure sensor for inertial measurements when flying.[16]

Motors are controlled by the STM32 via software PID-controllers that are developed by Bitcraze. For an easy access on controlling how the drone flies by the user application, Bitcraze has implemented their own flight control functions in their software development kit.

4.3.2 AI-deck

AI-deck 1.1 in Figure 4.4 is an expansion deck that provides Crazyflie with more computational capabilities, a camera and Wi-Fi connectivity [17]. AI-deck is responsible for controlling the main test application logic in this thesis and has three important parts regarding the test control loop:

1. GAP8, IoT application processor,
2. ESP32, Wi-Fi microcontroller,
3. Himax HM01B0 camera.

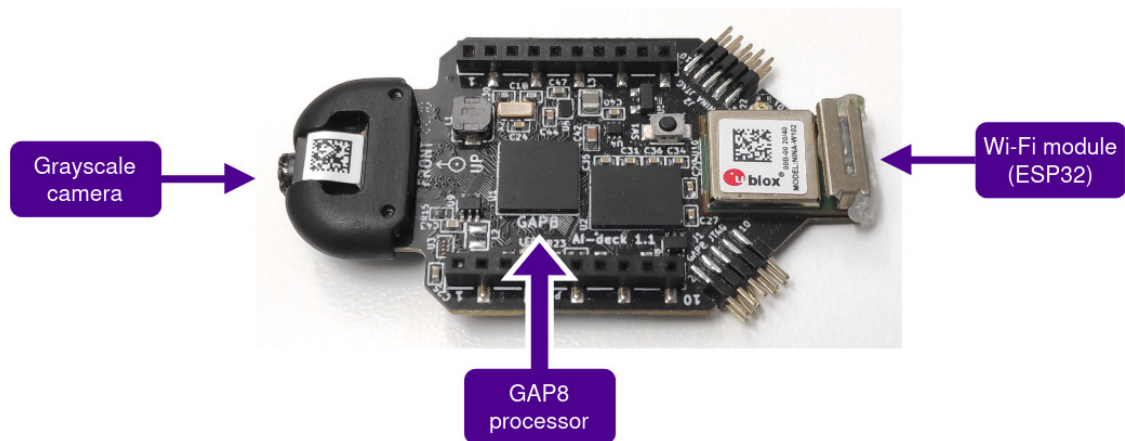


Figure 4.4. *AI-deck 1.1.*

GAP8 is an ultra low power IoT application processor made by GreenWaves Technologies. It has 1+8 RI5CY (RISC-V instruction set architecture based) cores, one of which is designated as the *fabric controller* (FC) and the rest as *cluster cores*. FC is considered as the "main" core and runs at 250 MHz in the test scenario. The cluster cores are made for parallel computing tasks and they are able to run at 170 MHz. GAP8 has internally 512 kB level 2 memory shared between all the cores, 64 kB level 1 memory shared between the cluster cores and then 8 kB for the FC. [18] GAP8 runs the *PoCL-R* software stack.

ESP32 is a Wi-Fi & Bluetooth module located inside a NINA-W102 microcontroller [17] made by u-blox. ESP32 is responsible of routing commands between GAP8 and either baseboard's STM32 (e.g. flight control commands) or a server behind Wi-Fi network through TCP/IP commands (e.g. *PoCL-R* commands).

Himax camera is an ultra low power 320x320 monochrome (grayscale) camera [17]. This camera is used to capture surroundings of the drone and sending them through Wi-Fi to *PoCL-R* daemon.

Both GAP8 and ESP32 can be flashed with custom firmware (application code) via radio frequencies using the Crazyradio PA USB-dongle or through JTAG interfaces with an appropriate programming/debugging device. The two JTAG pins can be seen pointing towards the right in the Figure 4.4 with the upper pins being used to flash the ESP32 and the lower pins to flash GAP8.

The AI-deck has both 8 MB of read/write HyperRAM memory and 64 MB of read-only HyperFlash memory (read-only during execution) that the GAP8 has an access to. There exists tools created by both the GAP8 manufacturer (GreenWaves Technologies) and the Crazyflie manufacturer (Bitcraze) that provide tools for flashing/using these external memories. However, during runtime e.g. HyperRAM is cumbersome to use as the AI-deck has no memory management unit.

4.3.3 Flow deck

Flow deck V2 in Figure 4.5 is an optical navigation expansion deck that is located under the baseboard. With this deck the drone is able to tell when it is moving in any direction with the help of two sensors: [19]

1. VL53L1x, an altitude sensor,
2. PMW3901, an optical flow sensor.

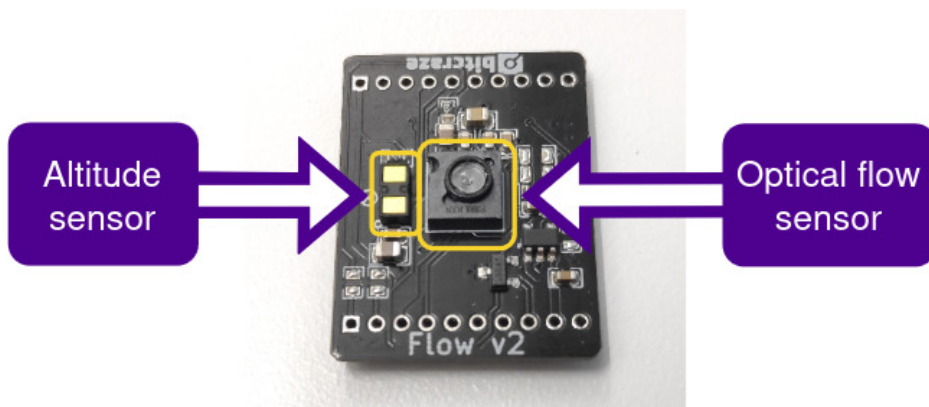


Figure 4.5. *Flow-deck v2.*

VL53L1x is a time-of-flight (ToF) based sensor that is capable of measuring drone altitude

up to 4 meters. ToF sensor works by sending a light pulses towards the ground and measuring the round trip times of the light. This measures the Z-coordinate (height) of the drone to the closest point of the surface below in a cone-like pattern [20].

PMW3901 is an optical flow positioning sensor. Optical flow sensor works by capturing a series of images of the ground and then calculating the motion of certain points in the images, producing an "optical flow" map. As such, the sensor produces relative position to previous images and is able to pinpoint the drone's relative X and Y coordinates to previous points.

With the help of Z coordinate from the ToF sensor and X and Y coordinates from the optical flow sensor, this expansion deck is able to position the drone in three dimensions when flying close to ground. According to Bitcraze, the deck is also eye safe [19].

4.3.4 Capabilities of actuators

Estimating the actuator capabilities is part of ISO 26262-3 Clause 5.4.1. The numeric estimations are important because during risk assessment these numbers are used to decide severity values (potential harm to humans) on different risks. Thus, as the standard is utilized in improving safety, the estimations on Crazyflie actuators are presented here.

When flying, the only actuators in use are motors which rotate propellers attached to them. The motor specifications listed in Bitcraze's website [21] give 14 000 RPM/V, rated voltage 4.2 V and rated current 1000 mA. The battery nominal voltage is 3.7 V and based on testing, the system operating voltage can hit around 4.1 V when fully charged. Using the rated voltage of motor, a safe estimation of maximum thrust output can be calculated. Python code used for calculations can be seen in detail in Appendix A.

Maximum RPM produced by motor:

$$4.2 V \cdot 14\,000 \frac{RPM}{V} = 58\,800 RPM.$$

Rotational frequency n in revolutions per second:

$$n = \frac{58\,800}{60} = 980.$$

Department of Aerospace Engineering from University of Illinois at Urbana-Champaign maintains a propeller database on their website. The database is a collection of different propellers and their measured properties, and Crazyflie propellers categorized as "tractor" and "pusher" are listed there. [22] In Crazyflie, these terms only describe their rotating directions. A tractor propeller produces upward lift when it rotates counter-clockwise di-

rection whereas a pusher produces lift when rotating clockwise direction. Apart from the direction of rotation, the propeller properties deviate a lot from each other depending on if the propeller in question is tractor or pusher type. For this reason, the maximum values that provide the highest thrust are used to estimate the whole actuator, which leads to using pusher specimen 1 to estimate the thrust coefficient of the drone propellers.

A thing to notice is that the website lists measured Crazyflie propeller properties only up to 20 000 RPM but considering that the motors could in theory rotate up to 58 800 RPM (about triple the RPM), heavy extrapolation is required. This may lead to serious inaccuracy but as a counter point, the data seems to form a linear or logarithmic functions on C_T values, especially on higher RPM.

Linear line least squares fitting

Consider a linear equation of a line

$$\bar{y} = m \cdot \bar{x} + b, \quad (4.1)$$

where \bar{y} is the y-intercept, m slope, \bar{x} the variable and b the intersection with y-axis.

Pusher specimen 1 average RPM is

$$\bar{x} = 12\,003.137\dots \approx 12\,000,$$

and average C_T

$$\bar{y} = 0.1557\dots \approx 0.156.$$

To obtain the slope of a line, the least squares method is used:

$$m = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2} \quad (4.2)$$

$$\approx 1.279 \cdot 10^{-6}.$$

Using line equation (4.1) to find the intersection b :

$$b = \bar{y} - m \cdot \bar{x}$$

$$\approx 0.140.$$

Now the approximation of a line is known. Extrapolation of C_T at the point of 58 800 RPM:

$$y_{ex} = m \cdot 58\,800 + b$$

$$\approx 0.216.$$

Logarithmic least squares fitting

Logarithmic least squares fitting with logarithmic function of form

$$\bar{y} = m \cdot \ln \bar{x} + b, \quad (4.3)$$

where the variables are the same as in line equation 4.1.

Coefficients m and b [23]:

$$m = \frac{n \sum (y_i \cdot \ln x_i) - \sum (y_i) \cdot \sum (\ln x_i)}{n \sum (\ln x_i)^2 - (\sum \ln x_i)^2} \quad (4.4)$$

$$\approx 0.0135,$$

$$b = \frac{\sum (y_i) - m \cdot \sum (\ln x_i)}{n} \quad (4.5)$$

$$\approx 0.030,$$

and extrapolation:

$$y_{ex} = m \cdot \ln (58800) + b$$

$$\approx 0.179.$$

Figure 4.6 shows the measured data points with both fittings. Linear fitting works better on higher RPM whereas logarithmic fitting suits better for even lower RPM values.

By using the performance data provided in the website, an estimation of actuator thrust [24] is

$$C_T = \frac{T}{\rho n^2 D^4}$$

$$T = C_T \rho n^2 D^4,$$

where T is force of thrust [N], C_T thrust coefficient (unitless), ρ fluid density of air [kg/m³], n propeller speed [revolutions per second] and D the propeller diameter [m]. By using extrapolation value of 0.216 (from straight line) and air density at 20°C in normal pressure (101.325 kPa) the produced thrust is

$$T = 0.215 \dots \cdot 1.2041 \frac{\text{kg}}{\text{m}^3} \cdot (980 \frac{1}{\text{s}})^2 \cdot (0.046 \text{ m})^4$$

$$\approx 1.12 \text{ N}.$$

Accordingly, using logarithmic extrapolation gives $T \approx 0.925 \text{ N}$. Based on these values, it seems that one actuator is able to produce $\approx 1 \text{ N}$ of force when it is operating at full

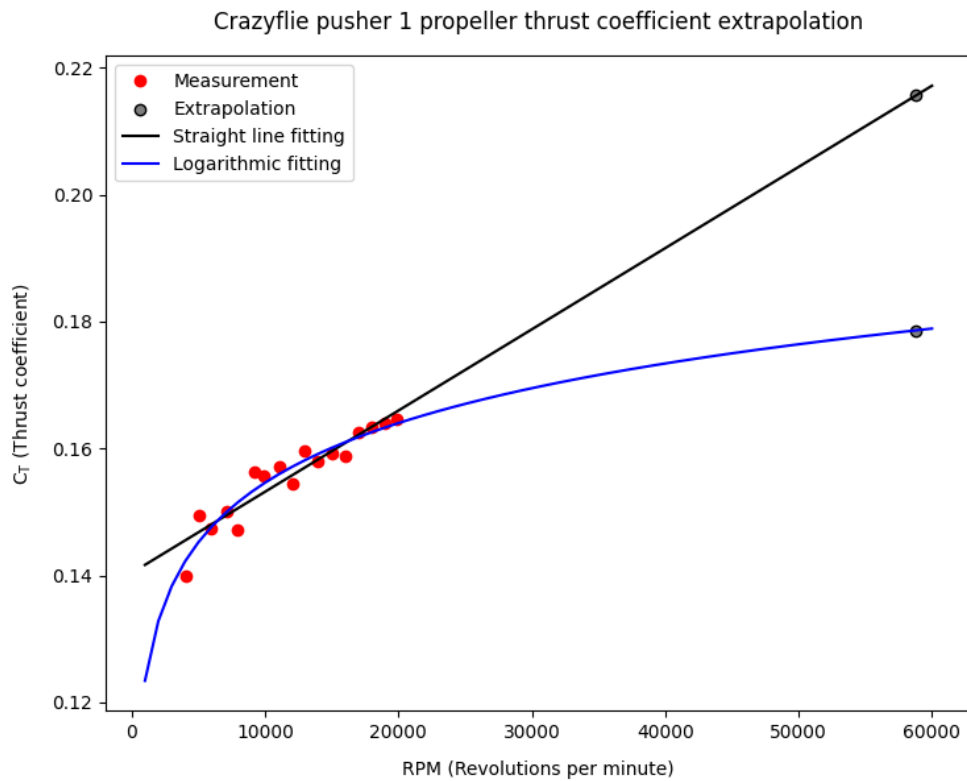


Figure 4.6. Straight and logarithmic fittings on measured data.

speed.

Realistically, the motor does not revolve at this high speed when a propeller is attached to it. The propeller is not tightly fit as it has a round pinhole in the center and the motor shaft is also round from the edges, so sliding between these two objects is possible. As said before, the above fittings heavily extrapolate the data to obtain approximations of thrust coefficients but in this case this is acceptable due to low exertion of force. The fittings are mathematically simple and may not replicate real propeller behaviour at higher RPMs.

Estimating energies upon impact

Thrust is the force that lifts the drone. With the thrust of 4 motors, the overall thrust is around 4.5 N. As a side note, it is good to recognize that the 4 propellers used in the drone consist of 2 thrust and 2 pusher propellers, and this 4.5 N estimation comes from using only 1 type of propeller (the one producing maximum thrust). In reality, the thrust is most likely less than that.

In indoor flight, the estimated force of thrust may last for a couple of meters before the drone hits a ceiling. The drone can also tip over during flight but if that happens, there's a backup system on the drone which forces the motors to lose their power and a restart is required to continue. A realistic scenario where high speeds could occur are during

takeoffs when the drone accelerates itself very quickly, or when the drone is allowed to have high speeds during flight via firmware changes. The maximum X, Y and Z velocities are set in the firmware, each of them being 1 m/s by default.

By following default speed values, the drone could reach 1 m/s in each X, Y and Z-directions. By following a 3-dimensional line, the drone could have an absolute speed of

$$\begin{aligned}
 v &= \sqrt{v_X^2 + v_Y^2 + v_Z^2} \\
 v &= \sqrt{\left(1 \frac{m}{s}\right)^2 + \left(1 \frac{m}{s}\right)^2 + \left(1 \frac{m}{s}\right)^2} \\
 &= 1.732 \dots \frac{m}{s},
 \end{aligned} \tag{4.6}$$

which equals energy-wise (drone weighs 36 grams)

$$\begin{aligned}
 E &= \frac{1}{2}mv^2 \\
 &= \frac{1}{2} \cdot 0.036 \text{ kg} \cdot \left(1.732 \dots \frac{m}{s}\right)^2 \\
 &= 0.054 \text{ J}.
 \end{aligned} \tag{4.7}$$

For comparison when using potential energy, this energy equals to the drone dropping from

$$\begin{aligned}
 E &= mgh \\
 h &= \frac{E}{mg} \\
 &= \frac{0.054 \text{ J}}{0.036 \text{ kg} \cdot 9.81 \text{ m/s}^2} \\
 &\approx 15 \text{ cm}.
 \end{aligned} \tag{4.8}$$

If the drone was to fly unrestricted with full throttle, the drone could reach an acceleration of

$$\begin{aligned}
 F &= ma \\
 a &= \frac{F}{m} \\
 &= \frac{4 \cdot 1.116 \dots \text{ N}}{0.036 \text{ kg}} \\
 &= 124.04 \dots \frac{m}{s^2},
 \end{aligned} \tag{4.9}$$

and hit the ceiling (in a room with 3 meters of height) after

$$\begin{aligned}
 s &= \frac{1}{2}at^2 \\
 t &= \sqrt{\frac{2s}{a}} \\
 &= \sqrt{\frac{2 \cdot 3 \text{ m}}{124.04 \dots \frac{\text{m}}{\text{s}^2}}} \\
 &= 0.219 \dots \text{ s},
 \end{aligned} \tag{4.10}$$

with the speed of

$$\begin{aligned}
 v &= at \\
 v &= a\sqrt{\frac{2s}{a}} \\
 &= 124.04 \dots \frac{\text{m}}{\text{s}^2} \cdot \sqrt{\frac{2 \cdot 3 \text{ m}}{124.04 \dots \frac{\text{m}}{\text{s}^2}}} \\
 &= 27.2 \dots \frac{\text{m}}{\text{s}}.
 \end{aligned} \tag{4.11}$$

With this speed, the energy would be

$$\begin{aligned}
 E &= \frac{1}{2} \cdot 0.036 \text{ kg} \cdot \left(27.2 \dots \frac{\text{m}}{\text{s}}\right)^2 \\
 &= 13.39 \dots \text{ J}.
 \end{aligned}$$

This energy already equals to dropping a 1 kg dumbbell from

$$\begin{aligned}
 h &= \frac{E}{mg} \\
 &= \frac{13.39 \dots \text{ J}}{1.00 \text{ kg} \cdot 9.81 \text{ m/s}^2} \\
 &\approx 1.37 \text{ m}.
 \end{aligned}$$

If the drone was to hit the ceiling and drop to the floor from 3 meters, the energy would be (without any initial speed)

$$\begin{aligned}
 E &= mgh \\
 &= 0.036 \text{ kg} \cdot 9.81 \frac{\text{m}}{\text{s}^2} \cdot 3 \text{ m} \\
 &= 1.05948 \text{ J}.
 \end{aligned} \tag{4.12}$$

4.4 Software development tools

PoCL-R (and *PoCL*) use mostly the C and OpenCL C languages in their implementations. OpenCL C is largely based on the C99 standard but with modifications to allow running parallel programs (kernels) in OpenCL capable devices. Some of its extensions are also utilizing the C11 standard. [25] Additionally, some features that *PoCL-R* offers are written in C++ such as the AI object detection code in the test scenario (see 4.7).

Crazyflie development environment consists mostly of C and Python code. For example, the firmware code flashed into the Crazyflie is C [15] whereas tools used on the PC side, such as the communication library when using Crazyflie PC client (cfclient) [26], are often written in Python.

The application logic that is flashed into the AI-deck is written in C and OpenCL C. OpenCL C is used due to using *PoCL-R*. Also, some external libraries are used in the application logic to power up the AI object detection. These libraries are ONNX Runtime and OpenCV. ONNX Runtime is a cross-platform AI inference and training library from Microsoft [27]. OpenCV is an open-source computer vision library from OpenCV group which is also used for computer vision tasks like AI object detection [28]. On the server side, OpenCV is also used to display the object detection results.

4.5 Remote server

Remote server is a laptop running on Intel i7-1370P (6 performance + 8 efficient cores totaling 20 threads) at 5,2 GHz, Nvidia GeForce MX550 2 GB GDDR6 and 32 GB (4800 MHz) of RAM. The laptop is included with a Wi-Fi module, which is directly in communication with the ESP32. This allows the test setup to not need an additional router, and based on empirical tests, this direct communication without an intermediary router performs much better in communication with the drone.

The server runs an AI inference pipeline when requested by the client. The AI model in use is pre-trained "YOLO-v8n-seg" (available from [29]), which specializes in segmentation tasks. Segmentation tasks specialize in recognizing objects in an image and providing boundaries and regions of where they exist in the image. The model is labelled as "nano" sized meaning that its size is kept minimal (6.74 MB). However, in the test scenario the YOLO model is run using ONNX runtime and thus the model size is actually even larger (13.8 MB) when the model is transferred into a compatible ONNX runtime model.

The problem with YOLO AI models in this test scenario is the required memory for the model. Even the smallest model sizes exceed the amount of memory available on the drone, so they cannot be used directly on it. As a result, the server side is actually

running these models instead of the drone. This way, the drone can still benefit from the bigger AI models through edge offloading.

4.6 Communication overview

The communication flow between different subsystems on Crazyflie and the server is presented in Figure 4.7. Starting off with Flow deck, it uses I2C (Inter-Integrated Circuit) protocol for communication between the ToF sensor and STM32, and SPI between the optical flow sensor and STM32 [30][31][32].

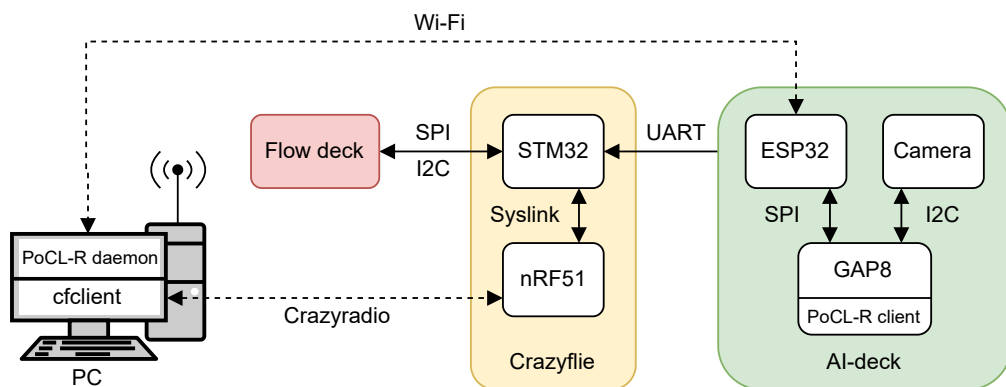


Figure 4.7. Communication flow in Crazyflie and external PC.

Inside Crazyflie, STM32 communicates with nRF51 through Syslink protocol. Syslink is a packet-based protocol where packets are delivered through a serial port at 1 MBaud. [33]

ESP32 on AI-deck and STM32 on Crazyflie communicate through UART protocol (Universal Asynchronous Receiver Transmitter) [34]. Synchronous protocols SPI (Serial Peripheral Interface) and I2C are used inside AI-deck. SPI is used between ESP32 and GAP8 [17], and I2C is used between GAP8 and the camera [35][36].

Figure 4.8 shows how Bitcraze has structured their communications between different MCUs [34]. Essentially, "functions" are labels that can be given to application data packets to separate their uses in the application code. This function information is encapsulated in CPX packets, which are forwarded through transport layer e.g. Wi-Fi. Physical medium is the actual physical method used to transfer data from one MCU to another. As can be seen in the communication stack structure, e.g. UART is considered part of both transport layer but also physical medium in the Crazyflie environment.

CPX protocol (Crazyflie Packet eXchange), developed by Bitcraze, is a hybrid protocol of packet and stream protocols. CPX communication is blocking and can create packet congestion but it can handle large packets. In Crazyflie communication stack terms, CPX protocol is above transport layer (see Figure 4.8) and is used to communicate the application logic between different MCUs on different expansion decks and baseboard. [34]

Functions	– Application data
CPX	– CPX packet
Transport	– CRTP, Wi-Fi, UART
Physical medium	– 2.4 GHz, UART

Figure 4.8. Communication stack structure in Crazyflie.

For example, *PoCL-R* specific packets have their own functions attached to the CPX packet to identify their usage in the *PoCL-R* communication instead of e.g. sending debug data to the *cfclient* (Bitcraze’s PC client software for drone control and flashing) console output. In order to be able to use *cfclient*, *Crazyradio PA* USB-dongle is required to be plugged in on the server side. This dongle uses radio frequencies to communicate with nRF51 on the Crazyflie. *Cfclient* can also be used to control the drone (e.g. with a gaming controller or the software UI) so it is used to send an emergency landing instruction to the drone. On emergency landing the drone shuts down all its motors immediately.

4.7 Test scenario

This section goes through the example scenario presented in Section 4.1 but in terms of the test equipment and vocabulary introduced in Sections 4.2 to 4.6. Additionally, couple of points regarding drone environment is presented at the end of this section.

In the test scenario, GAP8 runs the client version of *PoCL-R* and the PC runs the server daemon. First, the AI-deck’s camera is used to capture the drone surroundings. The captured image is sent over from camera to GAP8 which JPEG encodes it. JPEG encoding is done in order to save bandwidth as the transfer from GAP8 to ESP32 has been observed to be slow on the drone with large data such as an image. GAP8 forwards all *PoCL-R* commands and encoded JPEG image to ESP32, which sends the data over Wi-Fi to the server side.

On the server side a JPEG decompression is performed on the received image. Then the AI object detection by YOLO-v8 is run on the decoded input image and the object detection result is then sent back to ESP32 over Wi-Fi. ESP32 forwards the results to GAP8 which then uses the AI inference results to choose the target object that is used to control the drone. After finalizing the flight plan, GAP8 sends STM32 information about how much the drone should change its *yaw* (rotation) and height to follow an object. STM32 then controls the motors accordingly and drone changes its orientation. This marks the end of one iteration of the application logic and a new round is started.

Figure 4.9 shows the result of the AI object detection. A captured image during flight was given to the AI inference as an input. The output of this object detection is the bounding boxes of each recognized object in the image. After this the drone receives the information about the detected objects and their respective bounding boxes without the original image. The bounding boxes are only added on top of the original image on the server side for the purpose of easy validation for the drone operator but also to save time in network transfers.



Figure 4.9. An image captured by AI-deck with added AI object detection results.

Even though image compression/decompression is optional it helps tremendously with Crazyflie. In the case of sending a raw image of size 324x244x1 (Width x Height x Color-Channels, 79 056 bytes) to the server side, this typically takes ≈ 500 ms. For comparison, JPEG encoding a image with quality parameter of 90 (resulting compressed image sizes of around 12-14 kB) and then sending the compressed image takes ≈ 200 ms in total which is less than half the time of sending a raw image.

4.8 Standards and test environment

This section tells how the standards and regulations from Sections 2.1 and 2.2 are applied to the test scenario when using Crazyflie. Section 4.8.1 explains how the EU regulations are used to categorize Crazyflie and how to take proper safety actions. Section 4.8.2 describes what parts of ISO 26262 are used to improve safety in the test scenario. Section 4.8.3 takes advantage of ISO 19237 to explain how the AI object detection system can be evaluated in terms of safety.

4.8.1 Categorizing Crazyflie

Section 2.2 introduced the EU Regulations on UAS operations. This section explains the categorization of UAS operation when using Crazyflie. Only the "open" category requirements are presented more in detail as the categorization process eventually classifies the test scenario as an open UAS operation. UAS operations classified as "open" must comply with all of the following requirements [1, Article 4]:

1. UAS operations under the "open" category shall meet all the following requirements:
 - (a) Drone belongs to one of the classes C0-4 (based on e.g. drone mass, speed and size [2, Anx. parts 1-5]) of Regulation 2019/945, is privately built or complies with Article 20 of Regulation 2019/947.
 - (b) Drone MTOM is less than 25 kg.
 - (c) Drone is kept at a safe distance from people and does not fly over assemblies of people.
 - (d) Visual line of sight (VLOS) is to be maintained with the drone unless "follow-me" mode is used or some other person acts as "unmanned aircraft observer" (aids the remote pilot in safe flight conducting).
 - (e) Drone must fly below 120 meters from closest point to surface with the exception of overflying an obstacle.
 - (f) Drone does not carry dangerous goods or drop any material during flight.
2. The "open" category is divided into three subcategories and their requirements must be met accordingly depending on the category.

Considering the requirements of open UAS operations, Crazyflie meets the requirements in parts (b)-(f) in the test scenario as long as the drone operator keeps track of the drone during the flight accordingly. Regarding part (a), the drone does not bear any class labels so classification of C0 to C4 is not met. The drone is not either privately built as it is assembled from set parts available from the market [1, Art. 2, (16)]. Thereby to fulfill the requirement of part (a) Crazyflie needs to be specified according to Article 20 of Regulation 2019/947:

- (a) Drone belongs to subcategory A1 as per Part A of the Annex, and the maximum take-off mass (MOTM) is less than 250 g including its payload,
- (b) Drone belongs to subcategory A3 as per Part A of the Annex, and the MTOM is less than 25 kg including its fuel and payload.

The Annex Part A divides "open" UAS operations into three subcategories A1, A2 and A3 [1, Annex Part A]. Subcategory A1 has the loosest requirements whereas A3 has the most restrictive ones. Crazyflie falls into subcategory A1 and must comply with the

following requirements [1, Annex Part A]:

- (1-2) No overflying groups of people and if some persons are flown over, the overflight time must be minimized.
- (3) In "follow-me" mode, drone must not stray away further than 50 meters from the remote pilot.
- (4) Remote pilot has familiarized themselves with the drone manual, or in case of drone being class C1, also taken an online theoretical exam.
- (5) Drone must comply with one of the following specification points:
 - (a) privately built UAS has MTOM less than 250 g and maximum speed less than 19 m/s; or
 - (b) complies with Article 20 part (a); or
 - (c) is marked as class CO according to Regulation 2019/945; or
 - (d) is marked as class C1 according to Regulation 2019/945.

Parts (1) to (4) can be established by the drone operator. In part (5) parts (a), (c) and (d) are not met as the drone is not privately built and has no class labels attached to it. Thus part (b) must be complied with.

Coming back to part (a) in Article 20, Crazyflie is allowed to continue operating under sub-category A1 as the drone is in the meaning of EC conformity of Decision No 768/2008/EC of the European Parliament and of the Council, was placed on market before 1 July 2022 and has MTOM of less than 250 g [1, Article 20, (a)]. The drone bears a CE marking which means that the manufacturer indicates that the product is in conformity with the applicable requirements [37, "Definitions"].

At this point, it is established that operating Crazyflie in the test scenario falls into "open" UAS category. Additionally, it is good to recognize one extra bit of information regarding the drone operator when flying Crazyflie, namely the onboard camera of the AI-deck.

A general rule on a drone with a camera is that the drone operator must register themselves as a drone operator to a local aviation authority, unless the drone is marked as a toy according to Toy Safety Directive 2009/48/EC [38]. However, Traficom has established that the EU regulations do not apply to indoor operations [39]. This means that none of the EU Regulations are actually needed to be complied with nor is the drone operator required to register themselves to Traficom as a drone operator as long as the operations are conducted indoors.

The safety actions and rules presented above highly support a safe operation of drone usage. Even if the test scenario is held indoors, the drone operator should ensure safe operation for all the people present in the test scenario. Thereby the lists above are to be

followed when using Crazyflie in the test scenario.

4.8.2 Applying ISO 26262

Section 2.1.1 introduced ISO 26262 standard. This section further explains how the standard is applied with respect to Crazyflie. Figure 4.10 shows a subset of all 12 parts of the standard as these are directly related to the product development. Supporting processes (Part 8) and vocabulary (Part 1) are also included in the Figure as these are utilized throughout the development process.

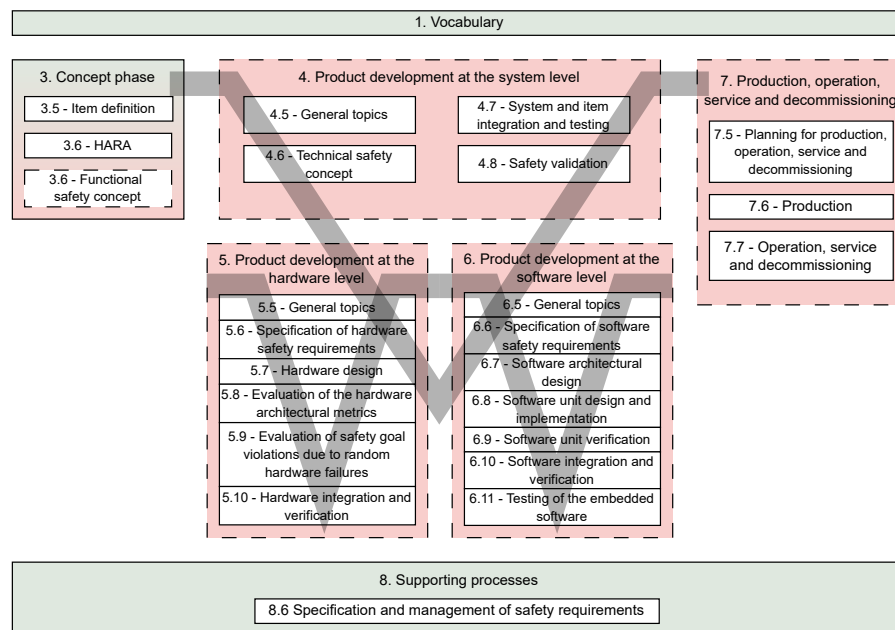


Figure 4.10. A subset of ISO 26262 series of standards forming a V-model. Adapted from [3, Figure 1].

These standards are interconnected via a V-model as is shown in the image. The hardware and software level developments are also interconnected via a V-model as they complement each other. For example, parts of the hardware level designs and requirements need to be taken into account during the software level development.

Additionally, the colors in the image present what standards and their clauses are applied in Crazyflie to improve safety. Green means that the clause or standard is applied and red means that they are not utilized. This is also represented by solid and dotted borders around the standards with solid meaning that the part is applied and dotted that it is not. For example, only some parts of ISO 26262-3 Concept phase is utilized while ISO 26262-4 is completely left out. Also, only the safety requirement specification clause from ISO 26262-8 is applied and the rest of the clauses that are not utilized are not shown in the Figure.

Part 3 is utilized the most to understand the underlying drone system. Part 3 lays the

foundation of applying all the subsequent parts of the standard and also creates a functional understanding of the system and its requirements. This part helps to define an item in the system to base hazard analysis and risk assessment on (HARA). This leads to creation of functional safety concept.

Part 4 describes how to apply the resulting functional safety concept from part 3 to incorporate safety functional safety requirements to make a technical safety concept and system architectural design. Technical safety concept would be used as a basis for product development at both the hardware and software levels but in this thesis part 4 is excluded.

Part 5 explaining hardware level development, part 6 explaining software development and part 7 explaining production, operating and decommissioning stages are also excluded. The reason for leaving these parts out is that ISO 26262 is a huge standard and complying with all the parts would be highly time-consuming task for the purpose of this thesis in this kind of research environment that is not even dealing with automotive systems. A fundamental understanding of the system is already reached by applying part 3 when specifying general safety goals.

4.8.3 Applying ISO 19237

Section 2.1.2 introduced the ISO 19237 standard. The standard set the minimum requirements for a "pedestrian detection and collision mitigation systems" (PDCMS) in vehicles in addition to the test procedures and passing criteria of these tests.

Starting off with the minimum PDCMS requirements, the standard summarizes the minimum requirements as information about the range to pedestrians, the motion of pedestrians, the motion of the subject vehicle and the driver commands and actions [6]. Based on these categorizations, the driver commands and actions do not apply other than the drone operator being able to instruct Crazyflie to perform an emergency landing via cfclient.

Motion-wise Crazyflie would need motion information from both pedestrians and itself. Crazyflie already monitors its motion: current X, Y, Z coordinates, the pitch, roll and yaw information and also the thrust on all motors. However, the difficult part is observing pedestrians and estimating their motions.

Pedestrian detection can only be performed by the AI object detection on the server side. The current AI-model in use (YOLOv8 segmentation model) excels in detecting objects in an image but it does not estimate the motions of detected objects like object speed or direction of movement. Ultralytics (creator of YOLO AI-models) has a wide range of AI-models some of which specialize in object tracking. Ultralytics has created an article on speed estimation with their tracking AI-models [40] but these tracking models are not used in this research project at the moment. There is work underway to generalize the

usage of ONNX Runtime in this research project so adopting new models will be easier in the future. For the time being though, the object detection model is only used in this thesis but using new AI-models is also a possibility for future works.

A difficult task is also measuring the range to pedestrians. With the current expansion deck configuration on the Crazyflie, there is no possibility to measure any distances on the horizontal plane crossing Crazyflie. Only the closest vertical distance to the ground is measured but that is not usable to detect e.g. pedestrians right in front of Crazyflie where the AI-deck camera is pointing.

There exists methods to estimate distances based only on the camera feed for example by using a simple pinhole camera model with an object with known size. This is not really generalizable as it only estimates one singular object's distance to the camera. All other objects (also within the same class) that the AI-inference recognize would also require the size information to be known beforehand. For detecting surrounding objects on a vehicle often some kind of LiDAR (light detection and ranging) system is used to accurately measure distances to objects around the vehicle. But as it stands, no sensor data of surroundings is available and image processing estimations are not applied in this thesis but could be worth investigating in future works.

The test procedures are test tracks for vehicles with certain illumination, mock of a person, vehicle and pedestrian speeds, ground surfaces and so on. These test setups are explained in detail in the standard but the general idea is as follows: a vehicle is moving a straight line and a pedestrian will move in front of the car path under certain illumination. When the vehicle is 18 meters away from the pedestrian, the PDMCS should recognize this situation (the pedestrian is crossing a street right in front of the vehicle) and start slowing down the vehicle. The vehicle should stop or reduce its speed sufficiently within the 18 meter range in order for the test to pass. The test pass criteria follow the test procedures in terms of vehicle context. The test passes if either of two different criteria is met:

1. The speed of the vehicle is less than 10 km/h upon impact with a pedestrian, or
2. the collision is avoided completely.

Considering the drone environment, similar test criteria can be applied. Section 6.3.2 shows the performance measurements when applying the test criteria on Crazyflie.

Regarding the whole standard it gives good points to pay attention to if Crazyflie was to move in 3 dimensions. 3-dimensional movement is still under investigation if it is worth to work on on this research project. Currently Crazyflie only moves in Z-axis and changes its yaw so PDCMS is not even implemented as of yet. Anyhow, these performance metrics are something to keep in mind if the research project continues with this Crazyflie or with a bigger and more capable drone.

5. ADAPTING ISO 26262-3

This chapter is dedicated to adapting the automotive standard ISO 26262 part 3. The standard is used the most in improving physical safety in the test scenario, which is why this whole chapter is spent on adapting the standard.

Section 4.1 brought up some key physical safety issues like the network latency and loss of connection to a remote edge server in an edge offloading scenario. Section 5.1 defines an ISO 26262 item called "pedestrian detection system" (PDS) in which the AI object detection and edge offloading parts belong to. This section describes the basic principles of the PDS. Section 5.2 further elaborates on the PDS by explaining what requirements the PDS poses on other items, and conversely, what the other items require from the PDS. After that, safety goals with respect to test scenario are formed in Section 5.3.

5.1 Defining an item

Defining items is a prerequisite to subsequent tasks in ISO 26262. Example items in vehicles could be "adaptive cruise control" or "lane keeping assist". In this thesis, the drone environment considers only the "pedestrian detection system" (PDS) item in detail (see Figure 5.1) while other items, such as "flight control system" from Section 5.2, are only named and given a short description of their functionalities. Understanding the basic functionalities of these other items complement the understanding of the core functionality the PDS offers, that is, locating people by using AI object detection.

Figure 5.1 depicts how the PDS item is defined. The main functionality that the PDS offers is locating people in given input images. This is done by using AI object detection. The PDS also includes supporting process of JPEG encoding as the drone produces JPEG encoded images. Client-server connection is a requirement for the AI object detection to work as the AI-inference is done on the server side in the edge offloading test scenario. Without client-server connection, the PDS will not work. The PDS will operate on Crazyflie's captured input images and produces an output of AI detected objects. The output will be used back in Crazyflie for flight planning.

This section follows the Clause 5.4.1 in ISO 26262-3 to define the requirements imposed on the PDS. According to the Clause, "requirements" contain everything from actual legal

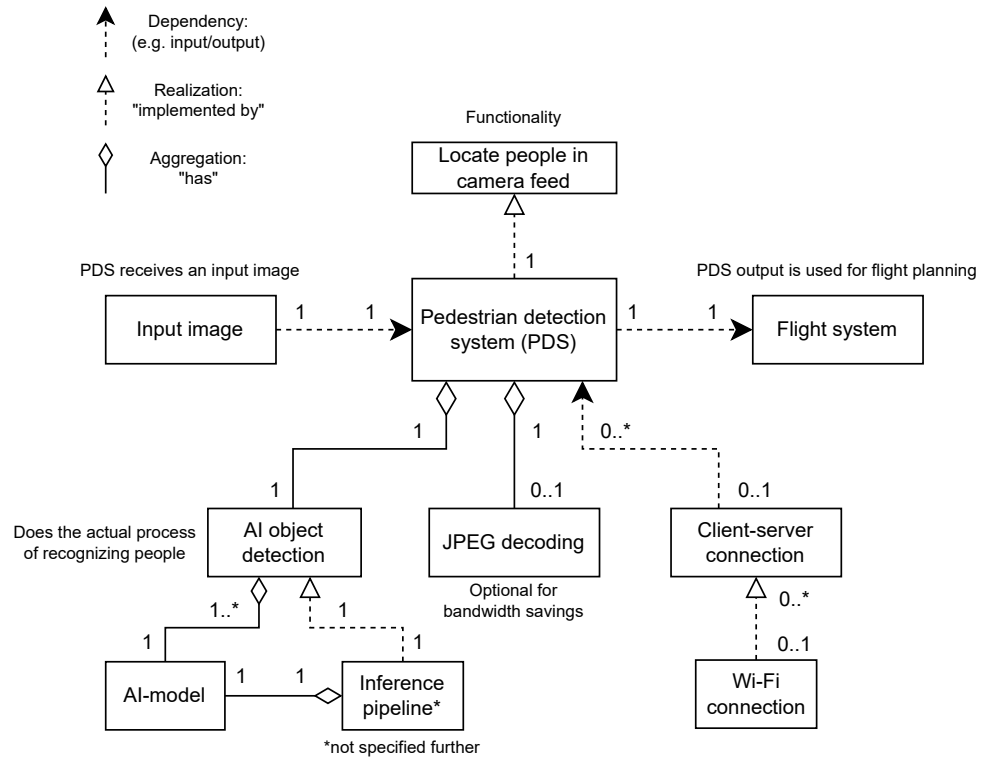


Figure 5.1. General concept of the "pedestrian detection system" (PDS) item in UML.

requirements to explaining the behavior of the item and consists of 6 points in total [5, Cl. 5.4.1]. Two of these points were already handled, namely the estimations of actuator capabilities in Section 4.3.4 and the EU legal requirements on Crazyflie in Section 4.8.1. The rest of this section deal with the other four points to define requirements of the PDS.

Section 5.1.1 explains the functional behavior of the drone in terms of different operating modes which impact the PDS. Section 5.1.2 describes the required quality, performance and availability metrics from the PDS. The constraints of the PDS, such as the functional dependencies on other items and the operating environment, are explained in Section 5.1.3. The consequences of known failure modes and hazards are presented in Section 5.1.4.

5.1.1 Functional behavior

When the PDS is on, the drone can be in four different stages. These stages are represented in Figure 5.2.

1) Flight mode off – The drone will not use its motors at any stage when it is powered on. This stage is primarily in use when testing new additions in software. The main application logic is still in use and the drone actively offloads captured images on the AI-deck camera to edge server for AI object detection.

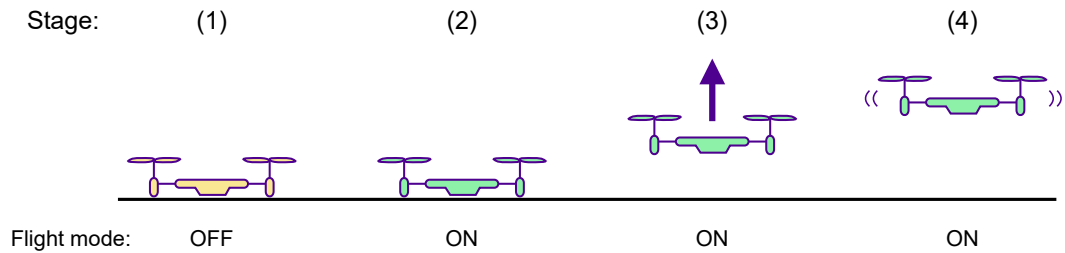


Figure 5.2. Four different stages of drone flight

2) Flight mode on, no takeoff yet – Flight mode on means that the drone is allowed to use its motors for moving around. During this stage, the drone should function exactly the same as if the flight mode was off. This means that the main application logic is up and running. The only difference to stage 1) is that the drone will take off after a specified amount of frames have been processed. The intention of this phase is to give the drone operator enough room to start the drone (press a physical button on the baseboard), lie the drone on a flat surface and see that the drone has done its initialization steps and the AI object detection is working as intended before the drone takes off.

3) Flight mode on, the drone is taking off – The drone will start spinning its motors and moves up in the air to a set altitude. During the takeoff, the application logic is working and people are being detected. However, during this stage the drone may not start following people. This is to ensure that the drone reaches a sufficient operation altitude. Small lateral movement is allowed as the drone is very small and thereby susceptible to disturbances in the air.

4) Flight mode on, takeoff is completed and the drone is flying – At this stage, the drone is free to fly. This means that the drone can adjust its position vertically (change its altitude) but also rotate itself (change its yaw). The application logic is working and tells where people are in the captured images, which is used to determine which person the drone turns towards. The drone must not displace itself laterally too far from the starting position as there are no sensors to detect e.g. walls, which could lead to collisions in 3-dimensional movement.

5.1.2 Quality, performance and availability of functionality

The person detection functionality is the core function of the PDS allowing drone movement. Consequently, people detection functionality needs to be on only when it is wanted that the drone moves. Without any targets to follow, the drone will levitate at its current altitude without reorienting itself. With such behaviour, there exists a natural safety mechanism in the application logic to avoid collisions.

When the drone wants to move, it is expected that the PDS is available when the control

parameters are being calculated. The required availability can be determined by doing test runs and seeing how much calculation time is spent on the drone side and how much on the server side.

The drone side performance increases the total execution time in image capturing, JPEG encoding and moving as these are the parts that only the drone is responsible of doing. Moving consists of GAP8 sending high level flight control parameters, like absolute yaw and height change from current position, to baseboard's STM32. STM32 calculates a flight trajectory and starts executing it with the help of motors.

Network conditions affect the sending image, JPEG decompression, AI object detection and reading results as these parts utilize *PoCL-R* and it needs to send OpenCL commands and data buffers between the client and the server, which introduces latency. During test runs in this thesis and also in the previous thesis [11], Crazyflie hinders the client-server communication more than the server side most likely because of the power saving features of ESP32 [11, Ch. 6.4.2].

On the server side the JPEG decompression and AI object detection increase the execution time. This is due to them being directly limited by the server side computational capabilities. Better hardware leads to faster AI inference and image decompression.

To determine the required availability parameter, a test run was performed with Crazyflie and the results of this run are shown in Table 5.1. Based on the results, the average processing time of one image is 407.1 ms when 300 images were inferred. Out of this total time, 110.7 ms is spend on purely network transfers when sending the JPEG compressed image and reading the AI object detection results. The drone side operations account for 191.8 ms and the server side 104.6 ms. The parts where the PDS is involved are highlighted in the table.

Table 5.1. Average times in a test scenario run when running AI inference on server side.

Frames	Averages (ms/frame)						
	Image capture	JPEG enc.	Sending image	JPEG dec.	AI obj. detection	Reading results	Moving
300	57.6	79.5	89.0	18.5	86.1	21.7	54.7

Based on the test run it can be deduced that the PDS (server side) must be available at least 215.3 ms, equal to 53%, of one image iteration cycle to serve its function. This availability parameter includes the network transfer and the JPEG decoding times so the actual AI inference part accounts only 40% of the total PDS execution time.

5.1.3 Constraints, dependencies and operating environment

The PDS is only in use after Crazyflie has booted up (gets energy from battery), self-tests during boot up have passed, all drone systems have been initialized (e.g. expansion decks and motor controller systems), a Wi-Fi connection has been established with an edge server running *PoCL-R* daemon and all the application logic initializations have succeeded. If any of aforementioned functionalities fail, the PDS will not start.

Operational environment will be indoors with varying amount of objects and people surrounding the drone. If there are multiple people present in close proximity of the drone when it is operating, the AI object detection of the PDS will recognize these people and they become potential objects to be followed meaning that the drone might turn to face them.

Lighting of the room has an impact on the drone's camera as the camera has a feature called AEG (automatic exposure and gain). This feature auto-adjusts some internal camera parameters to best capture images in the current environment. The adjusted image quality subsequently affects the AI-inference and how well the drone performs while operating. However, this feature calibrates the parameters only during the first initialization process when the camera is started, which leads to a question on where the drone should be facing when it is powered on. With some experiments on camera parameters, AEG enabled is sufficient when the drone is not directly facing a bright light source (e.g. a window with sun shining through) but the best results are obtained when AEG is off, the environment is controlled and the camera parameters are manually set. However, when using the drone in different environments, AEG on is a must to receive a satisfying performance from the AI object detection in the PDS.

5.1.4 Consequences of known hazards

A collection of Traficom's confirmed drone hazards was shown in Section 2.3. As the PDS will provide output results that will be used to fly the drone, these known hazards and risks are considered here.

The list contains a lot of risks that affect outdoor flights and human operators. Considering an indoor flight without an external operator flying a drone like in the test scenario, the following hazards are deemed relevant: taking off and landing, drone payload, motor failure, radio signal interference and battery charge level. Regarding the PDS, only the radio signal interference and battery charge level have an impact on its functionality due to connection losses or the drone running out of energy.

Running out of battery is self-explanatory as no system can work without energy. The radio signal interference, or in other words, quality of client-server connection is a big

issue if the quality is bad. This has a direct impact on the usability of the PDS in an edge offloading scenario as bad connection may lead to e.g. loss of packets and retransmissions. This reduces the reaction time on the drone side in the case of avoiding a collision, for instance.

The PDS itself has a natural behavioral shortfall due to the AI object detection. The quality of the AI inference depends on the underlying AI model but also different iterations of the inference produce different results on the same input image. When looking at a singular person in an image, the certainty, location, bounding box size and even the class might change between inference runs. This will lead to misclassifications or no recognitions at all.

5.1.5 Impact of actuators

Section 4.3.4 presented some numeric estimations on the Crazyflie actuator capabilities in a typical operating environment. This section introduces some additional points regarding the actuators.

Based on the approximations, it would seem that definitely the most dangerous part in indoor flights would be when the drone takes off if the speed is uncapped. The second most dangerous scenario would be if the drone was to drop down from the air. The least dangerous part is when the drone hits an object while flying with a limited speed of 1 m/s (per X, Y and Z-direction) as is the default case.

However, it is important to notice that the approximations only take the energy of hit into account. Depending on how the drone actually hits a person or an object the consequences of the impact change. For instance, if the drone is flying and the propellers hit a person's face or eye in the worst case, the consequences of the hit are more serious than the drone dropping on an inanimate object on the ground. Also, as the expansion decks of the drone are attached to sharp metal pins on the baseboard a hit with pins first could be much more harmful to people than e.g. a motor stand first hitting a person.

It is also good to recognize that these approximations do not consider air resistance that slows down the drone nor the air dynamics when the drone pushes its way through the air. Also, the motors behave ideally and the propellers according to extrapolated propeller data. The speed of the drone is capped in normal flight conditions and thus the estimated speed during takeoff will not be reached. Realistically, the energies would be smaller but as a general idea of which phases of the drone flight possess the most danger, these calculations provide a good basis on improving safety.

5.2 Item interactions

This Section follows the ISO-26262-3 Clause 5.4.2 to provide an understanding of item's interaction with other items and elements in the system. This includes defining what is required by other items from the PDS but also what the PDS requires from these other items.

Section 5.2.1 explains what the PDS requires from other items in the test scenario. Section 5.2.2 describes how the correct and incorrect behavior of the PDS will affect other items. Section 5.2.3 explains what other items require from the PDS.

5.2.1 Item requirements from other items

Figure 5.3 shows what the PDS requires from other items to serve its function. Items and elements represented by dotted lines in the image are external to the PDS which provide the necessary functionalities for the PDS. Additionally, the items part of the application logic are enclosed with a solid border while external systems are outside of the rectangle.

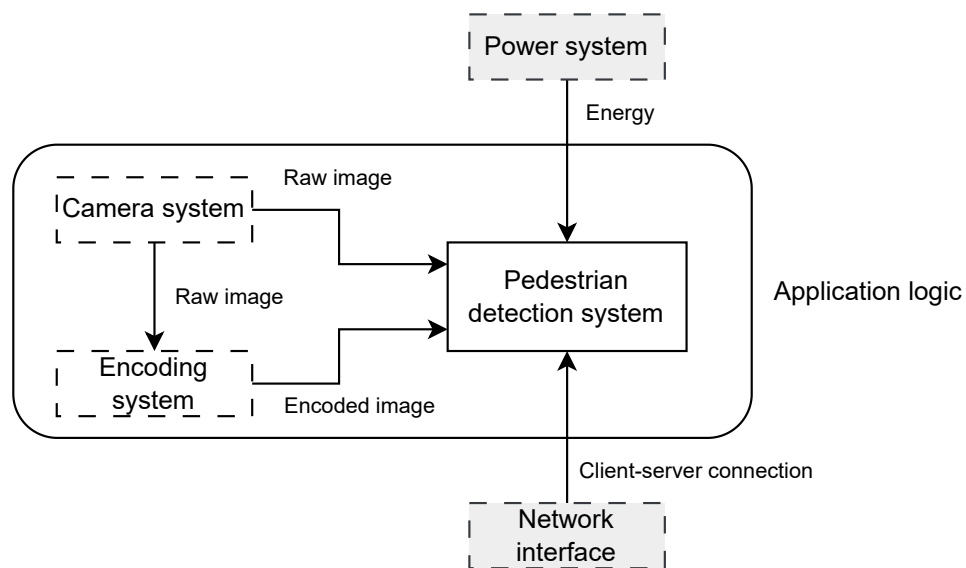


Figure 5.3. The PDS requirements from other items.

The PDS requires an input image. This is provided by the camera directly as a raw image or through an encoding system that produces a JPEG image. As the PDS includes a JPEG decoding system, both inputs are valid. At the moment, switching between direct raw image input and JPEG encoding on the fly is not implemented so either one must be used throughout the flight.

The size of the input image does not matter as the PDS itself modifies the input image to be of size 640x480 for the AI object detection. Neither does the color encoding matter as the PDS transforms that to BGR by itself as well. However, the original image size must

be conveyed to the PDS as that information is needed for the image resizing.

Regarding the image encoding, JPEG encoded images are preferred to raw images in the test scenario due to bandwidth savings as the data transfer from the drone to server is slow (see table 5.1). The drawback of this is worse image quality as the used JPEG encoding scheme is lossy. This affects the AI object detection quality but is deemed worthy due to faster network transfer times.

For the best inference results, the image should be as in Figure 5.4 as the used AI-model recognizes objects best when they are like how humans would see the world. This means that e.g. a person's legs are at the bottom of the image and not at the top. The Figure also shows the coordinate system used in OpenCV tools which are used in the AI-inference.

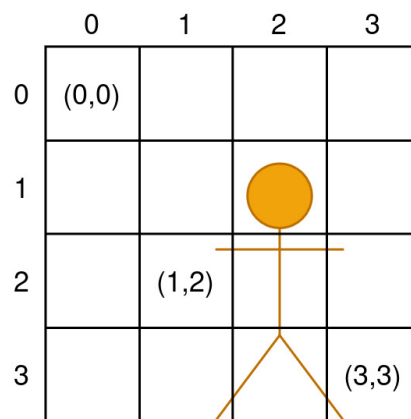


Figure 5.4. Optimal positioning of a person and the coordinate system used in OpenCV.

Considering requirements from parts outside of the application logic, there are requirements from a power system and network interface card. Naturally, the PDS requires some energy to run and on Crazyflie this is provided by an external battery. Considering the test scenario with a laptop PC, the energy comes from either a battery or an external power source. Network interface is also mandatory as it provides both the drone and the server the capability to interact with each other via Wi-Fi.

5.2.2 Assumptions on effects of item behavior

The pedestrian detection functionality is required to be able to move the drone. It is a core mechanic in order for the drone to behave as intended as no human will assist the drone when it is operating in the test scenario.

When the PDS is functioning properly, the PDS relays input parameters to a flight control decision system (FCS). The FCS then processes the PDS output and creates high level control commands which are then sent to STM32. STM32 handles the actuator system and uses these control commands to make the drone fly. Occasionally, the PDS might not yield any object detections from the AI inference and the drone does not move. Under

normal working conditions, the drone operator can still verify that the PDS is working by assessing both the cfclient (drone) and server logging data to ensure that the PDS is working as intended.

In the case of the PDS not working, no inputs will be provided to the FCS, which means that the drone will not re-orient itself at any point after taking off. Typically this is due to failures in the initialization steps even before the drone has taken off, or due to losing client-server connection during flight. In the test scenario, the failure of the PDS not providing the FCS any input values is considered as a system failure.

If a human operator is flying the drone via a controller while the PDS is active, these two items do not interfere with each other. However, human input will interfere with the actual flying parts of the drone system. Both the FCS makes and a remote pilot will create high level flight commands that are relayed to the actuator system. The drone behavior in this case is unknown. However, this is an issue to be solved when human-computer parallel control is required in the test scenario. As it stands, this is not a requirement from the PDS to solve and neither it is a feature to be implemented for the test scenario.

5.2.3 Requirements from the item

Figure 5.5 shows what is required from the PDS. These requirements come from other items and parts in the drone system. In the figure, the dotted rectangles present external items of the PDS. The requirements posed by the items are also described next to the item.

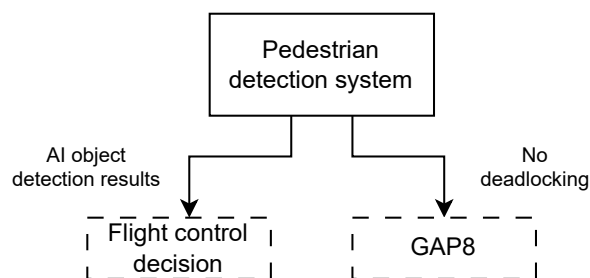


Figure 5.5. The external requirements from the PDS.

A flight control decision system (FCS) requires an output from the PDS. The output is a collection of classified objects or a special output that tells FCS that zero objects were found. The output is a C-array as shown in table 5.2. If any objects were detected, these detections are used to control the drone flight. If zero objects were found, the drone will not change its orientation but will hover in place in the air.

The array is a C-array with each array element being an unsigned integer. First element of the array tells how many instances were detected in the input image. After that comes a list of all detected instances. Each instance consists of 6 parameters, and instances

Table 5.2. Output form of DNN inference.

0	number of detections
1	class id
2	confidence value
3	bounding box x-coordinate
4	bounding box y-coordinate
5	bounding box width
6	bounding box height
7	class id
...	...
n	bounding box height

are sorted by confidence value (best confidence value at the top of the list). If there are no objects detected, the C-array is one integer long and the value of that integer is 0.

GAP8 runs the whole application logic including imaging, encoding/decoding and flight control system. All these systems demand computing power from the same CPU. Considering application logic software loop, none of these systems may stop the execution loop in such a way that the execution loop does not advance. This means that the PDS must provide the FCS an output whether or not any detections were found. Consequently, camera and encoding/decoding systems assume that the PDS is providing an output to the FCS, as the next iteration of the drone application logic loop will not start if the execution stopped during the earlier iteration (e.g. camera will not take a second image if there was a problem during the first iteration of the main control loop execution). This further bolsters the importance of properly handing out output information to the FCS.

5.2.4 Item functionality in different operational scenarios

Considering the functional behavior stages, the PDS behaves the same way regardless of whether flight mode is on or off in the Crazyflie. The only effected part is the FCS and moving parts which are already out of the scope of the PDS item. For the largest part how well the PDS can satisfy its functional requirement depends on the following factors: the quality of the input image, the used AI-model, the client-server connection quality and the server side computational capabilities.

First, the input image plays a role in determining the quality of AI inference. The camera takes grayscale images of size 324x244. An ideal image for the AI inference would be 640x480 RGB image, but this is not achievable in Crazyflie so extrapolation is required to enlarge the images, which further reduces the inference quality. There exists an RGB camera for the Crazyflie that provides 324x244 RGB image, but it's not usable at the

moment due to memory constraints. Using an RGB camera might be possible by utilizing HyperRAM, but the tradeoff is in increased network transfer times, which is why including an RGB camera is left out.

The camera has internal parameters which are set at the initialization phase of the camera during boot-up. These parameters control the image quality and thus directly affect the AI object detection quality. AEG (see Section 5.1.3) option is highly recommended on new environments, which leads generally to satisfactory image quality but even better image quality may be reached through setting the camera parameters manually for a controlled environment. However, this process is cumbersome and a controlled environment is often not the case when using the drone so AEG is enabled instead.

In case of JPEG encoding/decoding, the quality of the input image is lower depending on how much compression was done. This further affects the AI inference quality with heavily compressed images. The reason for using compression is reducing the network transfer latency overhead as higher compression leads to smaller image sizes. Image transfers over a network take a lot of time so reduced data amounts are desirable (see Sections 4.7 and 5.1.2).

While flying, the drone is always a little bit in motion. This motion is upwards/downwards, sideways or even forwards/backwards. This motion always affects the camera as it is fixed on the drone. This leads to inaccuracies in the image quality as the shutter speed of the camera is finite and not instant. When the drone is not flying, this effect does not matter that much as the drone can be laid down on a table, for example.

The used AI-model is a pre-trained model from Ultralytics. The training data uses lots of RGB images so grayscale input images are not ideal to produce the best results. Also, if the training was done with images captured with the AI-deck's camera, the results would be even better. The AI-model in use also plays a factor in the performance of object detection. The model used in this test case is the smallest YOLOv8n model memory-wise. The larger models have more variables and perform better in inference tasks.

The client-server connection depends on the distance between the drone and the server but also on the existing network traffic. Network congestion is a well-known issue in client-server architecture when there's a lot of traffic in the network which can increase latency in client-server communication. Crazyflie and the edge server communicate through Wi-Fi network which is using 2.4 GHz frequencies. Generally speaking, this frequency range can be crowded with traffic from lots of different devices so the distance between Crazyflie and the edge server must be kept small. Also, Wi-Fi communication is not the ideal way to communicate to targets far away but in the indoor test scenario it works well enough especially when Crazyflie is designed to be a low-power system.

The server side hardware also has an impact on how fast the JPEG decompression

and AI object detection are run. Better hardware leads to faster execution times, which leads to the drone being able to process more frames per seconds. Subsequently, better hardware contribute positively to the physical safety as the drone is more responsive to its surroundings.

5.3 Hazard analysis and risk assessment

In Sections 5.1 to 5.2 the pedestrian detection system (PDS) item was defined. This Section uses the PDS definition to perform a HARA (Hazard Analysis and Risk Assessment) in order to identify and classify hazardous events caused by malfunctioning behavior of the item. Safety goals are formulated with corresponding ASILs (Automotive Safety Integrity Levels) to mitigate or prevent the hazardous events in order to avoid realization of risks.

This Section follows ISO-26262-3 Clause 6.4.2 to understand the different operational situations and operating modes in which the item's malfunctioning behavior will lead to hazards. According to ISO-26262-3 Clause 6.4.2.3 Note 1 the causes of item's malfunction do not need to be considered in HARA for analysing the malfunctioning behavior. Hazards resulting only due to item behavior with no any item failures are also out of the scope of ISO-26262-3. In this test case, an example of correct item behavior possibly leading to a hazard is misclassification of an object in the AI object detection.

5.3.1 Situation analysis and hazard identification

A hazardous event is a combination of a hazard and an operational situation. An operational situation is any scenario that can happen during a vehicle's lifetime, e.g. a drone is taking off. A hazard is defined as a potential source of harm that is caused by malfunctioning behavior of an item. A harm is defined as physical injury or damage to a person. [4] As a reminder, the item under scope is called "pedestrian detection system" (PDS). A malfunctioning PDS can mean that the PDS does not receive any input images to perform AI-inference on or the PDS does not provide FCS any object detections, it provides erroneous values or the whole PDS is unavailable.

Section 5.1.1 introduced four different operational situations where the drone may be. Situations 1 and 2 do not possess any hazards in case of malfunctioning PDS. No harm to humans can occur when the drone is not flying or spinning its propellers.

During operational situation 3, malfunction in PDS can lead to one hazardous event, namely the PDS becoming unavailable.¹ In this case it is good to understand how the

¹ISO-26262-3 HARA does not consider potential causes of item malfunction. But as an example, in this test case unavailability of PDS is most likely caused due to loss of client-server connection as the server runs most of the PDS.

drone flies.

During a flight, a new control input overrides the previous control input. This means that every time that the drone is set to move somewhere, a new trajectory is calculated to reach the destination point, and the drone starts moving accordingly. If no new control inputs are given the drone will continue to fly towards its destination using the trajectory it calculated when the input was given. When reaching the destination, the drone hovers at its location until a new input is given.

If the PDS becomes unavailable during takeoff, the drone will elevate into a set altitude (controlled in firmware) and then hover in place. This poses no harm to humans even if the drone is unresponsive to the application logic because Bitcraze's cfclient can be used to directly command the Crazyflie through its baseboard communication systems, omitting the application logic running on GAP8 in the AI-deck completely. Cfclient can be used to make an emergency landing at any point during the drone flight. However, the emergency landing only shuts down the motors, which will make the drone drop to the ground. From small heights the drone drops on its legs quite often but higher drops always pose more potential damage to the drone. The emergency system has been tested during testing to work accordingly.

Operational situation 4 has two hazardous events: 1) the PDS becomes unavailable or 2) the PDS provides incorrect values to FCS. In the case of the PDS becoming unavailable, the same scenario happens as in operational situation 3. The PDS providing incorrect values means that the resulting AI-inference bounding box values are out-of-bounds of the given input image resolution. This is considered as a fault as the AI-inference has no information outside of what the camera is able to capture.

If the PDS provides erroneous values to the FCS, there will be a mismatch between the drone heading and the actual object location. Also, if the error in the values provided by the PDS are huge, the drone might try to reach its set destination too quickly. The drone movement API function has a parameter of "duration" which controls how long the drone should take to reach its set destination. If this value is set to be too low (drone tries to reach its destination too quickly) and the orientation parameters (X, Y, Z, yaw) are unrealistic to hit during that time period, the drone is in danger of instability, falling over and crashing.

Looking into the above operational situations, two hazardous events are recognized for the PDS in total:

- PDS becomes unavailable,
- PDS provides erroneous values to FCS.

As mentioned above, the consequences of hazardous events can be unresponsiveness of the drone, instability of flight or in most severe case crashing. Unresponsiveness is the

least severe consequence and leads to the drone hovering at place safely.

The second most severe hazard is instability. Instability means unexpected or unwanted deviation from the current trajectory (e.g. unnecessary swaying). Instability can happen at any point during the drone flight. Often times the drone is able to correct its position in the air but occasionally it is not the case. If the drone is not able to fix its position the drone will tip over and crash. This poses a hazard for humans. If the drone sways away from its trajectory, it can fly closer to humans and other objects. The drone has no sensors that can detect how close it is to some object (other than height measurement from the bottom of the drone), which means that the drone is not aware of its surroundings and cannot avoid hitting them.

The biggest impact is due to crashing. A crash can lead to damage to people, drone and the environment. The drone is a lightweight device and its propellers are made of plastic. During testing, multiple propellers were broken due to crashing even if the crash happened from small heights due to drone flipping over. Also, the antenna of the Wi-Fi module in the AI-deck was prone to break due to its fragile nature. This was fixed by hot gluing the antenna in place.

Damage to humans must be minded. The drone does not transfer much energy when it crashes with default speeds but the parts in the drone can be sharp. If the drone crashes, a propeller might break into smaller pieces or even fly off intact. It is good to recognize that these small parts may fly into somebody's eye in the worst case scenario. It has been observed during testing that the propellers have flied to the ceiling during a crash.

5.3.2 Classification of hazardous events

Each hazardous event is classified with respect to severity (S), probability of exposure (E) and controllability (C). These classifications are used to determine the vehicle's ASIL. [5, Clause 6.4.3] Severity estimates the potential of harm in 4 classes as shown in table 5.3.

Table 5.3. Four classes of severity (S)

Class	S0	S1	S2	S3
Description	No injuries	Light and moderate injuries	Severe and life-threatening injuries (survival probable)	Life-threatening injuries (survival uncertain), fatal injuries

Probability of exposure estimates how often the item under scope malfunctions in a representative sample of operational situations. This 5-level classification is shown in table 5.4.

Table 5.4. Five classes of probability of exposure (E)

Class	E0	E1	E2	E3	E4
Description	Incredible	Very low probability	Low probability	Medium probability	High probability

Controllability defines how probable it is for someone (e.g. the driver on a vehicle, or in our case, a drone operator) to gain control of a hazardous event to avoid harm. Table 5.5 shows the 4 controllability classes.

Table 5.5. Four classes of controllability (C)

Class	S0	S1	S2	S3
Description	Controllable in general	Simply controllable	Normally controllable	Difficult to control or uncontrollable

Based on the classification details above, each hazardous event is given a class in all three categories. The hazardous events are listed in table 5.6 with their respective class levels.

Table 5.6. Classified hazards

#	Hazardous event	Severity	Exposure	Controllability
1	PDS unavailable	S0	E3	C1
2	Erroneous values from PDS	S1	E1	C1

First off, let's consider a hazardous event of the PDS becoming unavailable. As previously mentioned in Section 5.3.1, the drone starts hovering after reaching the end of its current trajectory. At this point, the drone does not pose any harm to humans as long as the drone is kept in VLOS. The drone can be controlled back to ground via cfclient either smoothly with a controller or roughly by performing an emergency landing. In case of an emergency landing, it is easy for the drone operator to control the drone surroundings in such way that no harm will occur to humans. The biggest damage, if any, will be physical damage to the drone itself. For exposure, the PDS becoming unavailable is not that rare as the PDS system works via client-server connection. Based on tests, the Wi-Fi connection between the drone and a server can be weak and occasionally this leads to dropping the link between the drone and the server. For these reasons the given values are severity S0 (no injuries), exposure E3 (medium probability) and controllability C1 (simply controllable).

In the hazardous event of PDS providing erroneous values to the FCS, the consequences of this kind of hazard would make the drone unstable and potentially crash. As established with this nano-drone, the drone does not possess that much of a danger to humans but can still lead to some light damage in very unfortunate situations. This hazard is given severity level of S1 (light and moderate injuries).

Chances of faulty values happening very slim but it has been observed to happen in certain situations. One situation where this kind of hazard is proven to happen is when the drone camera is wrongly initialized and the camera image is upside down. In this case the drone operator should see the error when confirming the initializations before the drone takes off. The drone operator is responsible for maintaining a VLOS with the drone as per the EU regulations. Arbitrary erroneous values have not been observed and combining these reasons, the exposure level becomes E1 (Very low probability).

With respect to controllability, if the drone operator has a way to confirm images in real-time as is in the test scenario, the drone operator can decide to abort the drone takeoff e.g. by commanding the drone to make an emergency landing. In this scenario the drone purely disables itself on the ground and is not able to takeoff or fly before a complete restart of the drone including a power-off and power-on sequence. With this in mind, the drone operator should be able to tell very early on if the camera feed is faulty so the controllability level is assigned as level C1 (simply controllable).

5.3.3 ASIL determination

With regards to hazardous event classification done in Section 5.3.2, the following table 5.7 combines the severity, exposure and controllability levels with one ASIL for each identified hazard.

Table 5.7. *Classified hazards with their deduced ASILs*

#	Hazardous event	Severity	Exposure	Controllability	ASIL
1	PDS unavailable	S0	E3	C1	QM
2	Erroneous values from PDS	S1	E1	C1	QM

As seen from the table 5.7, both hazardous events are given ASIL level of QM. Unavailability of the PDS is assigned a QM level even though severity S0 requires no ASIL [5, Cl. 6.4.3.4]. The biggest reason for these low ASIL levels are the severities of the events. For example, in vehicles the actual impact of a crash is much higher than on a very lightweight nano-drone that is used indoors. Just based on the vehicle mass and speed the energies are a lot bigger in vehicles than drones which leads to more serious harms for humans and ASILs are designed according to those scenarios. Nevertheless, the ASILs for these hazards could be at worst of level A or B. Reaching these levels would require a crash with a human with a very highly probable and uncontrollable situation. As it stands, the test environment is very controlled and simple, which keeps the ASILs low.

5.3.4 Safety goals

Each hazardous event that is given an ASIL shall be determined with a *safety goal*. Safety goals describe the top-level safety requirements for an item and lead to functional safety requirements to avoid unreasonable risks for each hazardous event. Safety goals do not describe the technological solutions but the functional objectives in implementation-independent manner. Safety goals receive the same ASIL as the hazardous event it was derived from. ISO-26262-8 Clause 6.4.4 is used to help specifying the safety goals. [5]

Specifying the safety requirements will be done according to ASIL A as there are no methods for QM levels. Both the recognized hazards are of level QM and therefore the safety goals inherit the QM level. According to table 1 in ISO-26262-8 Clause 6.4.1, informal notations for safety requirements specifications are highly recommended and more formal notations are only recommended [41].

The following list describes the safety goals:

1. Safety requirement #1, hazardous event # 1, ASIL: QM.
The PDS shall be available at least 53 % of the total drone operating time in the air (see Section 5.1.2).
2. Safety requirement #2, hazardous event # 2, ASIL: QM.
The PDS output must be validated to be within the input image resolution boundaries before usage in drone control.

If the standard was fully applied these safety goals would be used in creating a "functional safety concept" with respect to Clause 7 in part 3 [5]. This leads to creation of functional safety requirements on different parts of the system under investigation. After that, other parts of the standard such as product development at the system, hardware and software level may be applied. However, applying these other parts of the whole ISO 26262 standard is an accumulative process which requires all the previous steps considered before reaching e.g. software level development. For the purpose of this thesis in this research project, applying all parts of the standard is considered highly excessive. The safety goals formed in this Section give enough understanding on *improving* the safety in the test scenario.

6. ACTIONS TO SATISFY SAFETY GOALS

Chapter 5 defined the PDS item by applying ISO 26262-3. The results of that chapter were safety goals in Section 5.3.4 that are used in this chapter to explain what are the measurements taken in the edge offloading test scenario to improve safety. Section 6.1 explains how the safety goal #2 is taken into account while Section 6.2 focuses on safety goal #1. Section 6.3 explains different tests that were performed to see how Crazyflie can comply with ISO 19237.

6.1 Output validation

Safety goal #2 is "The PDS output must be validated to be within the input image resolution boundaries before usage in drone control." This is one of the more trivial improvements in the application logic pipeline as a simple boundary check in the C-application logic code is sufficient to fulfill this safety goal. The output validation is performed after the drone receives the AI object detection results before the output is used in the FCS as shown in Figure 6.1.

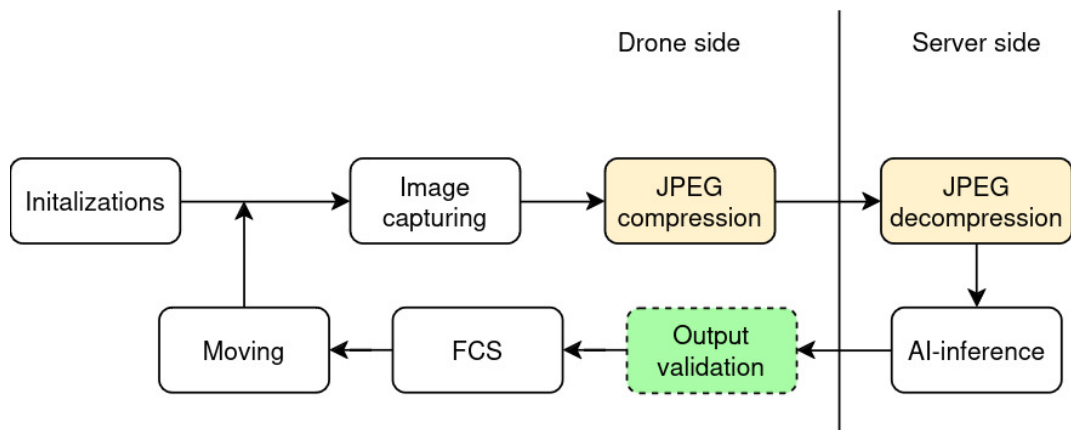


Figure 6.1. Output validation is added to application logic to satisfy safety goal 2.

Verifying that this safety requirement is met is left for the drone developers. This is easily verifiable on the software side as the developers have an easy access to the source code.

6.2 Local fallback

Safety goal #1 is "The PDS shall be available at least 53 % of the total drone operating time in the air." To meet this requirement in the application logic, a local execution branch was created alongside the remote execution.

The local execution branch will be utilized if the server connection is lost during execution. As of now, dropping or activating the server connection is only simulated with a variable on/off value due to easier development at this current stage. The code on ESP32 could be modified to monitor the Wi-Fi signal strength and tell GAP8 when the signal is too weak for proper client-server communication but implementing this is left out at this stage of the research project.

Figure 6.2 shows how the remote and local execution branches coexist in the system. The simulation variable "server connection" tells whether to use the remote or the local execution.

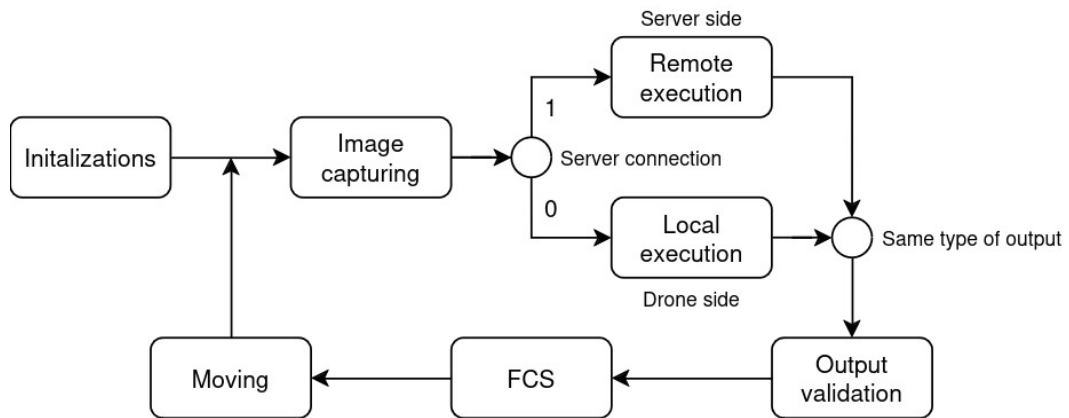


Figure 6.2. Application logic with remote and local execution branches.

Considering the implementation on Crazyflie, there are examples of how only the GAP8 processor successfully runs these models with the toolkits provided by GreenWaves Technologies. Also, a small ≈ 750 kB example AI image recognition (classifies a full image into a category, not to be confused with actual object detection within an image) using 2 classes was actually run on the Crazyflie *without* the remote execution software libraries or application logic. However, this is not sufficient in the test scenario as the edge off-loading is the main point of interest in the research project. So, work began on trying to combine both a larger AI-model and the Crazyflie development platform but then problems appeared due to memory constraints and bugs in the software tools.

The cumbersome usage of external memory (see Section 4.3.2) requires a major restructuring of the application code in order to use the external memory efficiently as the runnable code needs to always be loaded into the L1/L2 memories of the GAP8 processor. There exists a helping tool which slices larger application programs (over 512 kB

which is the maximum size in L2 memory) into chunks that are stored in HyperFlash and HyperRAM and then these chunks can be loaded into the L2 memory as needed. However, even if the code was restructured, there exists a big question mark whether or not the code will even run as the flashing process is buggy with the Crazyflie.

The AI image classification example required flashing via radio as the JTAG flashing does not work for some reason (known bug, see [42]). However, when flashing via radio if the flash image is too big, the flashing process hangs at 99% for some time and then the Crazyflie restarts itself and indicates via LED-blinks that self-tests have failed and the AI-deck code cannot be run. This is also a known issue from the past [43] and has been brought up multiple times in the Github issues Section but this problem seems to persist in our test drone even with the latest Crazyflie base-board firmware updates. The issue may lie in the image flash size which becomes multiple megabytes with the smallest pre-trained object detection models. All these issues lead to the decision to abandon the local AI-inference implementation for now and focus on something else.

The implemented solution is a collaboration with the server AI-inference. Whenever the server connection is lost, the last image of a pedestrian is used to create a template image of an object to follow. Then for the next camera images this template is used to match the best fit from the next image. When the best fit is found, the drone turns towards it. The application logic in this local execution case is presented in Figure 6.3. As shown in the Figure, the template is actually saved in the FCS as it iterates through all the identified objects on the AI-inference results. If a person was found, the template image is saved.

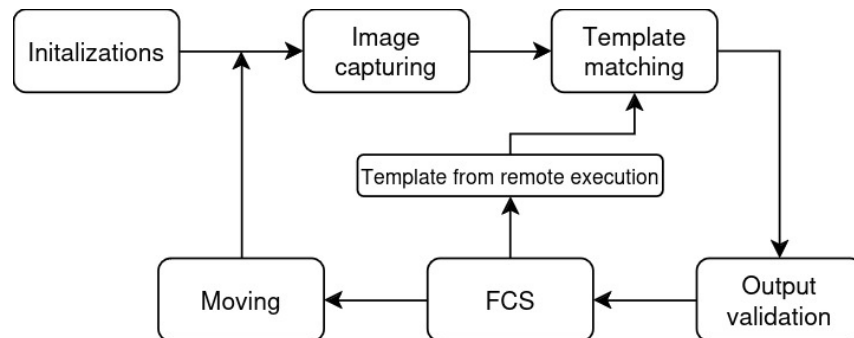


Figure 6.3. Application logic when using the local execution.

However, there exists plethora of problems with this solution. First of all, a server connection must be established before this method may even be used. This is due to the drone not having a template image already of a person to be followed. Secondly, the template image may contain a lot of "extra" data such as pieces of the environment around the person. Also, when the template image is saved, it is only from one specific angle of one specific spot of a one specific person. This means that following the person might prove difficult if e.g. the person's orientation changes between images. Another question that is raised is that if the drone was to fly in 3-dimensional space, how would the drone

know how far it is from any given object without any sensor data. All things considered, this template matching is a rudimentary solution for a hard problem to fix in the current timeline of the project with the current equipment so this solution was deemed adequate for now.

6.3 Remote execution time budget

Section 4.8.3 introduced ISO 19237 collision avoidance test criteria for vehicles that determine if the PDCMS in use is sufficient to completely avoid or negate the impact of collisions with pedestrians. This section introduces numerical values in terms of latency and initial Crazyflie flight speeds to comply with the test criteria for the PDS.

Section 6.3.1 shows numerical values from latency tests that track minimum, average and maximum latencies on different operations. Section 6.3.2 uses the latency values to calculate theoretical maximum safe flight speeds for the drone. Section 6.3.3 reflects on the results.

6.3.1 Latency measurements

The first test is a basic ping-pong test in which the drone sends a one byte OpenCL write command to the edge server. This is used to measure the smallest possible latency with the drone. In the second test a raw image of 79056 bytes is sent to the server. This accounts for 324x244x1 grayscale camera image being sent to the server. The third test encodes an input image (79056 bytes) to a JPEG image and sends it over to the edge server. The JPEG image sizes vary over time due to lossy JPEG encoding, so the JPEG image sizes are also presented. The final test considers reading a typical AI object detection output of 31 bytes (with maximum of 5 detections, see 5.2). The server side operations are excluded from the tests as better server side hardware will make the execution time closer to zero and network latency with Crazyflie is the most important measurement in this case.

The tests are performed at four different distances: 1, 10, 20 and 30 meters away from the edge server. The tests start when the drone starts executing the OpenCL command *clEnqueueWriteBuffer* which initiates the network transfer from host (drone) to remote server. The tests end when the server has replied and the drone can move on from the *clEnqueueWriteBuffer* command to the next application level command. When performing 1200 operations during a test, each command has a possibility to have a rounding error of 2 microseconds due to workarounds with the Crazyflie timers. This can attribute to total of 2.4 milliseconds error in the worst case for a given result value.

Table 6.1 shows the latencies when sending one byte over to the edge server from different distances. The values are also represented in Figure 6.1, which shows the average

values alongside minimum and maximum values.

Table 6.1. Latency measurements when sending 1 byte.

Distance (m)	Latency (ms)		
	min.	avg.	max.
1	29.2	47.3	93.2
10	29.8	48.1	91.0
20	31.4	50.0	107.1
30	31.1	51.7	140.5

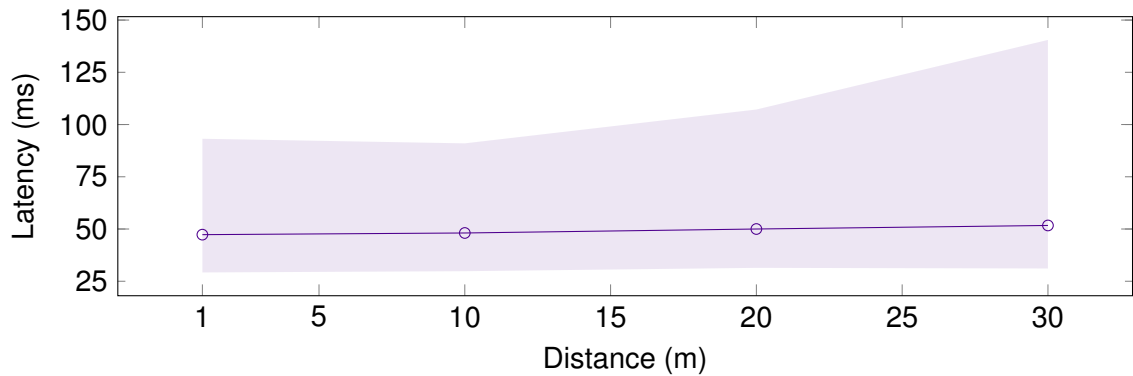


Figure 6.4. Latency measurements between 1 and 30 meters from the edge server when using `clEnqueueWriteBuffer` command to send one byte from Crazyflie to the server. Filled area depicts the range between minimum and maximum latencies at given range. The line represents the average latency. Jitter increased when the distance to the server increased.

Table 6.2 shows the results of the three remaining tests. In the Figure there are three columns which indicate the three different tests. "Raw image" shows the latencies when sending a raw image, "JPEG" constitutes the JPEG encoding and image sending parts, and "Reading" shows the latencies when Crazyflie receives the AI object detection output.

Table 6.2. Latency measurements with different operations in different distances.

Distance (m)	Raw image			JPEG			Reading		
	Latency (ms)			Latency (ms)			Latency (ms)		
	min.	avg.	max.	min.	avg.	max.	min.	avg.	max.
1	539	628	852	143	179	214	13.2	20.1	54.5
10	537	623	735	131	178	1756	13.8	24.5	84.9
20	536	625	846	134	203	2794	13.4	25.0	684
30	539	617	869	136	190	1922	13.6	22.8	373

Figure 6.5 depicts the "Raw image" results.

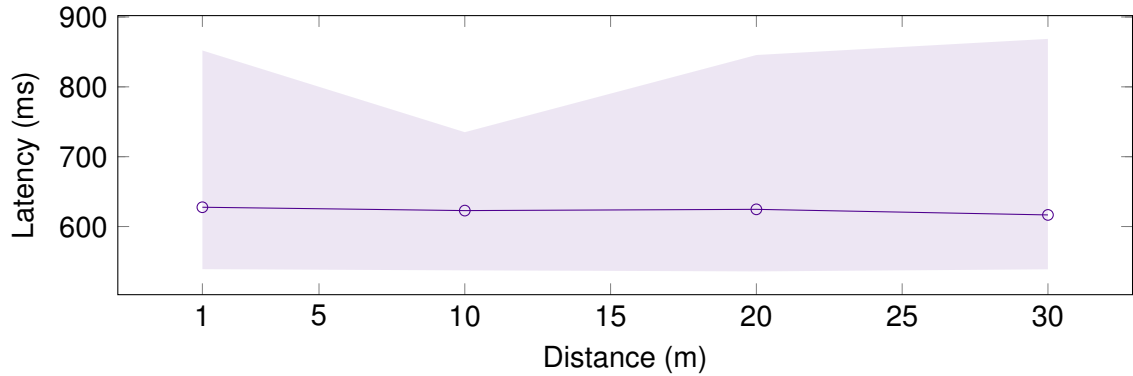


Figure 6.5. Test "Raw image" - Latency measurements between 1 and 30 meters from the edge server when sending a raw image of size 79056 bytes. Filled area depicts the range between minimum and maximum latencies at given range. The line represents the average latency. During this test, jitter was found to be at similar levels even with longer distances.

Figure 6.6 and Table 6.3 show the "JPEG" result.

Table 6.3. JPEG sizes.

Distance (m)	Size in bytes		
	min.	avg.	max.
1	7105	10411	11124
10	5976	9122	10411
20	6188	9332	10493
30	6178	9156	10066

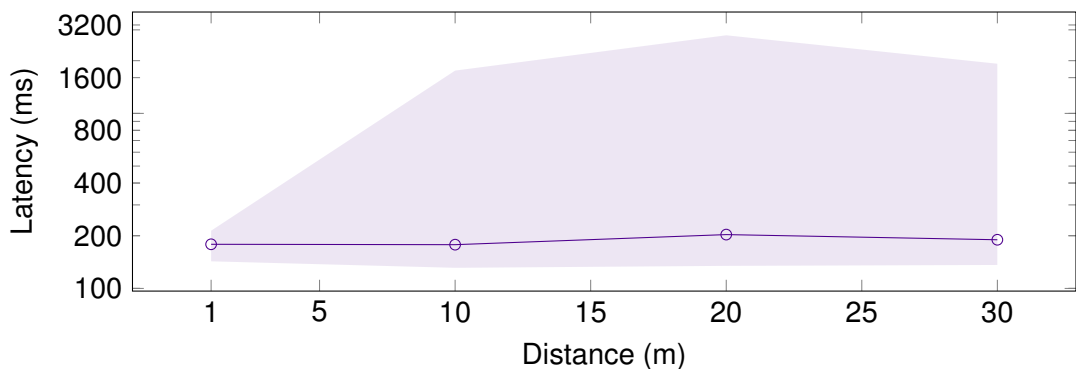


Figure 6.6. Test "JPEG" - Measurements of JPEG encoding time and network latency between 1 and 30 meters from the edge server. The sent JPEG image size varies between around 6 and 11 kilobytes. Filled area depicts the range between minimum and maximum latencies at given range. The line represents the average latency. Jitter vastly increased when the distance to the server increased.

Finally, Figure 6.7 shows the "Reading" results.

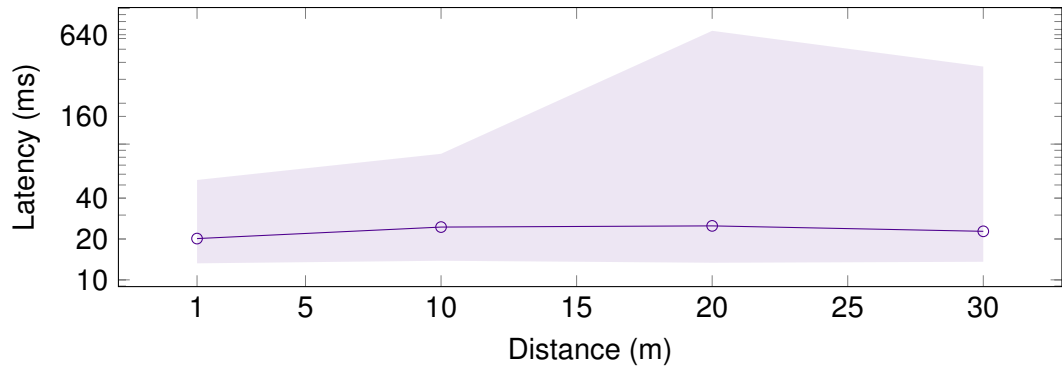


Figure 6.7. Test "Reading" - Latency measurements between 1 and 30 meters from the edge server when receiving AI object detection output of 31 bytes. Filled area depicts the range between minimum and maximum latencies at given range. The line represents the average latency. As with the JPEG test, jitter increased vastly with distance.

6.3.2 Maximum initial speeds

Let's consider the test criteria introduced in Section 4.8.3. At the moment it is unknown how fast the drone is actually capable of moving. A safe operating speed of 1 m/s is given in the firmware but this can be changed by the application developer. Also, it is unknown how fast the drone could brake in the air as that also depends on the speed of the drone. The drone should also stay stable during the braking so that the drone is able to reduce its speed accordingly. To estimate if the drone is capable of following the test criteria in ISO 19237, some approximations are shown here. The approximations are based on estimations on actuator capabilities in Section 4.3.4, the latency values in Table 6.2 and the moving time in Table 5.1.

There are several assumptions concerning the estimations. The drone is estimated to deaccelerate in 45° angle with full force while being completely stable. The fluctuations in air are not considered either. Also, it is assumed that the drone is able to initiate full braking immediately after the moving phase (the drone moves itself into 45° angle). In a 45° angle the deacceleration can be calculated using Equation 4.9 by dividing the force into x and y components and using the x component in the equation.

The following equations can be used to calculate maximum initial speed of the Crazyflie in order to pass the test criteria:

$$s(t) = s_0 + v_0t + \frac{1}{2}at^2 \quad (6.1)$$

$$v_1 = v_0 + at, \quad (6.2)$$

$$s_{total} = s_{react} + s_{brake} \quad (6.3)$$

where $s(t)$ is the total distance covered with respect to elapsed time t , s_0 the initial displacement, v_0 the initial speed, a the (de)acceleration and v_1 the speed at time t . s_{total} is the total distance covered when combining reaction distance s_{react} and braking distance s_{brake} .

Using Equations 6.1, 6.2 and 6.3 gives an equation that can be used to calculate the initial velocity v_0 :

$$\begin{aligned}
 s_{react} &= v_0 t_r, \\
 s_{brake} &= v_0 t_b + \frac{1}{2} a t_b^2, \\
 t_b &= \frac{v_1 - v_0}{a}, \\
 s_{total} &= -\frac{1}{2a} \cdot v_0^2 + t_r \cdot v_0^2 + \frac{v_1^2}{2a} \\
 v_0 &= \frac{-t_r \pm \sqrt{t_r^2 + \frac{1}{a} \left(\frac{v_1^2}{a} - 2 \cdot s_{total} \right)}}{-\frac{1}{a}}.
 \end{aligned} \tag{6.4}$$

Table 6.4 shows the maximum initial speeds with respect to minimum, average and maximum latencies from Table 6.2 to pass the ISO 19237 tests. The reaction time t_r values are calculated by using the maximum latency of each respective min./avg./max. column in Table 6.2 to account for worst case scenarios. The reaction time t_r includes sending an image, receiving the results and moving the drone to the braking position of 45° angle against the direction of current drone motion.

For instance, to calculate the maximum initial speed using raw images with average latencies, the values 628 ms of sending and 25.0 ms of reading are used along with the assumed constant moving time of 54.7 ms. The results are shown on column "Raw image" on row "avg." for final speeds of 0 and 10 km/h in Table 6.4. All the calculations are done in Python code and can be seen in appendix A.

Table 6.4. Maximum initial velocities with different operations.

		Raw image m/s		JPEG image m/s	
Required final speed		0	2.78	0	2.78
With latency	min.	24.15	24.20	40.62	40.69
	avg.	21.66	21.70	36.62	36.69
	max.	10.78	10.81	5.05	5.07

6.3.3 Reflecting on results

Considering these results, it is important to notice the restrictions of Crazyflie and the ideality of these estimations. Crazyflie has weak performance in terms of latencies and especially the variation when sending JPEG images. This is a big restricting factor for Crazyflie to fly safely when offloading control decisions as the maximum speed can drop to $1/7$ of the *average* value in the worst case scenario, resulting in a drop from ≈ 36 m/s to ≈ 5 m/s. Also, most likely these flight speeds cannot be reached as the drone needs to tilt itself forwards, which results the camera pointing towards the ground the faster the drone is allowed to fly. On fast speeds the PDS AI object detection will be given more input images of the ground than surroundings, which doesn't allow the PDS to work properly and thus safe operation is not secured.

Considering the test setup having a track of 18 meters, the drone camera does not provide sharp enough images to recognize objects that far. Based on some tests, humans are detected better than any objects but even then the maximum distance is at around 5 meters with the grayscale camera with the current AI model. Better camera, such as an RGB camera (available for Crazyflie) could produce better performance. Also, a bigger AI model might perform better in recognizing humans from bad quality images.

Idealised scenario of Crazyflie being able to position itself into 45° angle in the ≈ 50 ms and maintaining it throughout the braking process is optimistic. However, during the braking there also exists the y component which makes the drone fly up when braking. Safety-wise this is good feature as this most likely means that the drone will fly over the person to be avoided even if braking is not sufficient. Also, the actuators are considered ideal so the real-life performance would be worse.

Latency-wise an interesting finding is the big difference in maximum latencies of sending a raw image and JPEG image. The JPEG sending times were reaching over triple the time of sending raw images. This was thought to be an outlier in tests so a retake was performed on the tests but it was still reaching maximum values counted in seconds. This is most likely explained by congestion in 2.4 GHz frequencies as the test was done in university premises in daytime. However, raw images seem to have a steady rate of maximum latencies in all distances. One other explanation could be that the ESP32 goes on a power saving mode during the encoding process of the JPEG image and it varies a lot how fast the ESP32 is actually waking up. With raw images ESP32 was almost constantly transferring data.

7. CONCLUSION

During this thesis, physical safety was considered on an edge offloading scenario where a small edge device offloads a part of its control decisions to a remote edge server. First, Chapter 2 introduced key safety standards, regulations and known risks with drones. Then, Chapter 3 introduced edge computing principles. After this, the test environment was described more in detail. An example edge offloading scenario was described, which was recreated as a test scenario in this thesis. The test equipment was explained: the Crazyflie test edge device, the used edge server, the *PoCL-R* edge computing stack and how these three are linked together in this thesis. After this, applying regulations and safety standards started. As ISO 26262 played a big part in this thesis, a whole chapter was dedicated to it in Chapter 5. This chapter defined the "pedestrian detection system" which uses edge offloading to send part of Crazyflie's control decisions on a remote server. This scenario is not without its problems, so the chapter ends up in defining safety goals related to edge offloading when control decisions are offloaded. Finally, Chapter 6 showed how the safety goals were taken into account in this test scenario.

Chapter 2 answered to research question #2 by introducing the EU drone regulations 2019/947 and 2019/945 and the automotive standards ISO 26262 and ISO 19237. These were then used as a basis for physical safety improvements throughout the rest of this thesis. Regarding the regulations and standards, the EU drone regulations are directly related to the test scenario due to using a drone as the edge device in the test scenario. However, the automotive standards were applied as seen fit due to them being not directly related to drones. These standards still provided a good framework to consider different aspects of safety in the test scenario. As more research comes into drones and their use cases in every day life, safety standards for drones will most likely be formed. But until then, other approaches need to be considered and that's why established standards, like these automotive standards, were used in this thesis.

Chapter 6 considered research question #1 by introducing safety features that are required to safely offload control decisions. The chapter also introduces some numerical values in terms of latencies and Crazyflie specific flight speeds. In terms of safety, the variance in latencies is a problematic issue especially on Crazyflie. In terms of real-time control, using raw images provides more predictable performance but this comes at the cost of slower execution times.

As far as the feasibility of edge offloading control decisions goes, the test scenario produced very positive results. In general, edge offloading allows a larger variety of use cases due to increased computational capabilities, but the drawback is that when the connection is bad or completely lost, the edge device may not fulfill its functionality requirements. Thus, a local fallback will always be required if the offloaded control decisions have a potential for causing physical damage.

Considering the test scenario in this thesis, edge offloading allowed a completely new functionality for Crazyflie, namely flying independently (with an observer always present). Section 6.3.1 showed some latency metrics that can be considered challenging in terms of what can be offloaded. However, the bad latency metrics come down more to the used edge device (Crazyflie) rather than the performance of the software stack or the nature of offloaded parts. Omitting the edge device's inherent problems, the test scenario bolstered fundamental problems such as complete loss of connection. These problems help to consider what types of control decisions are actually safe to offload.

Future work could include the implementation of AI object detection locally alongside the possibility to use remote execution. This requires structural changes to the code but also updates to the software tools which is not controllable by the application logic software developer. Considering the software tools available now, the current local fallback implementation could be made better by using more robust and performant image processing algorithms. This will also require structural changes to the code to include better usage of external memory such as HyperRAM but it won't require relying on updates to the software development tools.

In order to further demonstrate the edge software stack capabilities it would be beneficial to make the drone fly in 3 dimensions. The previously mentioned advancements to the local fallback could be one step towards 3 dimensional movement but there still exists problems regarding the amount of sensor data available in Crazyflie. Crazyflie has no sensors to detect the distances to objects detected by the AI so a new test drone might be worth considering if 3 dimensional movement is wanted. If 3 dimensional movement is to be done in Crazyflie, there exists methods to estimate distances in an image by using e.g. a simple pinhole model. However, this requires a controlled test environment where there exists an object with a size known beforehand. This also introduces new physical safety issues such as the stability of the drone when the drone slows down and how well the drone maintains a safe distance to the objects it follows.

REFERENCES

- [1] European Union. “Commission Implementing Regulation (EU) 2019/947 of 24 May 2019 on the rules and procedures for the operation of unmanned aircraft”. In: *Official Journal of the European Union* L 152 (June 11, 2019), pp. 45–71. URL: http://data.europa.eu/eli/reg_impl/2019/947/oj.
- [2] European Union. “Commission Delegated Regulation (EU) 2019/945 of 12 March 2019 on unmanned aircraft systems and on third-country operators of unmanned aircraft systems”. In: *Official Journal of the European Union* L 152 (June 11, 2019), pp. 1–40. URL: http://data.europa.eu/eli/reg_del/2019/945/oj.
- [3] Technical Committee ISO/TC 22, Subcommittee SC 32. *ISO 26262-10:2018 Road vehicles - Functional safety - Part 10: Guidelines on ISO 26262*. 2nd ed. Standard. International Organization for Standardization, ISO. Dec. 2018.
- [4] Technical Committee ISO/TC 22, Subcommittee SC 32. *ISO 26262-9:2018 Road vehicles - Functional safety - Part 1: Vocabulary*. 2nd ed. Standard. International Organization for Standardization, ISO. Dec. 2018.
- [5] Technical Committee ISO/TC 22, Subcommittee SC 32. *ISO 26262-3:2018 Road vehicles - Functional safety - Part 3: Concept phase*. 2nd ed. Standard. International Organization for Standardization, ISO. Dec. 2018.
- [6] Technical Committee ISO/TC 204. *ISO 19237:2017 Intelligent transport systems — Pedestrian detection and collision mitigation systems (PDCMS) — Performance requirements and test procedures*. 1st ed. Standard. International Organization for Standardization, ISO. Dec. 2017.
- [7] The European Union Aviation Safety Agency (EASA). *Flying in your country - National Aviation Authorities*. May 2023. URL: <https://www.easa.europa.eu/en/domains/civil-drones/naa> (Accessed 4.10.2024).
- [8] The Finnish Transport and Communications Agency Traficom. *To fly as safely as possible – Air Safety*. Apr. 2023. URL: <https://www.droneinfo.fi/en/study-material/fly-safely-possible-air-safety> (Accessed 5.8.2024).
- [9] Liu, Yaqiong et al. “Toward Edge Intelligence: Multiaccess Edge Computing for 5G and Internet of Things”. In: *IEEE Internet of Things Journal* 7.8 (2020), pp. 6722–6747. DOI: 10.1109/JIOT.2020.3004500.
- [10] Shi, Bowen et al. “Offloading Guidelines for Augmented Reality Applications on Wearable Devices”. In: *MM '15*. Brisbane, Australia: Association for Computing Machinery, 2015, pp. 1271–1274. ISBN: 9781450334594. DOI: 10.1145/2733373.2806402. URL: <https://doi.org/10.1145/2733373.2806402>.

- [11] Uitto, Jyry. *Offloading computation with a minimized OpenCL runtime from a nano drone*. Faculty of Information Technology and Communication Sciences, Tampere University. 2022.
- [12] PoCL developers. *Portable Computing Language*. 2023. URL: <https://portablecl.org/> (Accessed 9.7.2024).
- [13] The Khronos Group Inc. *OpenCL*. 2024. URL: <https://www.khronos.org/api/openccl> (Accessed 19.3.2024).
- [14] Solanti, Jan et al. "PoCL-R: A Scalable Low Latency Distributed OpenCL Runtime". In: *Embedded Computer Systems: Architectures, Modeling, and Simulation*. 2022, pp. 78–94. ISBN: 978-3-031-04580-6.
- [15] Bitcraze AB. *Crazyflie Firmware*. GitHub Repository. GitHub. June 2024. URL: <https://github.com/bitcraze/crazyflie-firmware> (Accessed 28.6.2024).
- [16] Bitcraze AB. *Crazyflie 2.1*. 2023. URL: <https://www.bitcraze.io/products/crazyflie-2-1/> (Accessed 19.3.2024).
- [17] Bitcraze AB. *AI deck 1.1*. 2023. URL: <https://www.bitcraze.io/products/ai-deck/> (Accessed 1.7.2024).
- [18] GreenWaves Technologies. *GAP8 Hardware Reference Manual*. Version 1.5.5. 2019. URL: <http://www.greenwaves-technologies.com/gap8-datasheet> (Accessed 1.7.2024).
- [19] Bitcraze AB. *Flow deck V2*. 2023. URL: <https://www.bitcraze.io/products/flow-deck-v2/> (Accessed 1.7.2024).
- [20] Bitcraze AB. *Getting started with the Flow deck*. 2023. URL: <https://www.bitcraze.io/documentation/tutorials/getting-started-with-flow-deck/#measurement-details> (Accessed 4.7.2024).
- [21] Bitcraze AB. *4 x 7 mm DC-motor pack for Crazyflie 2.X*. 2024. URL: <https://store.bitcraze.io/collections/spare-parts-crazyflie-2-0/products/4-x-7-mm-dc-motor-pack-for-crazyflie-2> (Accessed 5.8.2024).
- [22] Brandt, J.B. et al. *UIUC Propeller Database, Vol 2*. University of Illinois at Urbana-Champaign, Department of Aerospace Engineering. May 2015. URL: <https://m-selig.ae.illinois.edu/props/volume-2/propDB-volume-2.html#Crazyflie> (Accessed 6.8.2024).
- [23] Wesstein, Eric W. *Least Squares Fitting–Logarithmic*. Wolfram MathWorld. July 2024. URL: <https://mathworld.wolfram.com/LeastSquaresFittingLogarithmic.html> (Accessed 7.8.2024).
- [24] Z. S. Spakovszky. *Performance of Propellers - Thrust Coefficient*. 2007. URL: <https://web.mit.edu/16.unified/www/FALL/thermodynamics/notes/node86.html#SECTION06374100000000000000> (Accessed 7.8.2024).
- [25] Khronos® OpenCL Working Group. *The OpenCL C Programming Language*. Apr. 2024. URL: https://registry.khronos.org/OpenCL/specs/3.0-unified/html/OpenCL_C.html#the-openccl-c-programming-language (Accessed 12.10.2024).

- [26] Bitcraze AB. *cflib: Crazyflie python library*. GitHub Repository. GitHub. Oct. 2024. URL: <https://github.com/bitcraze/crazyflie-lib-python> (Accessed 12.10.2024).
- [27] Microsoft. *ONNX Runtime*. 2024. URL: <https://onnxruntime.ai/> (Accessed 12.10.2024).
- [28] OpenCV team. *About*. 2024. URL: <https://opencv.org/about/> (Accessed 12.10.2024).
- [29] Ultralytics. *Ultralytics v8.2.0 Release Notes*. GitHub Repository. GitHub. Apr. 2024. URL: <https://github.com/ultralytics/assets/releases/tag/v8.2.0> (Accessed 26.7.2024).
- [30] Bitcraze AB. *flowdeck_v1v2.c*. GitHub Repository. GitHub. Mar. 2021. URL: https://github.com/bitcraze/crazyflie-firmware/blob/master/src/deck/drivers/src/flowdeck_v1v2.c (Accessed 4.7.2024).
- [31] Bitcraze AB. *v153l1x.c*. GitHub Repository. GitHub. Dec. 2023. URL: <https://github.com/bitcraze/crazyflie-firmware/blob/master/src/drivers/src/v153l1x.c> (Accessed 4.7.2024).
- [32] Bitcraze AB. *pmw3901.c*. GitHub Repository. GitHub. Mar. 2023. URL: <https://github.com/bitcraze/crazyflie-firmware/blob/master/src/drivers/src/pmw3901.c> (Accessed 4.7.2024).
- [33] Bitcraze AB. *Syslink protocol*. Feb. 2024. URL: <https://www.bitcraze.io/documentation/repository/crazyflie2-nrf-firmware/master/protocols/syslink/> (Accessed 4.7.2024).
- [34] Bitcraze AB. *CPX - Crazyflie Packet eXchange*. Nov. 2023. URL: <https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/cpx/> (Accessed 4.7.2024).
- [35] Bitcraze AB. *AI-deck*. Aug. 2019. URL: https://www.bitcraze.io/documentation/hardware/ai_deck_1_1/ai-deck-revc.pdf (Accessed 4.7.2024).
- [36] Himax Technologies Inc. *HM01B0 Ultralow Power CIS*. 2023. URL: <https://www.himax.com.tw/products/cmos-image-sensor/always-on-vision-sensors/hm01b0/> (Accessed 4.7.2024).
- [37] European Union. "Decision No 768/2008/EC of the European Parliament and of the Council of 9 July 2008 on a common framework for the marketing of products, and repealing Council Decision 93/465/EEC". In: *Official Journal of the European Union* L 218 (Aug. 13, 2008), pp. 82–128. URL: [http://data.europa.eu/eli/dec/2008/768\(1\)/oj](http://data.europa.eu/eli/dec/2008/768(1)/oj).
- [38] European Union Aviation Safety Agency. *FAQ n.116454 - Do I need to register my drone?* Oct. 2020. URL: <https://www.easa.europa.eu/en/faq/116454> (Accessed 1.8.2024).
- [39] The Finnish Transport and Communications Agency Traficom. *Dronelennättäjät*. 2022. URL: <https://www.droneinfo.fi/fi/dronelennattajat> (Accessed 1.8.2024).
- [40] Vina, Abirami. *Ultralytics YOLOv8 for Speed Estimation in Computer Vision Projects*. Ultralytics. May 2024. URL: <https://www.ultralytics.com/blog/ultralytics-yolov8-for-speed-estimation-in-computer-vision-projects> (Accessed 25.9.2024).

- [41] Technical Committee ISO/TC 22, Subcommittee SC 32. *ISO 26262-8:2018 Road vehicles - Functional safety - Part 8: Supporting processes*. 2nd ed. Standard. International Organization for Standardization, ISO. Dec. 2018.
- [42] Github user "gemenerik". *Classification example fails to construct CNN when flashed with JTAG, issue #143*. Mar. 2024. URL: <https://github.com/bitcraze/aideck-gap8-examples/issues/143> (Accessed 13.10.2024).
- [43] Github user "knmcguire". *CFloader flash img hangs at 99% with the classification demo, issue #104*. 2023. URL: <https://github.com/bitcraze/aideck-gap8-examples/issues/104> (Accessed 27.9.2024).

APPENDIX A: PYTHON CODE

The Python code used to produce the results in Section 4.3.4 and 6.3 is shown here. The python packages matplotlib and numpy are required as well as a package to show plots. For this purpose, PyQt6 was also installed. The plot can be seen by removing the commented out "plt.show()" in the function "actuator_capability()".

```
import matplotlib.pyplot as plt
import numpy as np
import math

# t_r = reaction time until braking starts
# v_end = speed at 18m mark
# a = (de)acceleration, +/- must be included in value
def initial_velocity(t_r, v_end, a):
    S_T = 18

    # Second degree polynomial
    a_poly = -1 / (2*a)
    b_poly = t_r
    c_poly = (v_end**2 / (2*a)) - S_T
    v0_plus = (-b_poly + math.sqrt(b_poly**2 - 4*a_poly*c_poly) ) \
              / (2*a_poly)
    # v0_nega = (-b_poly - math.sqrt(b_poly**2 - 4*a_poly*c_poly) ) \
    #          / (2*a_poly)

    # print(f"Initial speed (+): {v0_plus} m/s")
    # print(f"Initial speed (-): {v0_nega} m/s")

    v0 = v0_plus
    # print(f"----> Chosen: {v0}")

    s_r = v0 * t_r
    s_b = S_T - s_r
```

```

# print("Distances:")
# print("- React: {:.3f} m".format(s_r))
# print("- Brake: {:.3f} m".format(s_b))
# print("- Total: {:.3f} m".format(s_r + s_b))
# print("Test pass if initial speed < {:.3f}".format(v0))

print("v_end: {:.2f} m/s, t_r: {:.2f} s, v0: {:.2f} m/s" \
      .format(v_end, t_r, v0))

def actuator_capability():
    # x-term in (x,y)
    rpm = [4026.667, 5046.667, 5933.333, 7146.667, 7893.333,
           9166.667, 9926.667, 11086.667, 12080.000, 12940.000,
           13960.000, 15033.333, 16013.333, 16986.667, 17980.000,
           18973.333, 19860.000]

    # y-term in (x,y)
    ct = [0.139894, 0.149527, 0.147475, 0.150139, 0.147117,
          0.156218, 0.155704, 0.157201, 0.154470, 0.159684,
          0.158055, 0.159153, 0.158740, 0.162528, 0.163401,
          0.163912, 0.164534]

    N = len(rpm)
    print(f"Amount of elements: {N}")

    rpm_avg = sum(rpm) / N
    ct_avg = sum(ct) / N

    print(f"Averages:\n- RPM: {rpm_avg}\n- C_T: {ct_avg}")

    POINT_EXTRAPOLATION = 58800

    # 1)
    # Linear regression with least squares: a straight line
    # - Function of form  $y = m*x + b$ 

    sum_numerator = 0
    sum_denominator = 0

```

```

for i in range(N):
    sum_numerator += (rpm[i]-rpm_avg) * (ct[i]-ct_avg)
    sum_denominator += (rpm[i]-rpm_avg)**2

m = sum_numerator / sum_denominator
b = ct_avg - m * rpm_avg
y_line = m * POINT_EXTRAPOLATION + b

print("Straight line fitting:")
print(f"- m: {m}")
print(f"- b: {b}")
print(f"- y_line(58800) = {y_line}")

# Points of straight line fitting
points_x = np.arange(1000, 61000, 1000)
line_fit_y = points_x * m + b

# 2)
# Linear regression with least squares: a logarithmic line
# - Function of form  $y = m \cdot \ln(x) + b$ 

sum_yi_lnx = 0
sum_yi = 0
sum_lnx = 0
sum_lnx_squared = 0
squared_sum_lnx = 0

# np.log() is base e (= ln)
for i in range(N):
    sum_yi_lnx += ct[i] * np.log(rpm[i])
    sum_yi += ct[i]
    sum_lnx += np.log(rpm[i])
    sum_lnx_squared += np.log(rpm[i])**2

squared_sum_lnx = sum_lnx**2

m = (N * sum_yi_lnx - sum_yi * sum_lnx) / \
    (N * sum_lnx_squared - squared_sum_lnx)
b = (sum_yi - m * sum_lnx) / N

```

```

y_log = m * np.log(POINT_EXTRAPOLATION) + b

print("Logarithmic fitting:")
print(f"- m: {m}")
print(f"- b: {b}")
print(f"- y_log(58800) = {y_log}")

# Points of logarithmic function fitting
log_fit_y = m * np.log(points_x) + b

# Thrust
T1 = y_line * 1.2041 * 980**2 * (0.046)**4
T2 = y_log * 1.2041 * 980**2 * (0.046)**4
print(f"Thrust (y_line): {T1}")
print(f"Thrust (y_log): {T2}")

# Plotting
plt.plot(rpm, ct, "o", color="r", label="Measurement")
plt.scatter([POINT_EXTRAPOLATION] * 2, [y_line, y_log], c="grey", \
            edgecolors="black", label="Extrapolation")
plt.plot(points_x, line_fit_y, color="black", \
         label="Straight line fitting")
plt.plot(points_x, log_fit_y, color="blue", \
         label="Logarithmic fitting")
plt.title("Crazyflie pusher 1 propeller thrust" \
         " coefficient extrapolation", pad=15)
plt.xlabel("RPM (Revolutions per minute)", labelpad=10)
plt.ylabel("$\mathregular{C_T}$ (Thrust coefficient)", labelpad=10)
plt.legend()
# plt.show()
return T1

# -----

def main():
    # Actuator capability calculations + plotting
    T1 = actuator_capability()

```

```

print(f"T1: {T1}")

# Crazyflie initial speed calculations
print("\n----- Breaking estimation -----")

F_brake = math.cos(45 * math.pi / 180) * 4 * T1
a = -F_brake / 0.036
print(f"Thrust (x-axis): {F_brake} N")
print(f"(De)acceleration: {a} m/s^2")

# Distance until breaking actually happens:
# 1) send image
# 2) receive results
# 3) initiate braking (moving, assumed constant)
T_MOVING = 0.0547

# - Case 1) Full stop at 18 meter mark
# - Case 2) Speed 10 km/h at 18 meter mark
print("\n - 1) Full stop at 18 m mark\n" \
      " - 2) Speed 10 km/h at 18 m mark")

v_end = 0

# t_r = t_send + t_rcv + t_moving

# 1.1) min values, raw image
t_send = 539 / 1000
t_rcv = 13.8 / 1000
v_end = 0
print("\nMin + raw:")
initial_velocity(t_r=(t_send+t_rcv+T_MOVING), v_end=v_end, a=a)
v_end = 10/3.6
initial_velocity(t_r=(t_send+t_rcv+T_MOVING), v_end=v_end, a=a)

# 1.2) min values, JPEG image
t_send = 143 / 1000
t_rcv = 13.8 / 1000
v_end = 0
print("\nMin + JPEG:")
initial_velocity(t_r=(t_send+t_rcv+T_MOVING), v_end=v_end, a=a)

```

```
v_end = 10/3.6
initial_velocity(t_r=(t_send+t_rcv+T_MOVING), v_end=v_end, a=a)

# 2.1) avg values, raw image
t_send = 628 / 1000
t_rcv = 25.0 / 1000
v_end = 0
print("\nAvg + raw:")
initial_velocity(t_r=(t_send+t_rcv+T_MOVING), v_end=v_end, a=a)
v_end = 10/3.6
initial_velocity(t_r=(t_send+t_rcv+T_MOVING), v_end=v_end, a=a)

# 2.2) avg values, JPEG image
t_send = 203 / 1000
t_rcv = 25.0 / 1000
v_end = 0
print("\nAvg + JPEG:")
initial_velocity(t_r=(t_send+t_rcv+T_MOVING), v_end=v_end, a=a)
v_end = 10/3.6
initial_velocity(t_r=(t_send+t_rcv+T_MOVING), v_end=v_end, a=a)

# 3.1) max values, raw image
t_send = 869 / 1000
t_rcv = 684 / 1000
v_end = 0
print("\nMax + raw:")
initial_velocity(t_r=(t_send+t_rcv+T_MOVING), v_end=v_end, a=a)
v_end = 10/3.6
initial_velocity(t_r=(t_send+t_rcv+T_MOVING), v_end=v_end, a=a)

# 3.2) max values, JPEG image
t_send = 2794 / 1000
t_rcv = 684 / 1000
v_end = 0
print("\nMax + JPEG:")
initial_velocity(t_r=(t_send+t_rcv+T_MOVING), v_end=v_end, a=a)
v_end = 10/3.6
initial_velocity(t_r=(t_send+t_rcv+T_MOVING), v_end=v_end, a=a)

exit(0)
```

main()

The code above produces the following output:

Amount of elements: 17

Averages:

- RPM: 12003.137294117645
- C_T: 0.15575011764705882

Straight line fitting:

- m: 1.2791210212581925e-06
- b: 0.14039665241310476
- y_line(58800) = 0.21560896846308647

Logarithmic fitting:

- m: 0.013547381362769156
- b: 0.02984036296173307
- y_log(58800) = 0.178616311521198

Thrust (y_line): 1.1163820791058483

Thrust (y_log): 0.9248411633321835

T1: 1.1163820791058483

----- Breaking estimation -----

Thrust (x-axis): 3.1576053541235285 N

(De)acceleration: -87.71125983676468 m/s²

- 1) Full stop at 18 m mark
- 2) Speed 10 km/h at 18 m mark

Min + raw:

v_end: 0.00 m/s, t_r: 0.61 s, v0: 24.15 m/s

v_end: 2.78 m/s, t_r: 0.61 s, v0: 24.20 m/s

Min + JPEG:

v_end: 0.00 m/s, t_r: 0.21 s, v0: 40.62 m/s

v_end: 2.78 m/s, t_r: 0.21 s, v0: 40.69 m/s

Avg + raw:

v_end: 0.00 m/s, t_r: 0.71 s, v0: 21.66 m/s

v_end: 2.78 m/s, t_r: 0.71 s, v0: 21.70 m/s

Avg + JPEG:

v_end: 0.00 m/s, t_r: 0.28 s, v0: 36.62 m/s

v_end: 2.78 m/s, t_r: 0.28 s, v0: 36.69 m/s

Max + raw:

v_end: 0.00 m/s, t_r: 1.61 s, v0: 10.78 m/s

v_end: 2.78 m/s, t_r: 1.61 s, v0: 10.81 m/s

Max + JPEG:

v_end: 0.00 m/s, t_r: 3.53 s, v0: 5.05 m/s

v_end: 2.78 m/s, t_r: 3.53 s, v0: 5.07 m/s