ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

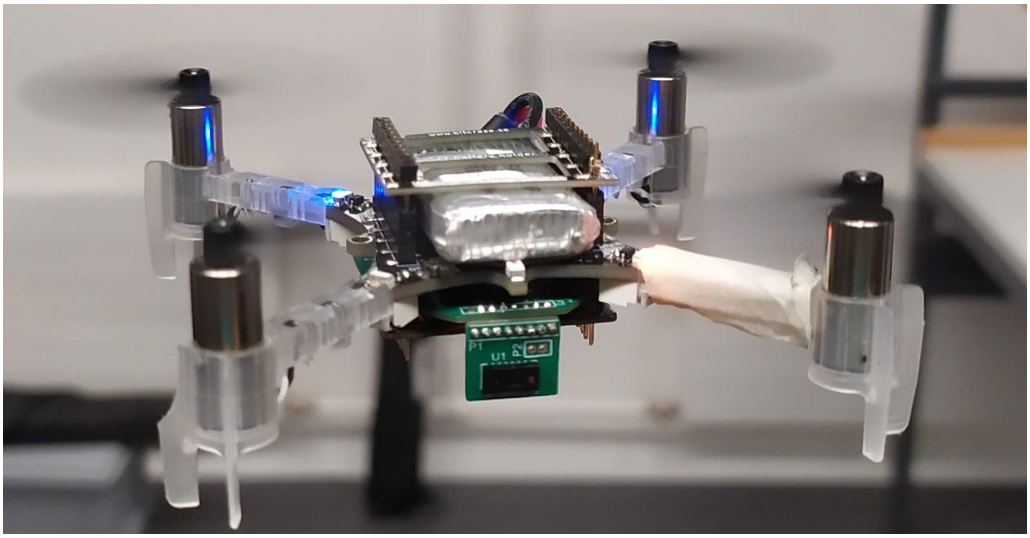
CENTER FOR PROJECT-BASED LEARNING

Department of Information Technology and Electrical Engineering

Autumn Semester 2022

Nano-Drones: Enabling Indoor Collision Avoidance With a Miniaturized Multi-Zone Time of Flight Sensor

Master Thesis



Iman Ostovar
iostovar@student.ethz.ch

March 2022

Supervisors: Dr. Tommaso Polonelli, tommaso.polonelli@pbl.ee.ethz.ch
Vlad Niculescu, vladn@iis.ee.ethz.ch
Hanna Müller, hanmuell@iis.ee.ethz.ch
Dr. Michele Magno, michele.magno@pbl.ee.ethz.ch

Professor: Prof. Dr. L. Benini, lbenini@iis.ee.ethz.ch

Acknowledgments

At this point, I would like to express a special thanks to my supervisors, Dr. Tommaso Polonelli, Vlad Niculescu, Hanna Müller, Dr. Michele Magno, who, even in times of the pandemic and social distancing, accompanied and supported me in weekly online and in-person meetings with informative inputs and understandable answers to all my questions. I also want to thank Prof. Dr. Luca Benini for working on my thesis at his department. In addition, thanks to STMicroelectronics for the support provided during the development of this work. Moreover, this work was partially supported by Politecnico di Torino outgoing mobility program and EDISU international mobility grant. At last, thanks to Prof. Dr. Ernesto Sanchez for his guidance and support.

Abstract

Unmanned aerial vehicles (UAVs) are active research topics, especially the nano and micro subclass. They are centimeter-size drones with minimal onboard computational capabilities. These light-weights platforms provide good agility and movement freedom in indoor environments(i.e. GPS-denied environments). However, it is still a significant challenge due to the limited onboard computational capabilities to enable autonomous navigation (or even primary obstacle avoidance abilities) using standard image sensors. Vision-based perception algorithms used routinely on standard-size drones prevent their use with state-of-the-art nano-UAVs.
This work demonstrates the possibility of using a new multi-zone Time of Flight (ToF) sensor to enhance autonomous navigation with a significantly lower computational load rather than most common vision-based solutions. In particular, the novel integrated ToF sensor is characterized for the first time in literature in-field using an *ad hoc* light-weight PCB and the Crazyflie nano-UAV.

Furthermore, an optimized approach to calculate collision probability from each ToF sensor frame has been proposed relying on empirical data. The main goal is to develop a new methodology capable of obstacle avoidance fusing only one 8*8 ToF on the front. The final system proved reliable (>95%) in-field obstacle avoidance capabilities when flying in indoor environments with dynamic obstacles.

Declaration of Originality

I, at this moment, confirm that I am the sole author of the written work enclosed here and compiled it in my own words. Parts excepted are corrections of form and content by the supervisor. For a detailed version of the declaration of originality, please refer to Appendix B

Iman Ostovar,
Zurich, March 2022

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nano and micro quadcopters, characterized by an overall payload of a few tens of grams, are becoming particularly interesting in many fields, including search and rescue, aerial inspection, first aid, and indoor surveillance [1]. However, applied research on autonomous nano-drones has revealed great challenging problems. Miniaturized hardware leads to extremely low onboard computational resources, making it challenging to enable complex navigation missions and more basic tasks, e.g., obstacle avoidance [2]. Considering the recent advancements of embedded microcontrollers, such as AI-focused mW processors, now it is possible to deploy real-time image processing algorithms even onboard of the nano-drones. However, the amount of random access memory (RAM) usually available onboard is insufficient to keep high-resolution images [3]. Even the latest [4] research on insect-sized robots can only stream images to a local gateway without any onboard processing. As a result, common vision-based navigation using cameras still shows limitations when exploring unknown environments or at flight speeds higher than 0.5 m/ s [5]. Compared to standard image sensors, optical distance sensors extract the context-depth of the frontal view, operating at hundreds of Hz and with reduced computational complexity [6]. This work presents an obstacle avoidance system designed for nano-drones. Our demonstrator is based on the Crazyflie platform from Bitcraze [1]. Furthermore, it exploits a novel, miniaturized, and light-weight 64-pixel time-of-flight (ToF) sensor from STMicroelectronics (i.e., VL53L5CX).
The in-field demo proves the potential of our work to enable safe collision avoidance and path-planning at a negligible computational demand compared to vision-based approaches.

[1] www.bitcraze.io

1.1. Contribution

The main contribution is adding indoor navigation capability to nano-UAVs using a front-facing ST ToF sensor.

At first, this novel sensor was deployed on a Crazyflie platform.

Secondly, as no previous data was available regarding multi-zone ToF sensors on nano-drones, a data-logger has been implemented to collect valuable data. Later, this data would be used to calibrate and characterize the sensor.

Thirdly, after preliminary tests, a light-weight solution has been proposed to fuse sensor data for obstacle avoidance and indoor navigation.
The work is structured into three phases; refer to Appendix A for the detailed task description.

Chapter 2

Related Work

Recent works presented alternative solutions to the typical vision-based perception approach in the way of autonomous nano-drones.

- PULP DroNet single-camera CNN: In this approach, the drone uses a new parallel ultra-low-power (PULP) system-on-chip (SoC). It uses a camera and CNN to process images. The final result shows it is unreliable, and the behavior is data-set dependent [5].
- Camera with radar detector: In this approach, another radar will be added to the camera for indoor purposes; however, fusing radar data increases complexity. Moreover, blur obstacles due to dust or low light increase error probability [7].
- LASER ranger: Using LASERs for obstacle avoidance is a needed reinforcement learning methodology. This approach is unreliable, especially in high-density obstacle environments [8].
- Ultra wide band (UWB): To use this approach, it is necessary to have an environment map in advance, to be aware of the obstacles and their size. In the case of other drones with UWB, the number of drones would be limited due to increasing latency [9].
- Millimeter waves: This methodology enables the drone to measure obstacle velocity and distance while light condition has a minimal effect on the measurements. Moreover, it works in long ranges. However, it can not give the outline of the obstacle; besides, there is an accuracy issue with relatively stationary targets [10] [11].
- Single point ToF: Using this approach can give distance to the object, but it is impossible to understand context as it only gives the distance to the single point obstacle.

However, many challenges have remained to achieve a reliable methodology.

Chapter 3

Background

3.1. ToF Sensor

A time-of-flight camera (ToF camera) is a range imaging camera system deploying time-of-flight techniques to determine the distance between the camera and the object. This procedure would be done for each point of the image. The distance would be figured out by measuring the round trip time of an artificial light signal provided by a LASER or an LED. The procedure is described at Fig. 3.1.

3.1.1. ST ToF Sensor

"STMicroelectronics's 4th generation of FlightSense™ sensors offers a multi-zone ranging sensor to create a 64-zone mini depth map up to 4 m. ST ToF sensors are an all-in-one (emitter, receiver, and processor) system for an easy, cost-effective, and small footprint integration" . [1]

- Accurate and high-speed
- All-in-one
- Low power
- Using 940nm wave
- Measure distance independent to the targets surface features

[1] https://www.st.com/en/imaging-and-photonics-solutions/time-of-flight-sensors.html

Figure 3.1.: ToF of a light pulse reflecting off an obstacle, the distance would be the result of multiplication of the light speed and round trip time of the signal.

The VL53L5CX[2] is a ToF multi-zone ranging sensor produced by STMicroelectronics. It is possible to achieve a millimeter accuracy in various ambient lighting conditions and a wide range of cover glass materials. The most important feature of the VL53L5CX is the multi-zone capability, which can be configured as an 8x8 (max 15 Hz) or 4x4 (max 60 Hz) matrix with the field of view (FoV) of 61°; however, the real FoV can slightly change due to different environmental conditions and sensor configurations.

An error flag is reported in case of ToF miss-calculation or interference at 940 nm light-wave. In this way, noise and errors can be easily filtered out.

3.2. Crazyflie Platform

This project was done on Crazyflie open hardware/software platform Fig. 3.3a. The drone itself is small and weighs about 27 grams. There is also an additional payload that weighs 15 grams. It has many features that makes it suitable for this project. It has low latency, long-range radio, and low Bluetooth energy. The main application microcontroller unit is from STMicroelectronics, STM32F405. The platform supports an expansion deck and has real-time logging capability. The drone's power is 7.6 W without considering the additional payload.

The drone would be equipped with ToF deck as shown in Fig. 3.4.

CrazyFlie software programming overview:

- Python for the Crazyflie PC API and client
- C for the Crazyflie firmware

[2] www.st.com/en/imaging-and-photonics-solutions/vl53l5cx.html

Figure 3.2.: ToF sensor 8x8 multi-zone ranging sensor with a wide field of view

3.2.1. Crazyflie Libraries

3.2.1.1. Crazyflie Firmware

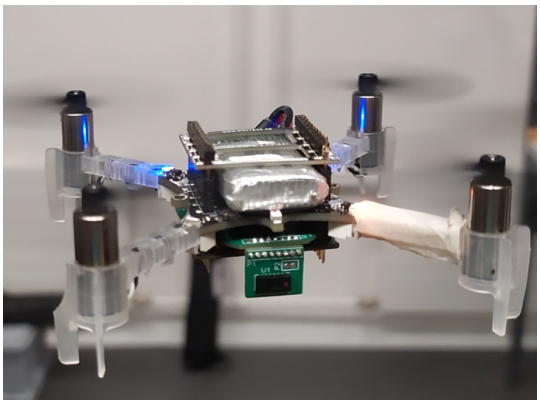
Project [3] contains the source code for the firmware used in the Crazyflie range of platforms, including the Crazyflie 2.X [4].

3.2.1.2. Crazyflie Clients Python

Using the Crazyflie PC client [5], makes it possible to flash and control the Crazyflie. It implements the user interface and high-level control. This project is mainly based on cflib Python library. The Crazyflie PC client Fig. 3.3a supports connecting one Crazyflie

[3] https://github.com/bitcraze/crazyflie-firmware
[4] GitHub commit used for this project: 0f36a866955074c7a267deec0d69399e8dcc11be
[5] https://www.bitcraze.io/2018/03/crazyflie-clients/

(a) Crazyflie 2.1 platform



(b) Hardware design of the VL53L5CX on the Crazyflie 2.1.

Figure 3.3.: Hardware platform

using the Crazyradio [6] dongle or direct USB connection to Crazyflie 2.X. It is capable of telemetry and parameter updating even during the flight. The Crazyflie PC client is using the *crazyflie-lib-python* to communicate with the Crazyflie.[7]

3.2.1.3. Crazyflie Python Library

cflib is an API written in Python used to communicate with the Crazyflie and Crazyflie 2.X quadcopters. It is intended to be used by client software to communicate with and control a Crazyflie quadcopter. For instance, our custom Crazyflie PC client uses the cflib.[8]

3.2.1.4. Crazyflie AI-deck Examples

From this library, only "wifi_jpeg_streamer" example is loaded to processor for stream images from camera to the custom client [9].

[6] https://www.bitcraze.io/products/crazyradio-pa/

[7] https://github.com/bitcraze/crazyflie-lib-python GitHub commit used for this project: bd904fa89df7c0813edd8fff8979a01a59f56455

[8] https://github.com/bitcraze/crazyflie-lib-python GitHub commit used for this project: 358de04be72c2cdfadfcc47589d1ff20e9629f03

[9] https://github.com/bitcraze/AIdeck_examples GitHub commit used for this project: 7d4c182fe9d2f8d9b75a063e8d824058b133a443

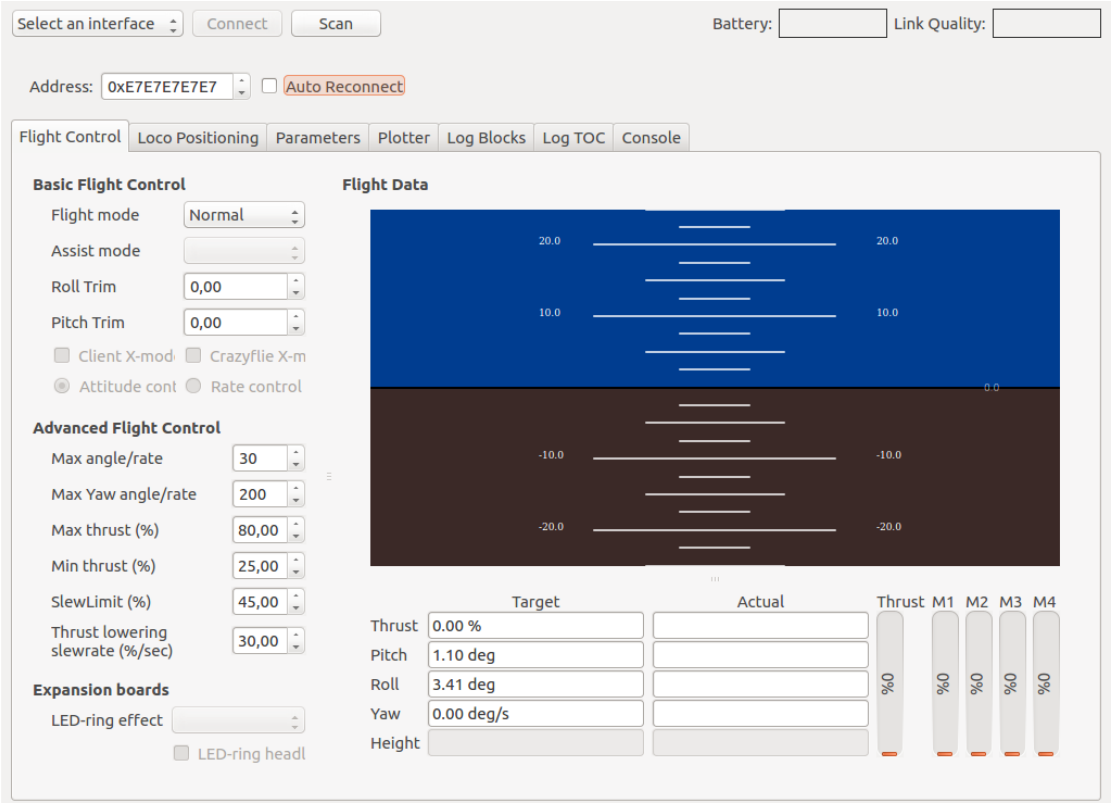


Figure 3.4.: CrazyFlie PC client

3.3. STM32 Cortex-M4

STM32F4 is a series of high-performance MCUs with DSP and FPU instructions. These processors are based on Arm® Cortex®-M4 and currently used the processor for this project, in particular, is: "STM32F405: 168 MHz CPU/210 DMIPS, up to 1 Mbyte of flash memory"

3.4. Visual Studio Code

Visual studio code is a source-code editor made by Microsoft. It is mainly used for editing/interpreting our Python and editing/compiling C projects.
Used version: 9.1.0

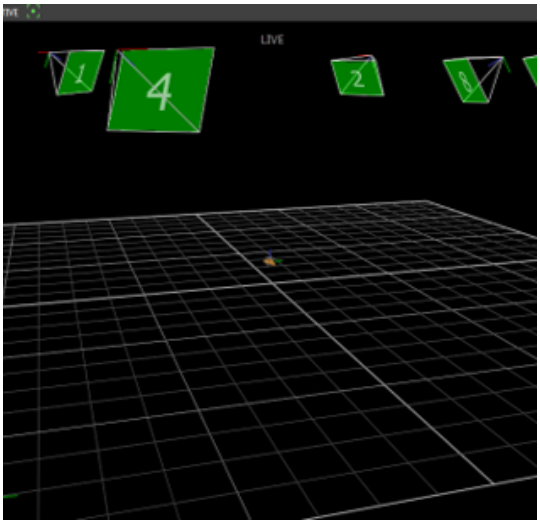
3.5. Python

Python is a high-level, general-purpose programming language. Here, the Python client side of our application is mainly based on cflib and other Python libraries.
Used version: Python 3.10.0

3.6. Vicon System

It is a camera-based solution for position and angle estimation. It uses multiple cameras pre-installed around the field. It uses the infrared camera for ranging and could reach very accurate (mm-range) distance measurements. It uses a very precise camera Fig. 3.5a. There is a room at ETH with eight Vicon cameras capable of positioning objects with very accurate precision Fig. 3.5b.

(a) Vicon camera



(b) Vicon room platform

Figure 3.5.: Vicon

Chapter 4

Hardware / Firmware / Algorithm Implementation

4.1. Hardware

4.1.1. Overview

The full system consists of:

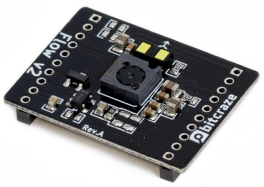
- Main CrazyFlie drone: Fig. 4.1a.
- AI-deck: To collect camera images Fig. 4.1b.
- Flow-deck: Necessary for flight control Fig. 4.1c.
- ToF-deck: Fig. 4.1d.

4.1.2. Requirements

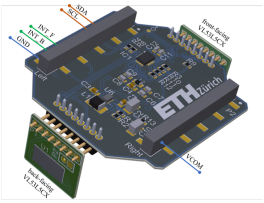
There are specific requirements the board has to fulfill. Since this board is an expansion deck for the Crazyflie, the interface to the drone and the connector placement have to be the same. Moreover, it should also be compatible with the other decks used, including flow-deck and AI-deck. The main board has to be small to fit onto the drone, and the height should not be too high to touch the ground. Otherwise, the deck is easily damaged during landing or crashes. In addition, all decks have to weigh less than 15 g for the drone to lift them.

(a) CrazyFlie drone

(b) AI-deck



(c) Flow-deck



(d) ToF-deck

Figure 4.1.: ToF development platform

4.1.3. ToF Deck

It is a custom design deck connected to the Crazyflie and supports two sensor boards on it. Its 3D platform is shown in Fig. 4.2.

4.1.3.1. ToF Deck Main Board

This board consists of four main parts. The first part is "power". The power section converts volatile li-ion battery voltage (3.0-4.2V) to a stable voltage of 3V using **TPS62233DRYR** DC to DC regulator. This voltage is used for all digital circuits.
The second part is the "voltage filter". VL53L5cx needs two voltages, one for the analog part and one for the digital. This filter separates these two powers. In this way, voltage noises do not decrease ToF's measurement accuracy. So, I put this filter to avoid the digital noise interference on the analog circuits.
The third part is "GPIO management". The Crazyflie has a limited number of free I/Os in its interconnection. Each ToF sensor needs two I/Os, one for control power mode and another for handling interrupts. Using TCA6408ARSVR IC, I solved this problem. It is an IC that is controlled by I2C and it handles some of GPIOs (inputs, outputs, and also input changes). Using only I2C, this is possible to manage output ports and read input pins' status. Moreover, It could sense changes in inputs and make interrupt. Consequently, the ToF deck only uses one GPIO pin to trigger microcontroller's interrupt, and GPIO expander IC would control other GPIOs. The board has some configuration 0 Ω resistors to change board configuration. The tested configuration among jumpers (0 Ω Resistors) R9-R11-R12-R2-R8 should not be soldered. The board also includes some LEDs connected to the GPIO expander for debugging purposes.

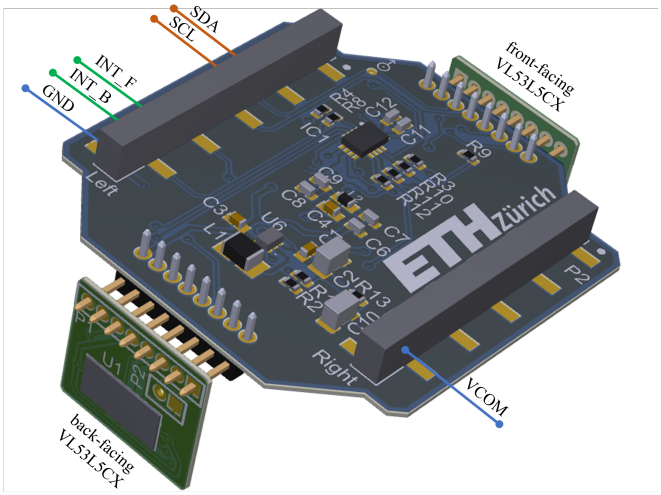


Figure 4.2.: The ToF deck, design and interconnection overview

The last part is "Pin Headers". There are four series of pin headers. Two of them are mapped to Crazyflie interconnection, connected to the Crazyflie board. From this interconnection, on the left side, I2C pins (SDA, SCL), I/O (pin N.7) and Ground (GND) are connected. Only VCOM, the battery power line, is connected on the right side. Other pins are left float. The other two pin headers are located in the front and back of the ToF deck, connecting the two sensor boards to the ToF deck board. It includes an I2C connection, analog and digital circuits power (both working with 3v), reset, low power control and interrupt pins. The final board is shown in Fig. 4.3.

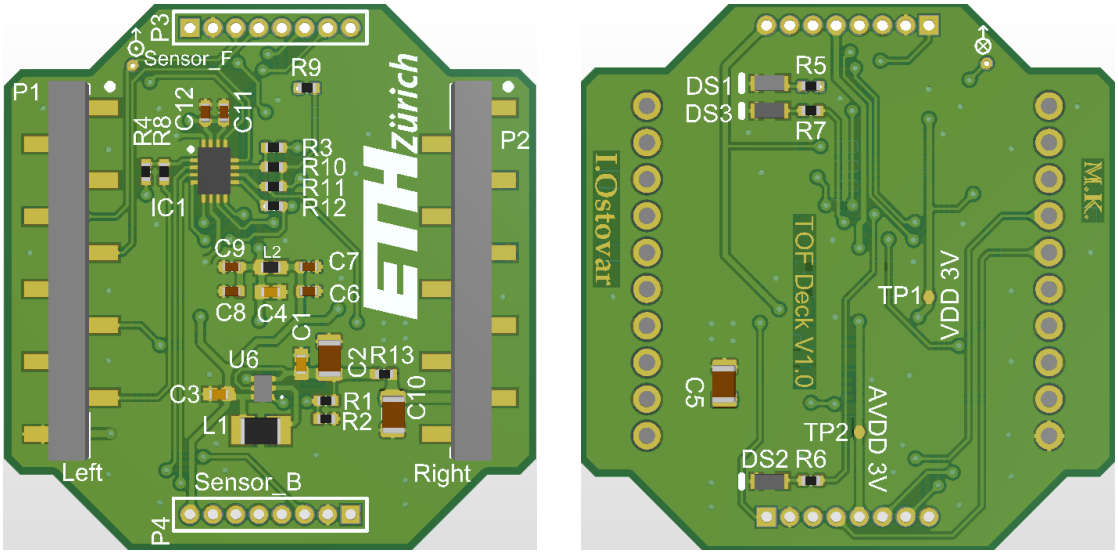
4.1.3.2. ToF Sensor Board

There are two sensor boards, one in the front and one in the back of the drone. The sensor boards consist of pull-up resistors for configuring pins and three capacitors. The capacitors are used for analog and digital inputs of ToF, which makes the input voltage of ToF more stable and lowers the noises. The final board is shown in Fig. 4.4.

4.2. Software

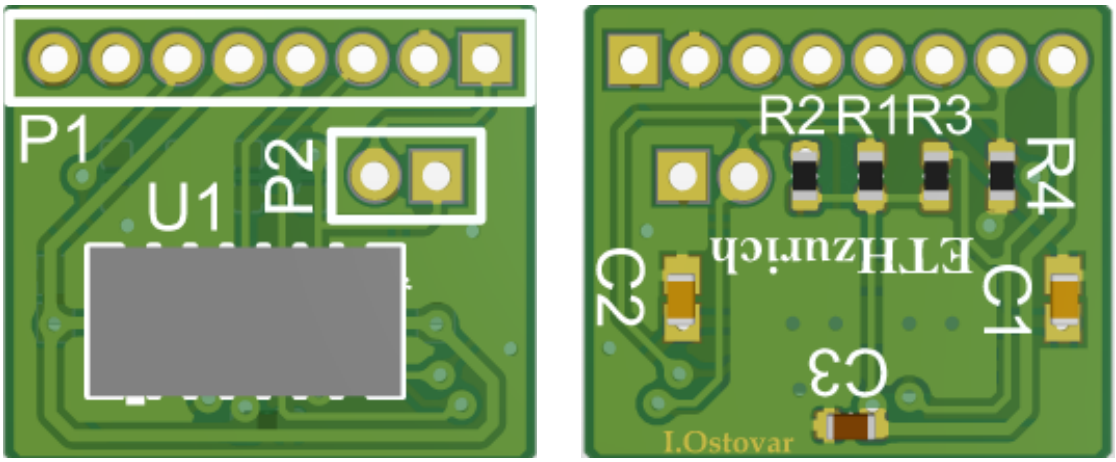
4.2.1. Firmware

The main code is an application implemented over the CrazyFlie main firmware. The main code includes initialization, interrupt handling, and the main loop. In the loop,



(a) ToF deck PCB top view (b) ToF deck PCB bottom view

Figure 4.3.: ToF deck PCB



(a) Sensor PCB top view (b) Sensor PCB bottom view

Figure 4.4.: Sensor PCB

ToF data would be collected and processed. Then based on the process result, the flight plan would be updated. Top-level flowchart is described in Fig. 4.5.

4.2.1.1. Initialize and Configure VL53L5cx

Before configuring the sensors, as all VL53L5cx sensors have similar I2C addresses (0x52), changing the sensors' default address is necessary. Other sensors connected to the same I2C line should be set to low power mode to do this change. Then, the enabled sensor address would be updated using proper commands. It should be done for all sensors.

In the next step, sensors would be configured in this way:

- Resolution: 8X8
- Targets order: Closest target
- Ranging mode: Continuous
- Ranging frequency: 15 Hz (max rate with this resolution)

Now, after configuration is done successfully, sensors ranging would be enabled. In this configuration, sensors would do ranging automatically and generate interrupts when ranging is finished, and data is ready to be collected. Data collection has been done via the I2C protocol like other communication to the sensors. There is a capability to have more than one sensor onboard, so it can have more than one interrupt. So, to manage these sensors, interrupts have been connected to GPIO expander; when any of them are triggered, GPIO expander triggers microcontroller. Later, the microcontroller checks which sensor caused this interrupt and will collect data.

4.2.1.2. Process ToF Data

Only the front sensor is enabled, and the process would be done using its data. After successfully collecting sensor data, it is time to process it. Each sensor frame could include different data, such as:

- Ambient noise in KCPS/SPADS
- Number of valid target detected for one zone
- The number of spades enabled for this ranging
- Signal returned to the sensor in KCPS/SPADS
- Sigma of the current distance in mm
- Measured distance in mm
- Estimated reflectance in percent

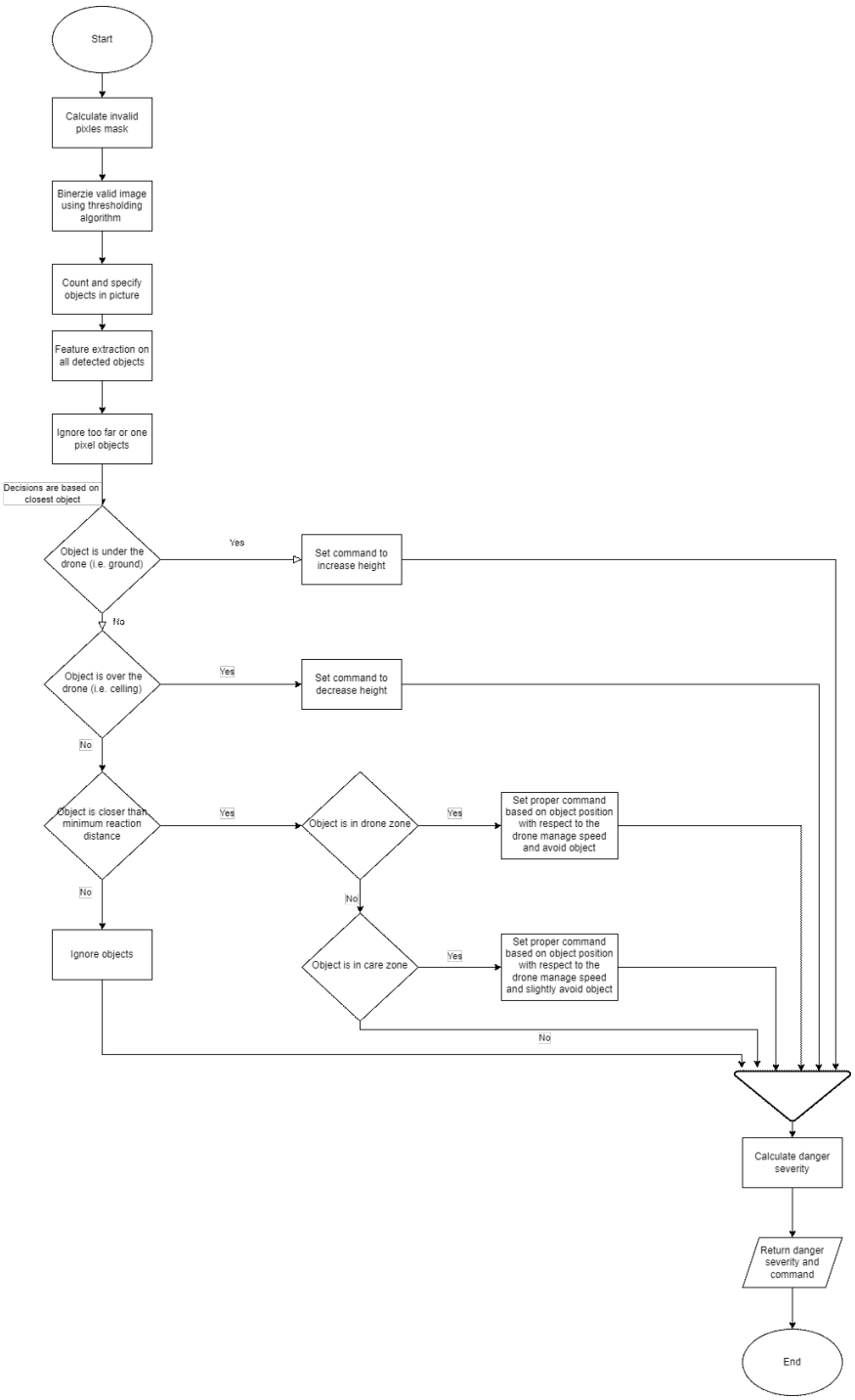


Figure 4.5.: ToF application flowchart

- Status indicating the measurement validity (five & nine means ranging OK)
- Motion detector results

However, for our purposes, only three arrays would be collected from the sensor: *number of valid targets detected for one zone* and *status indicating the measurement validity* for pixel validity check, and *measured distance*.

In the first step of the process, using validity values and the number of targets, invalid pixels could be indicated and removed. The *measurement validity* of each pixel should be five or nine. Other numbers are considered invalid. The maximum number of targets to detect in each pixel has been set to one using this configuration. So, the *number of valid targets* is either one or zero. Zero means no target found in the pixel area. As a result, valid pixels are pixels with the status "five or nine" and "one" as the number detected targets.

In the next step, distances of valid pixels would be binarized, with the threshold *MAX_DISTANCE_TO_PROCESS*. It has been set 2000 (in mm) for now. This means obstacles further than 2 m would be ignored.

Using binarized images, an optimized algorithm could detect all objects in the picture and their corresponding pixels. For now, the algorithm is capable of at most four targets. This boundary has been set to the limit of processor stack (ram) usage.

After object detection, it is possible to do feature extraction on all objects. Most important features include:

- Target minimum distance
- Target middle position
- Number of target pixels
- Target borders

These features are used in a complex decision tree to avoid the object in front of the drone. At first, targets made of pixels less than *MIN_PIXEL_NUMBER* would be ignored. Other targets would be prioritized based on their distances to the drone. Then closest target would be selected to be avoided first.

Here is some predefined value and regions (shown in Fig. 4.6) which would be used in the decision-making process:

- DRONE ZONE: Described with a top-left point(x,y) and bottom-right point(x,y). This zone is the most critical zone to be avoided. It means that if a drone goes forward and a static obstacle is in this zone, the drone will indeed crash with it; hence any object in this zone should be avoided fast, and the speed decrease should be considered too.
  Optimized points: top-left point(3, 3) and bottom-right point(5, 4).

- CARE ZONE: Described with a top-left point(x,y) and bottom-right point(x,y). It means that if a drone goes forward and an obstacle is in this zone, the drone may crash with; but as the objects may move in the future, it is better to slightly avoid the object in advance. Another purpose of this, is that if quadcopters get too close to an object or a wall they may lose their stability. Hence, any object in this zone should be avoided but smoothly, and the speed decrease should be smaller than the drone zone.
  Optimized points: top-left point(2, 2) and bottom-right point(5, 5).
- MIDDLE POS: This is the position of the drone's point of gravity concerning the sensor. As in this configuration, the sensor board is under the drone; the middle position of the image is not precisely in the middle of the sensor but a bit higher.
  Optimized points: Middle point(3, 3.5).
- GROUND BORDER: It corresponds to the objects in the bottom part of the image (i.e. ground). The drone could avoid objects in this zone by increasing altitude and rotation. It is described using a horizontal border(X).
  Optimized ground border(6)
- CEILING BORDER: It is the same as the ground zone but at the top of the image (i.e. ceiling). The drone should avoid these objects by decreasing altitude. It described using a horizontal border(X).
  Optimized ceiling border(2)

Here is some volatile value that would be used in the decision-making process:

- Forward Reaction Distance: The maximum distance that the drone will care about to avoid, and further obstacles will be ignored. This value should be set based on the drone speed. At higher speeds, it should be increased up to the MAX_DISTANCE_TO_PROCESS. It should be decreased for narrow tunnels to make the flight smoother and give the drone the ability to move through them (and not rotate and go back). This value could be updated before or during the flight using CrazyFlie Client.

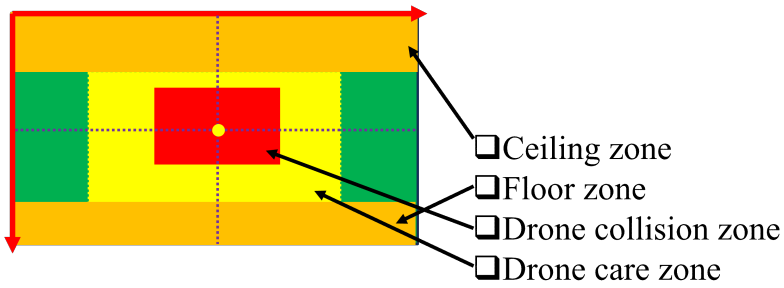


Figure 4.6.: ToF matrix zones

- Forward Slow Distance: This value should be set based on the drone speed and acceleration. If the distances to the obstacle in the drone zone get lower than this value, the drone decreases the speed and rotates to have safe avoidance. This value could be updated before or during the flight using the CrazyFlie client.
- Forward Stop Distance: For obstacles in the drone zone closer than this value, the drone will stop entirely and rotate in its position to the free area. If the drone gets closer than this distance to the obstacle, it may lose stability. This value could be updated before or during the flight using the CrazyFlie client.

Based on the described parameters, the most dangerous object's zone would be determined between these zones:

- Upper left
- Upper right
- Lower left
- Lower right

Based on the obstacle's distance, position and size, a flight command will be generated. Each flight command consists of **move-command** and **danger-severity**. Danger-severity is a float number between 0.0 to 1.0. *Zero* corresponds to the most severe collision condition and *one* demonstrates safe condition.
Move-command will be one of these:

- Command Error
- Stop
- Fast_Right
- Right
- Care_Forward_and_Slow_Right
- Slow_Forward_and_Right
- Fast_Left
- Left
- Care_Forward_and_Slow_Left
- Slow_Forward_and_Left
- Increase_Altitude
- Decrease_Altitude
- Care_Forward
- Slow_Forward

- Forward
- Take_off
- Land

The decision tree would generate flight commands. Based on the most dangerous object's position and distance to the drone, the decision would be taken compared with the described parameter. For example, for the object on the top right side of the drone, in the care zone in the distance between the stop and slow distance, the decision would be *slow_forward_and_left*, and the danger-severity would be calculated in this way:

$$DangerSeverity = (\frac{target\_min\_distance \ - \ forward\_slow\_distance}{forward\_slow\_distance \ - \ forward\_stop\_distance}) \quad (4.1)$$

Moreover, a short history of decisions would be kept in memory to avoid dead-ends stuck. In case of the convex obstacle at the stop distance in front, the drone will rotate to avoid the closer border of the convex. For example (in case the right border is closer), it turns left to avoid a closer object, but while it rotates, the left object gets closer and comes into the drone zone, while the right border of the convex gets further and gets out of the drone zone. The left object gets more dangerous at some point, so the drone rotates right to avoid it. This loop continues, and the drone gets stuck there. To avoid this condition, by utilizing commands history, the drone generates a fast-rotate (*Fast_Left or Fast_Right*) command until getting out of the convex. In the case of other specific navigation procedures, it could be added to the function *Handle_Exception_Commands()*, to be considered. The final process flowchart is shown in Fig. 4.7.

4.2.1.3. Drone Flight Control

This part of the firmware is responsible for deploying flight commands described at Fig. 4.8. It has some critical parameters, including:

- MAX_VX: This is the maximum velocity the drone will go, in case of no obstacles in front of the drone to care about them. Otherwise, based on the collision probability, a value between "0.0" to "MAX VX" would be assigned to the drone velocity. This value could be updated before or during the flight using the CrazyFlie client.
- MAX_AX: This value will limit the drone's forward acceleration during the flight. So, it limits the sudden pitch of the quadcopter. Too much pitch will cause the ToF front to rotate toward the ground, so its front view would be disturbed. Moreover, it is also a safety consideration. Too high acceleration may cause losing the drone stability.
- MANEUVER_RATE: This is the coefficient to control drone agility in rotations. Any value over "1.0" make the drone more agile, and a value below "1.0" makes the drone more careful in rotation. It is better for high speeds to increase this value.

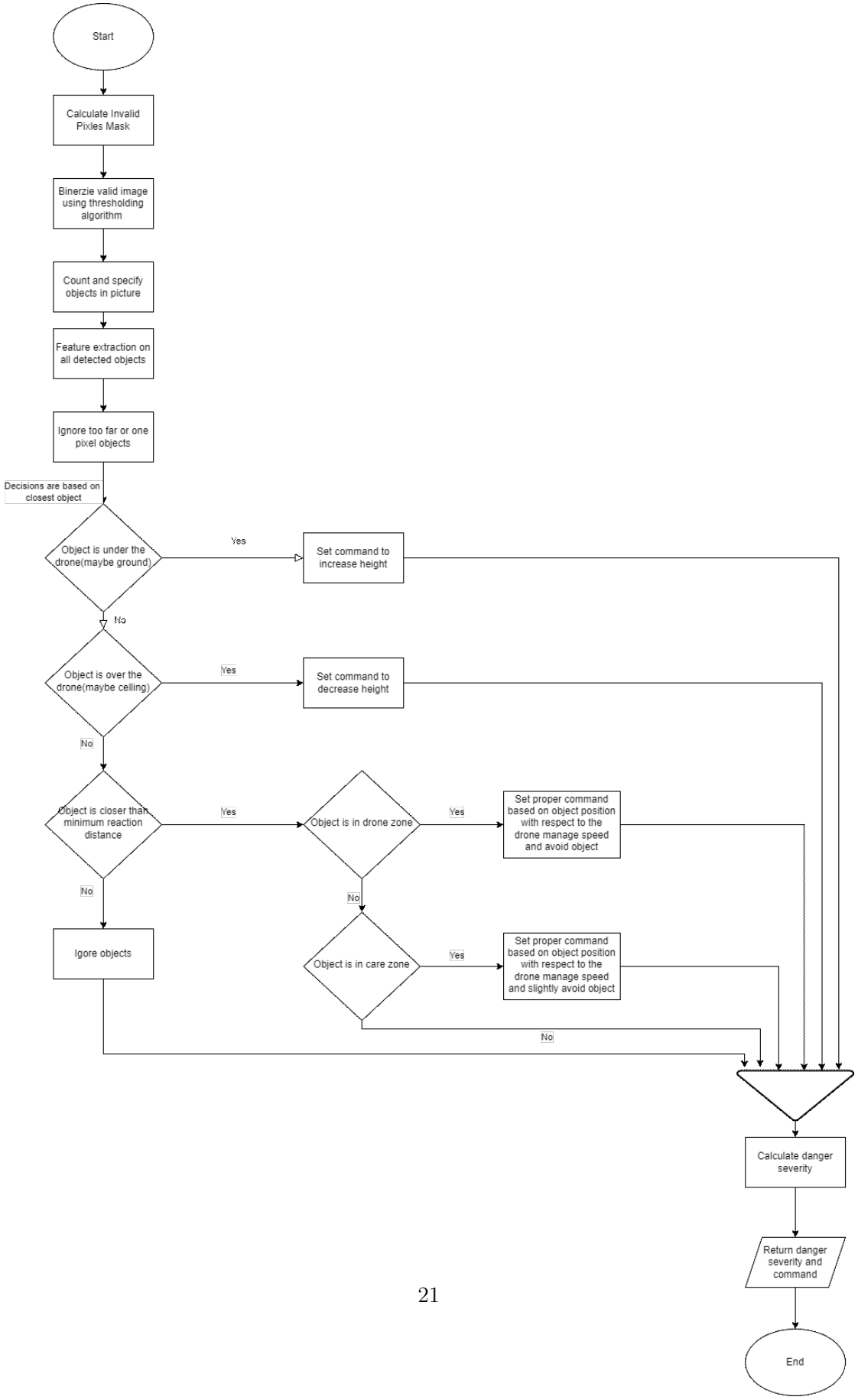


Figure 4.7.: ToF process flowchart

However, a lower maneuver rate for narrow holes and lower speeds will help move through them more precisely. In case of high rotation coefficient, the drone may not pass narrow corridors and escape maneuver and come back.

- START_TAKE_OFF_HEIGHT: This is the default drone altitude. The drone first sets the height to this value at takeoff then starts flying forward. In the case of objects in the ground zone or ceiling zone, the drone will update the altitude; after a while, when the object is avoided, the drone will change the height to the default value again.

By considering those parameters, flight command deployment will be done. The first part of the flight command is the move command, which is the direction of the drone's future move. Each *move command* uses predefined values which are optimized based on the tests. These coefficients are used in the calculation of the maneuvers: for example, for objects in the care zone, coefficients are in the way that the drone decreases the speed a bit and rotate with a low yaw rate; however, for a close object in drone collision zone, decrease in speed is higher, and drone uses higher yaw rate to avoid it.

There are also some restrictions like maximum, and minimum altitude included.

Flight control also saves some status of the drone to avoid sudden movements. It keeps flight time and drone status (landed or already took off). A "To_Fly" parameter could be set before or during the flight, setting the flight time in seconds.

The first command that flight control should receive is "takeoff"; other commands would be discarded. The drone should not be connected to the charger; otherwise, the command would be discarded. After the drone successfully takes off, now other commands would be considered. At this point, the drone will land in one of these cases:

- Land command: The flight control receives a command to land
- Low battery: In case of a low battery, flight control discards all commands and will land immediately.
- Reach To_Fly value: Drone will land when the drone has been flying reaches the To_Fly value in seconds.

The second part of each flight command is danger-severity. The drone will set its speed considering this value and max speed. The forward speed control policy is described in Fig. 4.9.

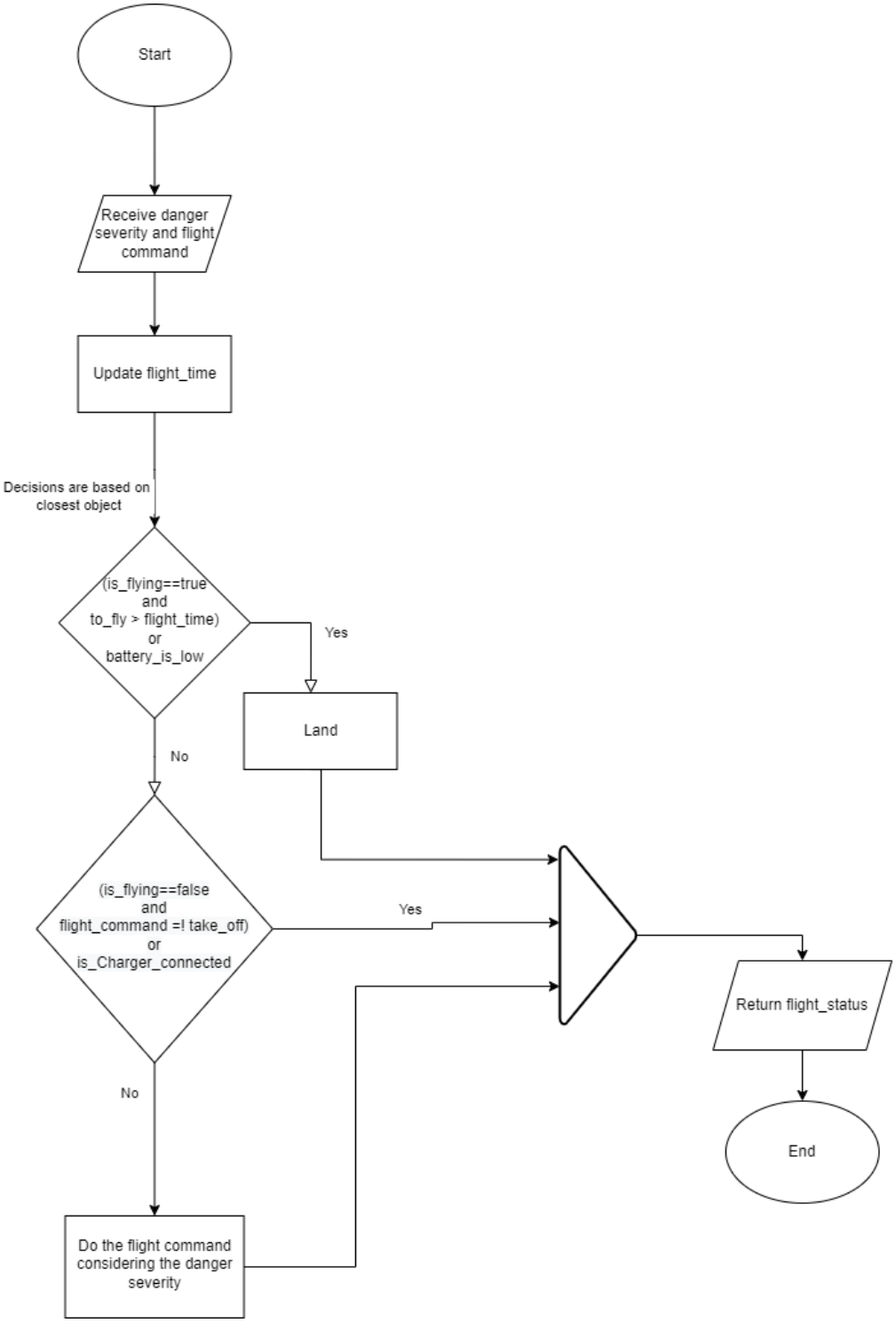


Figure 4.8.: Flight control flowchart

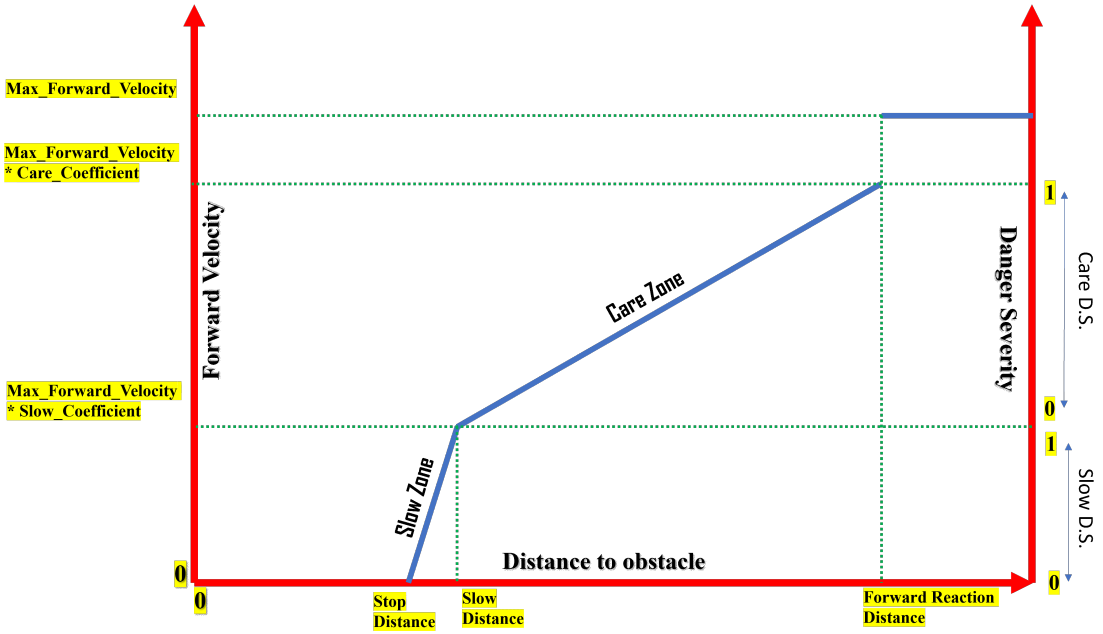


Figure 4.9.: Forward velocity adjustment policy

4.2.2. Python Client

The customized client is a Python application based on cflib[1] from bitcraze[2]. Mainly, it is made of different synchronized processes:

4.2.2.1. Main Process

This process is a bridge for other processes to connect , synchronize their data and communicate.

- Update CrazyFlie position: CrazyFlie has an estimator for position and orientation. This estimator works well but has small drifts, especially on rotations. As a result, after some minutes of flight, the estimation errors accumulate up to some point that is not accurate enough. There exists a feature in the primary process to continuously update CrazyFlie inner estimation with accurate Vicon positioning data.
- There is a particular function to do the process on ToF and generate a command for drone flight control. It was first used to test angle calculation algorithms and

[1] https://github.com/bitcraze/crazyflie-lib-python
[2] https://www.bitcraze.io/documentation/repository/crazyflie-clients-python/2021.1/installation/run_from_source/

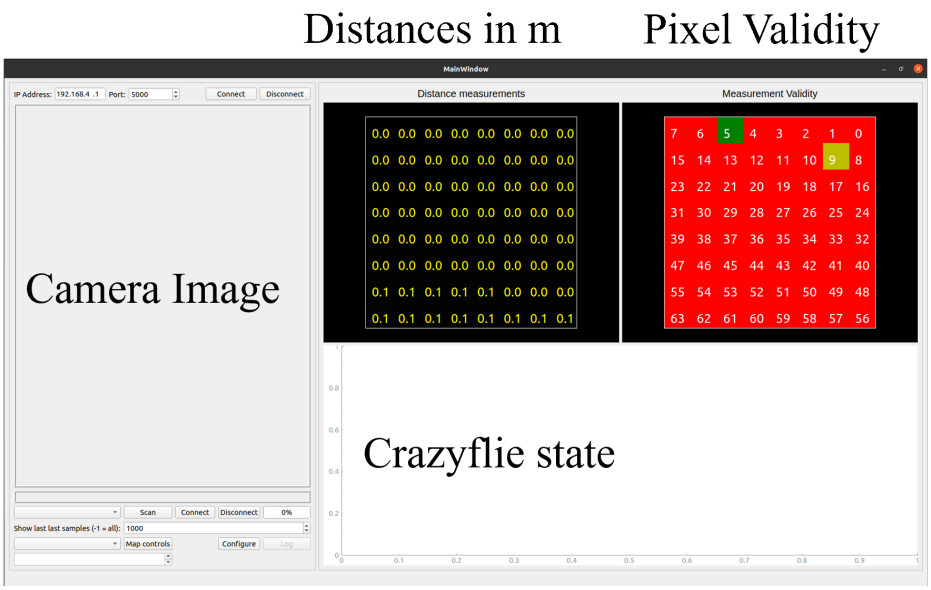


Figure 4.10.: Custom client graphic user interface (GUI)

later for autonomous navigation. However, after validating the proposed obstacle avoidance algorithm, the algorithm moved on to the drone board. At last, this feature has been depreciated but could be used for testing new algorithms.

4.2.2.2. GUI

The GUI shown at Fig. 4.10 is capable of updating plots in real-time. plots are:

- ToF 8*8 distances
- ToF 8*8 pixel status
- Camera images
- CrazyFlie parameters graph

There is also a configuration part for the WiFi connection to AI_Deck, communication to CrazyFlie via Crazyradio, and joystick.

The joystick configuration would be saved at *control_config.json* file. Furthermore, settings related to the CrazyFlie logger are saved at *log_config.json*.

4.2.2.3. Vicon Communicator

This is one of the processors of the custom CrazyFlie client. It is responsible for communicating with the Vicon system and collecting data. It has some configuration parameters, including:

- IP: Default Ethernet IP to connect is 192.168.10.1
- Period: The rate to collect data from the Vicon system
- Subjects: List of the object's name to collect its data from the Vicon system. The Vicon application makes it possible to define some objects with different names. Here, this list should include all the objects we are interested in.
- Address to save: Address to save the data file in
- Time stamp: Synchronized time with CrazyFlie, updating during the process. It would be included in the data log together with the position and orientation of the object at that time.

4.2.2.4. CrazyFlie Communicator

This process will manage all communication with CrazyFlie, including sending commands to the drone and collecting data. It also collects ToF data from the drone, then decodes and saves it. To do logging, it uses the configuration in *log_ config.json* file. There is a list of parameters requested from the drone in this file. The content of the file could be updated using GUI. This process uses *cflib.crazyflie* library.

4.2.2.5. CrazyFlie Controller

This process will handle flight control. It generates a command for Crazyflie based on joystick commands. Joystick control map is available at *control_ config.json* file. There is a list of buttons to be mapped to a flight command in this file. the content of the file could be updated using GUI.

4.2.2.6. CrazyFlie Camera Streamer

This process handles communication with AI_deck via WiFi. It has some parameters consisting of:

- Deck IP: IP of the deck to connect it via WiFi
- Address to save: Directory to save the images

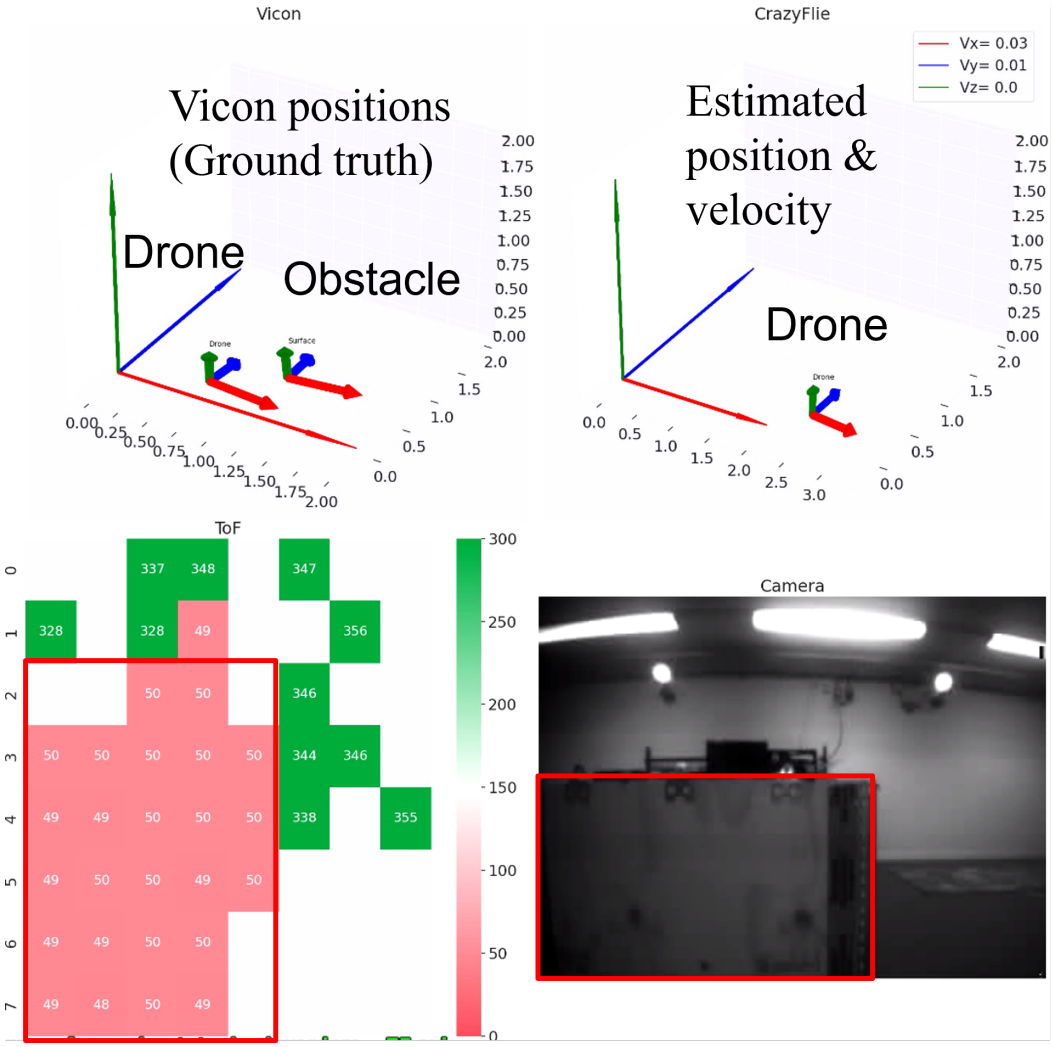


Figure 4.11.: Flight visualizer sample frame

- Time stamp: Synchronized time with CrazyFlie, updating during the process. It would be set to the image names.

4.2.3. Flight Visualizer

This is a Python application used for data validation and flight simulation. It is an optimized multi-process application that can make videos and do post-processes on them. At first, the directory address would be set. Then all data-set in this directory would be read from binary **.dat* files (data-set structure described in detail at Appendix D).

At the next step, all data would be decoded. Then it validates each data-set and prints

out analyze report on the terminal. The report includes different data-set parts, frame rate, duration of the test, etc. The data has now been read and decoded; in the last step, the data are synchronized together using their time stamps and then packed.

The next step is to process decoded data and visualize it. Four different data would be visualized shown in Fig. 4.1a:

- Vicon: This data includes the ordination and position of different objects. Each object would be mapped to a three-axis arrow.
- ToF matrix: This data includes the pixel distance and also pixel validity. This is mapped to an 8*8 matrix with heat-map colors. Invalid pixels would be ignored, and valid pixels are colored based on their distance to the obstacle.
- CrazyFlie parameters: This includes parameters from CrazyFlie like estimated position, orientation, velocity in different axis, battery level, etc. The most useful data was orientation, position, and velocity in our case. The estimated position would plot a 3D map like Vicon data. The velocity would be printed out on the top right part of the image.
- Camera images: This data would be plotted. The plotted image is the synced one with the CrazyFlie time stamp.

All other post-process algorithms could be easily implemented and tested at the next step. Here, for example, I tested two algorithms. One is to calculate the angle to the object, and the other one is the primary version of obstacle avoidance. After validation of the algorithm, it was implemented onboard.

Some more valuable functions are developed to simplify visualization and post-processing.

- Euler_to_rotMat(): It will make a 3D rotation matrix from Euler angular parameters.
- Euler_from_quaternion(): This function convert quaternion values to Euler angles. (Values correspond to orientation from Vicon data is in quaternion)
- Arrow3d(): This function will plot a 3D arrow with set parameters in Python's 3D plot. This is an ad-hoc function developed by me.
- Is_intersect(): This function would return the intersection of the two Python lists.

Chapter 5

Results

5.1. Time-of-Flight Characterization and Calibration

This section presents a performance evaluation of the ToF ranging sensor in terms of accuracy, pixel error rate, and maximum range. The open-source project has been released on GitHub[1].

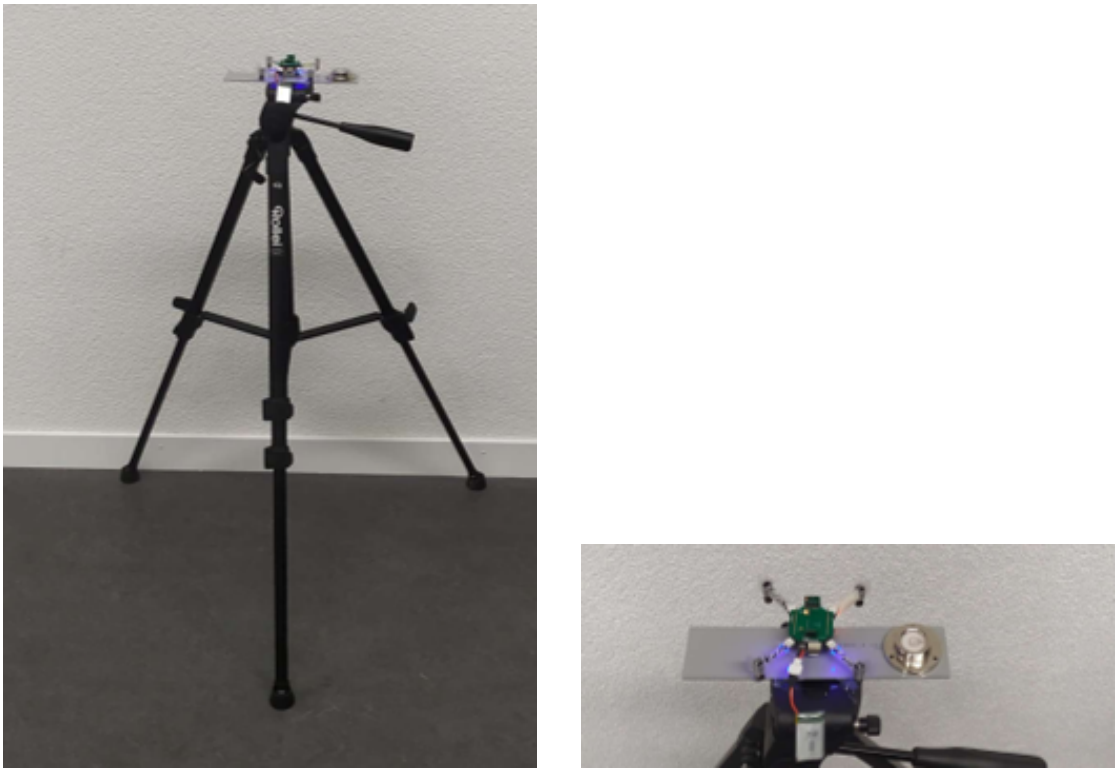
5.1.1. Distance Measurement Setup

In our experimental setup, we place the ToF ranging sensor on a tripod at the height of 1 m. The drone would be put upside down to be fixed with adhesive to the camera stand shown in Fig. 5.1. The drone would be put on a known distance parallel with the wall to the big objects (large enough to span the whole sensor field of view) for the primary tests. The test has been done with a smooth wall and a big wooden plate, as a big obstacle, both in a different light situation.

During our experiments, we sweep the distance between the sensor and the wall in the range 20 cm – 3 m with a step of 0.2 m. We acquire at least 1000 frames in the 8x8 configuration at the maximum rate of 15 Hz for each step, which we save for post-processing purposes. The whole acquisition process described above is performed for four situations:

- White wall /ambient light
- Wooden plate /ambient light
- White wall /darkness

[1] github.com/ETH-PBL/Matrix_ToF_Drones



(a) Setup front view (b) Setup top view

Figure 5.1.: Setup for distance measurement with VL53L5CX

- Wooden plate /darkness

Since both the ambient light and the background color (which influences the amount of reflected light) influence the sensor performance, we chose these four scenarios to maximize the generalization of this characterization.

5.1.2. Distance Measurement Results

We start our evaluation by analyzing the distance error for each pixel at a given distance. Fig. 5.2 shows the mean error and standard deviation of the distance, calculated based on the >1000 acquired frames when the sensor is placed at 1 m in front of the white wall with normal ambient light. There is a visible gradient in the distance mean error from left to right because of the non-perfect alignment between the sensor and the wall. Another consideration is that the corners do not respect this pattern, and lead to mean errors about $25\% - 60\%$ higher than the neighbor pixels. Overall, the mean error is in the range of 19.3 mm – 42 mm. About 90% of the pixels have a standard deviation in

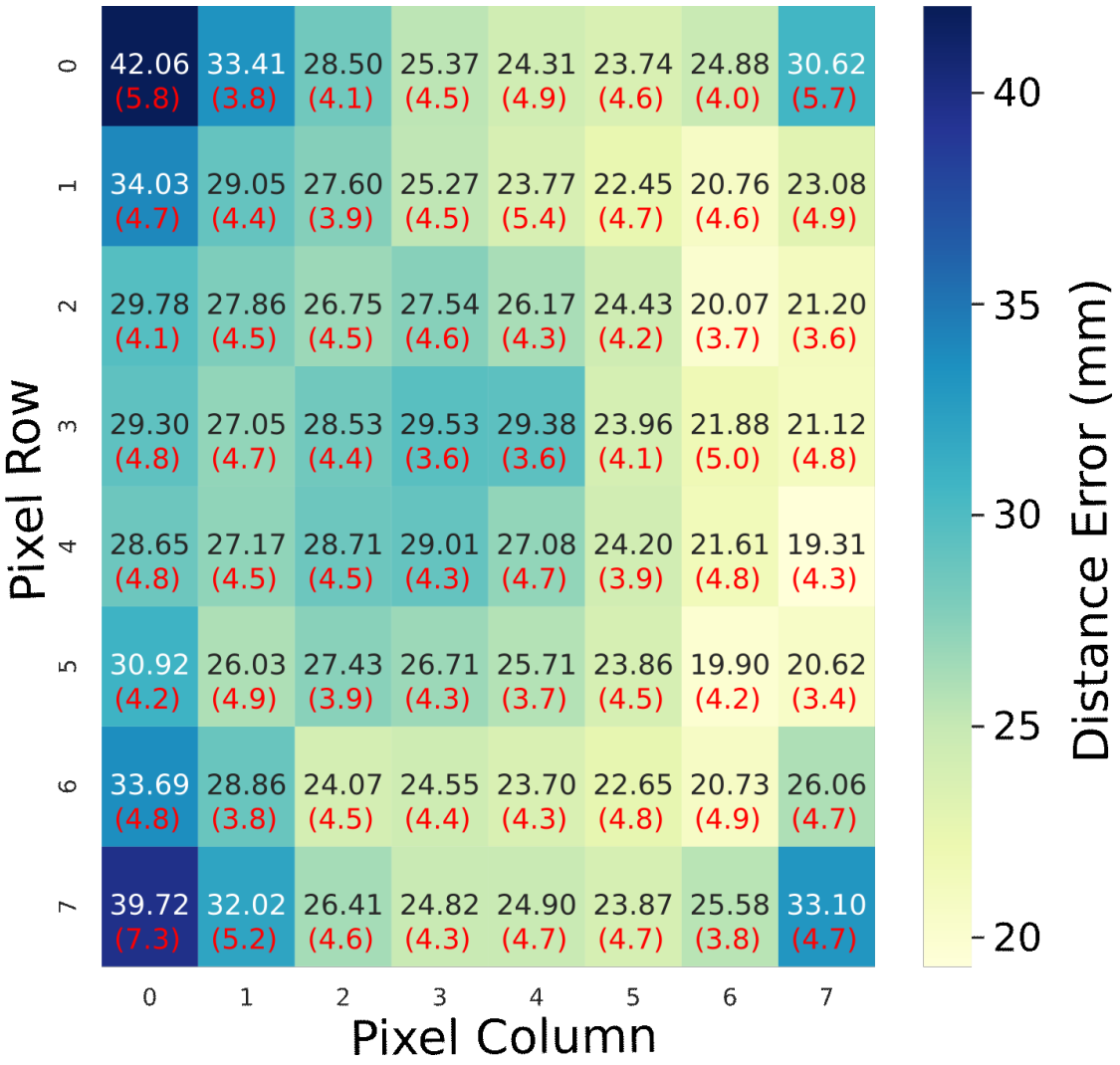


Figure 5.2.: VL53L5CX pixel-per-pixel characterization at 1 m and $\beta = 0°$. Values are in mm. Each pixel includes the offset on the top and the variance on the bottom, computed over 1000 successive samples in a fixed position.

the range 3.4 mm – 5 mm, while the maximum value is 7.31 mm. Once more, the corners lead to relatively larger values compared to the other pixels.

In proceeding, the statistics of the distance error for one single pixel of the sensor would be investigated. Fig. 5.3 shows the results for the distance error as a function of the reference distance within the range 20 cm – 300 cm and a step of 40 cm. In this experiment, we provide the results for all four scenarios. On one hand, there is a similarity between the two experiments corresponds to the white wall, where for each reference distance values, the median difference is mostly less than 0.5 cm. In addition, there is a similarity between the two brown wall experiments, which proves that the ambient light has a weak impact on the accuracy of the distance measurements. On the other hand, the more substantial impact is caused by the background color the sensor points to because the difference in accuracy is over 2 cm when comparing the different walls cases; However, it looks more like a constant bias (offset in distances) rather than noise. Moreover, in terms of the *min-max* and inter-quartile difference, the results are about the same for all four scenarios, given a certain reference distance. While for a reference distance equal to 20 cm, the *min-max* difference is below 1 cm, a reference distance of 300 cm leads to a *min-max* of over 8 cm and an inter-quartile difference of about 2 cm.

Finally, Fig. 5.4 shows the relation between the invalid pixels and the reference distance. In particular, specific environmental factors, such as solid ambient lighting, large distances, or poorly reflective objects, can influence the number of successful measurements in a frame. The experiments with a white background scenario lead to a loss lower than 1% for a sensor-wall distance of 200 cm. This is a better result than the brown background scenario, which achieves losses of 2% and 5% for the cases with ambient light and darkness. In conclusion, regardless of the scenario, the pixel validity is higher than 95% for distances up to 200 cm and about 50% for 260 cm.

5.1.3. Angle Calculation Setup

Here, nano-drone is fixed in specified distance to a wide, flat obstacle (i.e., a wall), with a surface more extensive than the sensor FoV and a predefined angle.

Fig. 5.5 shows the drone facing the obstacle with an angle β and a distance d. Each pixel is addressed as $p_{m,n}$, see Fig. 5.6. The 45° FoV is described by the θ angle, while d_n represents the projected planar distance of the C_n column, where $n, m \in [0, 7]$, $n, m \in \mathbb{N}$. To compute the steering angle, proportional to the drone's yaw angle, the average distance along each column would be computed, considering only the valid pixels ($d_n = (1/m) \sum_{m=0}^{7} p_{m,n} \mid p_{m,n}$ is valid). If N_n is the number of invalid pixels for the column n, then the whole column is discarded if N_n is higher than a threshold N_{max} and the column is considered invalid. The probability of having an invalid column C_n is proportional to the dependency between invalid pixels and distance is given in Fig. 5.4.

Concerning Fig. 5.5, the obstacle's approach angle (β) is estimated from a single frame. r and l are the indexes of the first valid columns starting from the left side and right side,

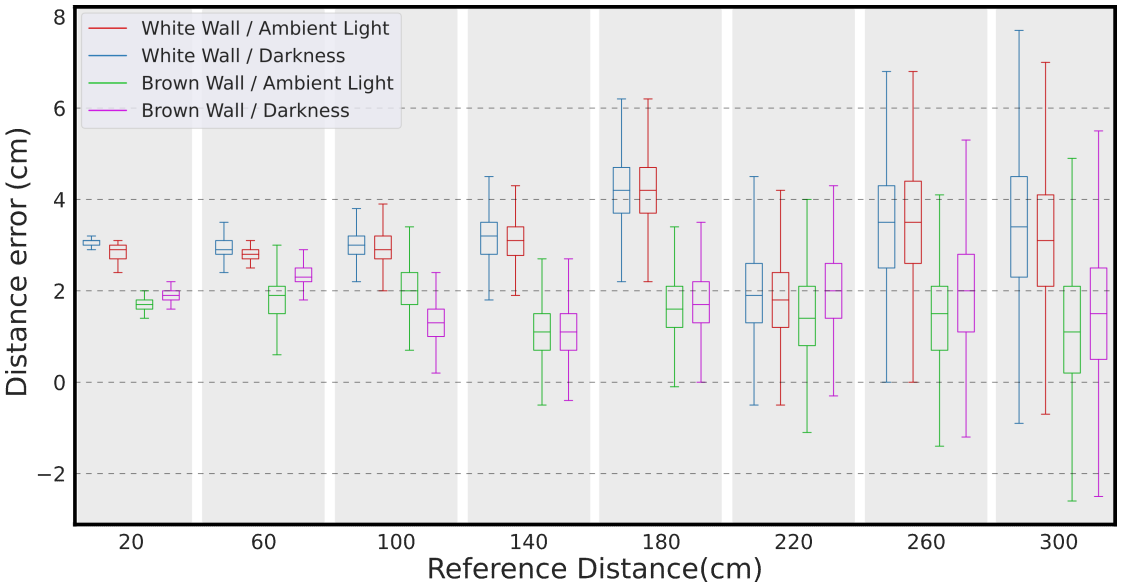


Figure 5.3.: The measurement error is a function of the absolute distance. The variance and offset characterization is performed for the distance range 20 cm – 3 m with a step of 40 cm. The characterization is performed for the four different scenarios are mentioned in the legend.

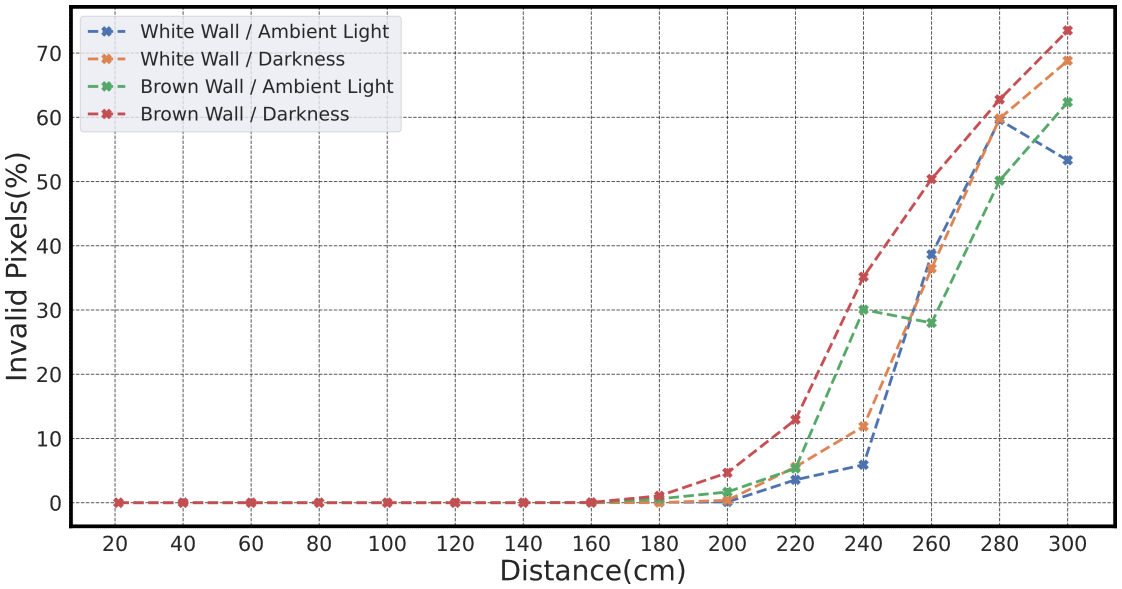


Figure 5.4.: The pixel validity is a function of the distance. The figure shows how the average percentage of valid pixels per frame decreases when the absolute distance increases. Tests are performed for a distance range of 20 cm – 3 m with a step of 20 cm.

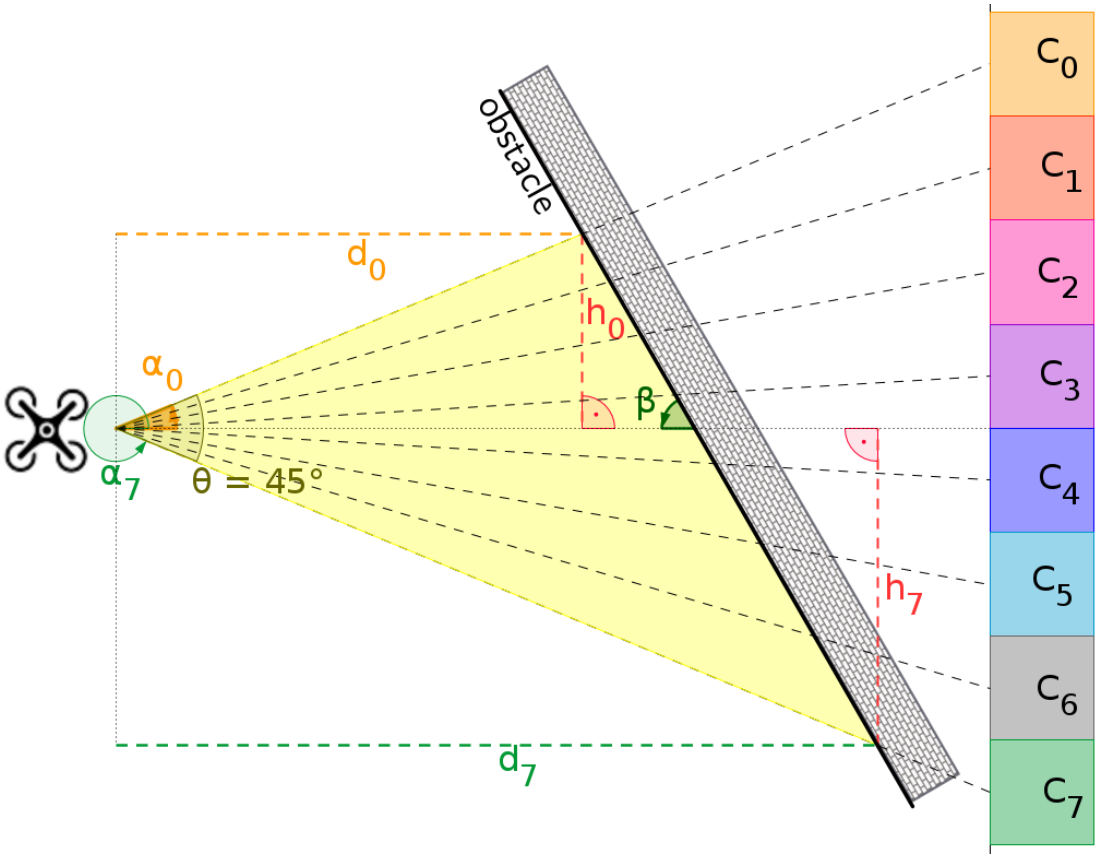
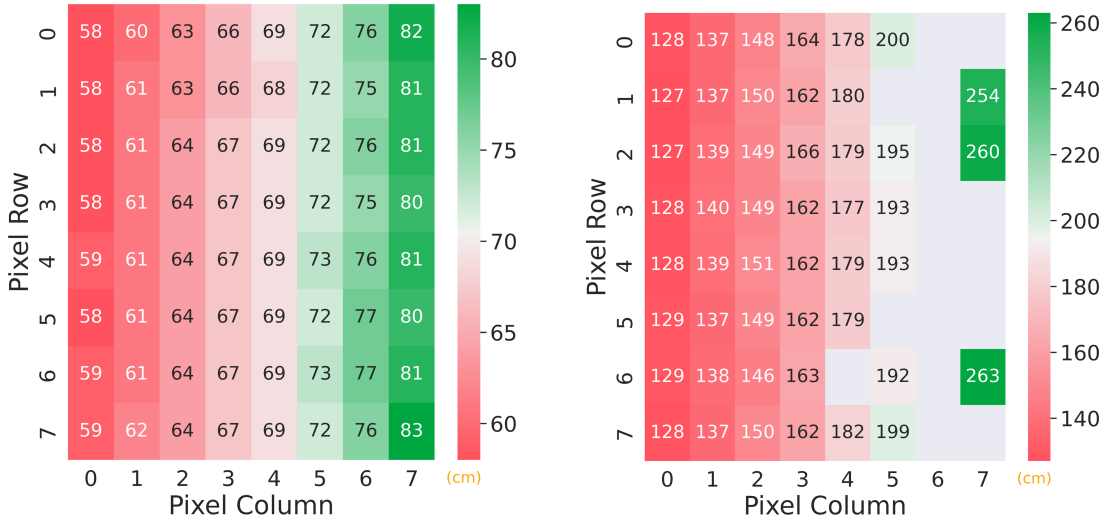


Figure 5.5.: The drone faces an obstacle with an angle β. C_x is the corresponding column associated with the 8x8 matrix, while d_x is the project's planner distance. The term h_x is calculated using the ToF sensor FoV and the measured d_x.

respectively. If the number of valid columns is smaller than 2, the frame is considered void and thus discarded. Considering the listed physical and mathematical conditions and taking Fig. 5.5 as a reference, angle β can be estimated, as expressed in Eq. 5.1. In consideration of the relative FoV, which differs from the nominal specification due to a variable number of valid columns, Eq. 5.2 calculates the new sensor scope. Note that in Fig. 5.5, $h_n = \tan(\alpha_n) \cdot d_n$.

$$\beta = \arctan\left(\frac{\tan(\alpha_l) \cdot d_l - \tan(\alpha_r) \cdot d_r}{d_r - d_l}\right), \tag{5.1}$$

$$\alpha_n = \arctan\left(\frac{7 - 2n}{7} \tan\left(\frac{\theta}{2}\right)\right). \tag{5.2}$$



(a) The drone faces an obstacle at exactly $\beta = 60°$ angle. The center of the matrix lies at 64 cm from the obstacle surface. All the pixels are valid.

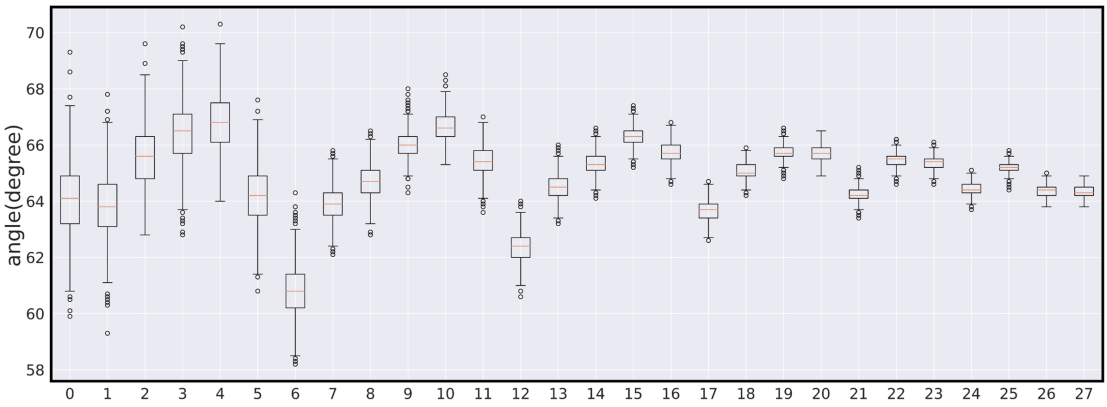
(b) The drone faces an obstacle at exactly $\beta = 45°$ angle. The center of the matrix lies at 161 cm from the obstacle surface. 25% of the pixels are invalid.

Figure 5.6.: Two examples of the drone facing an obstacle at different distances and angles. Gray pixels point out invalid pixels. The distance is provided in centimeters and is not calibrated.

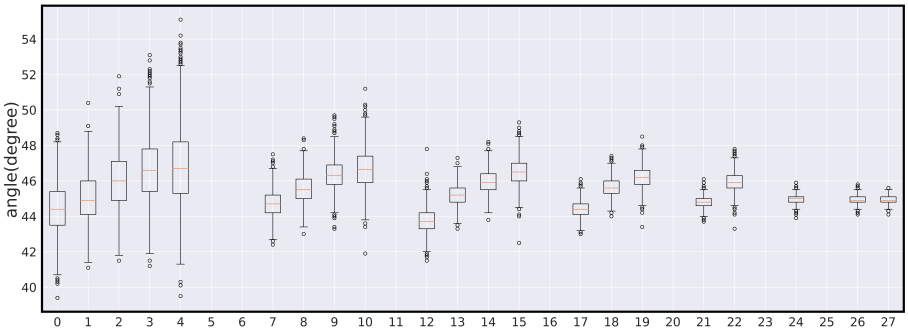
5.1.4. Angle Calculation Results

Figures 5.6a and 5.6b show an insight into a particular flying test, in which the nano-drone is approaching a wall with different distances and β angles. In Fig. 5.6a, the obstacle is found at a distance of 64 cm from the drone with a β angle of 60°, while in Fig. 5.6b the obstacle is located at 161 cm and an angle of 45°. The latter shows an invalid pixel rate of 25%. Despite the reduced number of valid pixels in Fig. 5.6b, the β angle can still be computed using Eq. 5.1 due to the sufficient number of valid columns. An initial calibration is considered, which uses the pixel-by-pixel characterization presented in Fig. 5.2 to perform offset compensation.

Considering Fig. 5.6a and Eq. 5.1, β can be extracted from any pair of d_l and d_r. For example, the angle extracted from d_0 - d_7 is 59°, while considering d_1 - d_3, β is estimated at 58°. In total, 28 combinations are possible; the statistics result for more than 1000 frames corresponds to angle calculation algorithm using 8*8 ToF has been described at Fig.5.7. The first seven columns of boxplots are related respectively to seven combinations of neighbors columns. The following six columns correspond to the combination of columns with one column distance in between. This procedure continues until the last boxplot corresponds to the first and last column combination. As in 45° image some columns set as invalid due to excessive invalid pixels, their corresponding contributing to box plots



(a) 28 combinations results for the drone faces an obstacle at exactly $\beta = 60°$ angle. The center of the matrix lies at 64 cm from the obstacle surface. All the pixels are valid.



(b) 28 combinations results for the drone faces an obstacle at exactly $\beta = 45°$ angle. The center of the matrix lies at 161 cm from the obstacle surface. 25% of the pixels are invalid.

Figure 5.7.: Two examples of the drone facing an obstacle at different distances and angles. Gray pixels point out invalid pixels. The distance is provided in centimeters and is not calibrated.

have been missed Fig.5.7b. All in all, It is estimated a precision of $\pm 4°$ for the cases of d_l and d_r lower than 160 cm (i.e., no invalid pixels).

In Fig. 5.6b, the noise presence is indicated by void gray pixels. Indeed, columns $4 - 7$ point to distances above 160 cm with a non-zero probability to have invalid pixels/columns, according to Fig. 5.4. However, also, in this case, the approaching angle can be computed. By imposing N_{max} equal to 4 (i.e., a 50% error tolerance) then d_5 is the first considered column ($N_n > N_{max}$) when starting from the right. In this case, we calculated $\beta = 46°$, with an average precision of $\pm 6°$.

The proposed approach can also be used to consider rows for vertical movements and can be applied for all the obstacles wider than two columns/rows. In the case of a small object close to the UAV, background pixels can be discarded by invalidating pixels outside the

view of interest. Moreover, the sensor automatically labels the pixels associated with out-of-range distances as invalid. Our case study also demonstrated the system robustness, which can work in scenarios where up to 75% of the pixels are invalid.

5.2. Obstacle Avoidance Using One Front ToF

At this section, We see the proposed obstacle avoidance algorithm results.

5.2.1. Complexity

The algorithm is implemented on STM32F405 MCU (168MHz, 196kB RAM). It is made of two main parts: The first is the process ToF image and takes 17010 cycles. The second part is flight control, which takes 133 cycles. All these values are the mean value while disabling all drone interrupts. The drone has lots of low-level processes to do, so by disabling the interrupts, the drone will stop working soon. Consequently, to simulate the actual test, cycles are also measured in case of all interrupts on, and both processes take together 68080 cycles which correspond to 450 µs. As the frame rate of the sensor is 15 Hz, there is 66 ms available between each frame. So the whole process, including interrupts, takes 0.7% of available time.

5.2.2. Reliability

Our system proved reliable (>95%) in-field obstacle avoidance capabilities when flying in indoor environments with dynamic obstacles. Mostly, the cause of the failures are among these two situations:

- Drone drift and state estimator error: Intrinsic quad-copters drift which moves drone too close to the wall in sides or back (out of ToF field-of-view)
- The drone flies too close to the wall, especially in the sides: This causes the drone to lose its stability and crashes
- Large glass surfaces perpendicular to the drone trajectory: It is invisible for the VL53L5CX sensor

5.2.3. Ratings

The CrazyFlie weighs 27 g and can fly up to seven minutes without any extra payload using a fully charged 350 mAh.

Any additional weight decreases the flight time, and any payload heavier than 15 g prevents the drone from taking off. To maximize the drone's flight time, the designed PCB

weights only 2.2 g(1.86 g deck + 2*0.21 g sensor board), increasing the total mass by only 5%.

Depending on the configuration, the acquisition rate is 15 Hz (8x8 pixels) or 60 Hz (4x4 pixels) with a horizontal and vertical field of view (FoV) of 45° and a power envelope of 286 mW.

The actual FoV can slightly change; its detection volume depends on the environment and sensor configuration and target distance, reflectance, ambient light level, sensor resolution, sharpener, ranging mode, and integration time.

Chapter 6

Discussion

6.1. ToF Sensor Suitability

We were unsure if the sensor would work because it was a new sensor (even not commercialized) when I started this project (August 2021). No one had tried this sensor on drones; as a result, no data-set was available. The first step is to deploy the sensor on the open SW/HW platform, which is reliable enough to handle all low-level tasks. The next step is to characterize the sensor and check whether it is proper for drone applications or not. As the preliminary test results seem good accuracy and reliability in different environments, the next test had proposed. Secondary tests are in the case of drone flying. It includes some simple maneuvers like taking-off/Landing and rotating around. The sensor also seems reliable in this test.

ToF data is much easier to be used in algorithms as it has much fewer data concerning vision-based methodologies. Also, it is more simple to do feature extraction on ToF data rather than camera images.

6.2. Obstacle Avoidance Reliability

The proposed methodology shows outstanding reliability even with complex objects and narrow tunnels. In the project repository, all tests have been done using firmware at git commit "test MODE": 4e4ece61.

1. In this scenario, the drone easily passes the first test at the maximum speed of 2 m/s.

2. This scenario has been tried with different speeds. Results have shown no crash in all tests. The pathwidth is between 1.3-1.4 m, and the corners should be good enough without a hole, so the ToF sensor does not get confused.
3. This scenario was replaced with the test with two rotations, one 30 and another 150, with no crash in different configurations, at maximum speed and acceleration.
4. In this scenario, the drone goes through a narrow 1 m tunnel with one 10 cm width obstacle in the middle. This is the minimum width that the drone could pass without a crash. The tunnel width decreased until the tunnel's wall impeded the drone's airflow, and the drone could not keep height and stability anymore.
5. Most crashes in this scenario happen to avoid objects with small heights. In this case, the drone could go over and crash due to Optical Flow (on flow-deck) sensor limitation. Otherwise, the drone passes the test with a more than 95% successful rate.
6. The drone is can fly non-stop up to 7 min until its flight time is finished or the battery is drained.

6.3. Compare to Related Works

Here, the proposed system could reach very high reliability using a miniaturized light-weight ToF sensor to utilize an optimized algorithm. Both software and hardware have requirements to be deployed on conventional nano-drone. This approach outperforms state of the art solution on nano-drones, with its reliability shown in the table 6.2.

Other methodologies also have more complexity and stricter limitation like Ultra Wide-Band (UWB) distance measurement. UWB's approach needs that the obstacles be defined in advance. This means the flight map is necessary for navigating through obstacles. In the case of avoiding other drones in swarms, the number of drones would be limited by the UWB latency.

Table 6.1.: Algorithm test scenarios

Scenario #	Name	Description	Obs.
1	Dynamic obstacle avoidance vs. speed	The drone is flying straight. When the drone is located at 1.5m from a reference point, the user steps in the front of the drone (in that reference point). The idea is to evaluate the maximum speed of the drone at which it can still brake in time.	Vicon should be used.
2	drone steering in a 90deg pipe	The idea is to have the drone flying in a small tunnel with a 90deg angle and check at which speed the drone can still succeed the steer. The width of the pipe should be big enough, around 1.3 - 1.5m.	Vicon should be used.
3	drone steering in a 45deg pipe	The same as previous, but with a 45deg angle	Vicon should be used.
4	drone flying in a narrow pipe	The idea is to have the drone flying straight in a narrow tunnel without obstacles and check the minimum width of the tunnel at which it can still successfully fly through it without a crash. The length of the tunnel should be about 4m. The same experiment can be repeated with an obstacle in the middle of the tunnel.	
5	Flying in environments with people	Have the drone fly indoors, at a constant speed, not necessarily very fast (maybe 0.5-0.7m/s). The idea is to have static and dynamic people in the room and prove that the drone can avoid them while flying for a minimum time: let us say one, two, or three minutes, depending on the performance of the avoidance algorithm.	
6	Evaluate the maximum indoor flight time	The idea is to place the drone in a room with multiple static and dynamic obstacles and evaluate the maximum flight time - ideally until the battery runs out.	

Table 6.2.: Reliability compare to state of the art

	One front 8*8 ToF	4 LASER rangers	Camera+radar	UWB avoid drones
No obstacle	99%	100%	100%	100%
Low obstacle density	99%	85%	95%	98%
High obstacle density	98%	75%	80%	80%

Chapter 7

Conclusion and Future Work

Starting from the ST's pre-commercialized multi-zone ToF sensor, we reached a reliable obstacle avoidance system on nano-drones at the end. At first, the main problem was to find out that, in any way, this new sensor is suitable for the nano-drones use case. In the first step, a new customized development PCB should have been designed to integrate the sensor in the Crazyflie prototype nano-drone. One of the challenges was strict constraints on the PCBs; The sensor's vertical PCB should have a very short height, not touching the ground to avoid damage to the board. Also, the main board uses the variable voltage of the drone battery to provide filtered analog and digital voltages. Moreover, Crazyflie supports many different decks with different capabilities and as a result, different pin configurations. In our case, the flow-deck and AI-deck, in addition to the customized ToF deck, are connected to the main board via the shared connector. As a result, the ToF deck should be compatible with all of this and no other previous working boards are available. In the next step, a complex logger system has been designed to collect synchronized data from different sensors including, the VICON system, WIFI camera, and drone parameters. Having access to a rich dataset collected in many different conditions allows performing different post-processing analyses to calibrate and characterize the sensor. In the end, the results show that the characterization of the sensor is suitable for nano-drone navigation and obstacle avoidance. In the last step, an innovative algorithm has been proposed to control the drone speed and direction using only the ToF sensor's data for the first time. The first version of the algorithm has been tested offline on a dataset that includes camera images and precise drone position from the VICON system as the ground truth. After showing the good result on offline tests, the algorithm was implemented in the computer control system in python to refine more and also do the in-flight test. Thereafter, the final algorithm is optimized and implemented in the drone's firmware to do the final in-flight test exploiting only the on-board processing capability of the nano-drone. Subsequently, a large number of tests have been done in many different in-door situations to debug the algorithm and calibrate algorithm parameters more precisely to

have a faster and more reliable control system, and to test the new control system's capability. As discussed before, the proposed system provides reliable and lightweight navigation and obstacle avoidance capability to be used as a state-of-the-art method on nano-drones.

As the platform works fine, the work could be continued by :

- First, to optimize and make current obstacle avoidance better, faster, and more reliable. As it is possible to run simple neural networks on AI-deck, AI approaches also could be considered as an alternative.
- Second, to add an autonomous navigation approach along with obstacle avoidance. Currently, the drone will go forward and avoid an obstacle in its way. Adding a smart navigation algorithm would be the next big step in this project.
- Third, provide a system that uses this new sensor on the nano-drones swarm to reach higher reliability.

Appendix A

Task Description

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Center Project-Based Learning

Tasks Description at the
Department of Information Technology and
Electrical Engineering

for

Iman Ostovar

On the exploration of new sensors for autonomous nano-drones

Advisors: Dr. Tommaso Polonelli
Vlad Niculescu
Hanna Müller

Professor: Prof. Dr. Luca Benini

Handout Date: 15.08.2021
Due Date: 15.02.2022

1 Project Goals (14 weeks)

Nowadays, the industry is pushing the evolution of autonomous flying vehicles (UAV) toward rapidly decreasing form factors and increasing of computational capabilities. Usually, the development is either in the one direction (adding intelligence, computational and sensing capabilities) or the other (miniaturization). This means that on nano-sized aerial vehicles flight control, navigation, and planning capabilities are either absent or based on monocular low-resolution cameras. In contrast, relatively big-sized drones (i.e., DJI or Matek) have been proven capable of impressive autonomous sense-and-act capabilities, running in real-time complex and computationally intensive multi-sensor and vision-based control systems.

Therefore, this project focuses on this challenging class of Unmanned Autonomous Vehicles, characterized by few centimeters in diameter, tens of grams in weight, and a few Watts total power consumption, of which 85% is to be dedicated to the propellers, leaving a total power budget for onboard sensing, computation, control actuation of only a few hundred mW. Thus, thesis focuses on enabling the nano-drone navigation without the support of BN or RGB camera by adding additional sensing capabilities through state-of-the-art off-the-shelf ICs. Through the usage of multi-zone ranging sensors (VL53L5CX) and mm-waves radars for obstacle avoidance and UWB ranging for indoor localization, the student will design, develop and, finally, test a working demo with a flying nano-drone (Crazyflie).

Tasks (6 months)

The project will be split into four phases, as described below:

Phase 1 (1 Month)

1. Investigate and study the state-of-the-art technology of nano-drones, UWB ranging and ToF sensors.

2. Introduction to PCB design, Altium: schematics and layout;

3. Getting started with the Crazyflie environment, IDE, SDK, DEKS, and cross compilation/programming;

4. Hardware design, ToF deck for the Crazyflie (1 or 2 VL53L5CX supported)

Phase 2 (2 Months)

1. Real-time data logger using the Bitcraze Crazyradio. The following data must be collected: drone attitude, control commands, VL53L5CX ToF data, UWB relative localization, absolute localization (VICON).

2. Dataset collection in different environments: indoor, outdoor, with static and dynamic obstacles, narrow corridors/pipes, etc.

3. Dataset classification and labelling.

Phase 3 (2 Months)

1. Exploration of autonomous navigation algorithms (e.g., CNN) capable of obstacle avoidance and steering using one (or more) VL53L5CX and the UWB relative localization.

2. (Optional)Exploration of path planning through Model Predictive Control.

3. Implementation of the algorithm on the Crazyflie platform.

4. Flying demo

Phase 4 (1 Months)

1. Finalizing the tests.

2. Write the final report and prepare the presentation.

3. Journal/Conference paper publication

Milestones

By the end of **Phase 1** the following should be completed:

- Crazyflie deck.
- Good understanding of the Crazyflie platform.

By the end of **Phase 2** the following should be completed:

- Real-time data logger over the Crazyradio.
- Dataset collection.

By the end of **Phase 3** the following should be completed:

- Flying demo with obstacle avoidance capabilities.

By the end of **Phase 4** the following should be completed:

- Project documentation.
- Final presentation.
- Final report, including final results.

Thesis Organization

The Master's degree program concludes with a Master's thesis that lasts six months. The project includes an oral presentation and a written report (the Master's thesis), and it is graded. Before starting the project, the Master thesis must be registered in mystudies ("Projects/papers/theses"). You will be admitted to the Master thesis only if both semester projects are successfully completed. During the thesis, students will gain initial experience in the independent solution of a technical-scientific problem by applying the acquired specialist and social skills. The grade is based on the following: (i) Difficulty of the project; (ii) Student effort and learning curve; (iii) Results in terms of quality and quantity; (iv) final presentation; (v) documentation.

Before starting, the project must be registered in myStudies ("Projects/papers/theses").

Weekly Report

There will be a weekly report sent by the candidate at the end of every week. The main purpose of this report is to document the project's progress and should be used by the student as a way to communicate any problems that arise during the week.
Weekly meetings will be held between the student and the assistants. The exact time and location of these meetings will be determined within the first week of the project in order to fit the students and the assistants schedule. These meetings will be used to evaluate the status and progress of the project. Beside these regular meetings, additional meetings can be organized to address urgent issues as well. The report, along with all other relevant documents (source code, datasheets, papers, etc), should be uploaded to a clouding service.

Final Report and paper

Documentation is an important and often overlooked aspect of engineering. One final report has to be completed within this project. The common language of engineering is de facto English. Therefore, the final report of the work is preferred to be written in English. Any form of word processing software is allowed for writing the reports, nevertheless the use of LaTeX or any other vector drawing software (for block diagrams) is strongly encouraged by the IIS staff.

PDF copies of the report are to be turned in.

Appendix B

Declaration of Originality

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Nano-drones: enabling indoor collision avoidance with a miniaturized Multi-zone Time of Flight sensor

Authored by (in block letters):

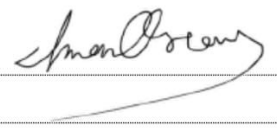
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Iman	Ostovar

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zurich, 23.02.2022	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.

Appendix C

File Structure

- Documents/
  - AI-deck 1.1 ..... AI-deck documents
  - Crazyflie ..... CrazyFlie main board HW & SW documents
  - DC2DC ..... DC to DC convector TPS62230 documents
  - flow V2 Deck ..... Flow-deck documents
  - Loco Positioning deck ..... Loco-positioning-deck documents
  - miscellaneous ..... tca6408a gpio expander and power filtering techniques
  - Papers ..... Useful papers and previous project reports
  - Report ..... Master thesis report and presentation
  - TOF sensor ..... VL53L5cx SW & HW documents
  - Vicon Sdk ..... VICON platform software requirement
  - intilize Vicomroom.sh ..... VICON platform installer
- Hardware/
  - manufactured_version ..... Latest board send for manufacturer
  - MultiBoard_Project ..... Latest all boards together project
  - Old Projects ..... Preivious useful projects
  - Rules ..... Design rules used for PCB design
  - SensorBoard ..... Latest Sensor board project
  - TofDeck ..... Latest ToF-deck project
- Software/
  - app_tof_matrix ..... ToF application based on CrazyFlie firmware
  - CrazyFlieProjects ..... All Crazyflie project repository (external repository)
  - dataset-building-framework .. Logger Client, flight visualizer, & other python projects
  - dev ..... test codes used for developments
  - Sample Codes ..... Sample codes used in both firmware and python client
  - ViconRoom ..... VICON room sample project (external repository)
- Data Log/

- `Angle validator` ..... Data-log related to the test of angle calculator formula
- `first flight test` ..... Preliminary flight tests
- `Flight DataSet` ..... The main Data-set directory
- `LogResults` ..... Visualized of some data-sets
- `object movement _ for camera sync` .... Data-log related to the Camera-ToF synchronization
- `Stationary Test` ..... Data-log related to the on-ground test
- `ToF Distance Test` ..... Data-log related to the accuracy validation
- `vicon test` ..... Data-log related to the VICON validation

Appendix D

Data-set

Flight data-set folders and the test scenario have been completely described at *Data Log/Flight DataSet/Flight Dataset Description.xlsx*. List of folders described at TABLE D.1.

Each folder consists of :

- *state_ Crazyflie_ group00.csv*: This file is a text-based table including CrazyFlie's primary state estimations like:
  - 3D position (X, Y, Z)
  - Velocity in all 3D axis (VX, VY, VZ)

  All in a row with their corresponding time stamp.
- *state_ Crazyflie_ group00.dat*: Same content with the previous file, but as a binary file. It is encoded with Pickle (a Python library).
- *state_ Crazyflie_ group01.csv*: This file is a text-based table including CrazyFlie's secondary state estimations like:
  - 3D Euler orientation (roll, pitch, yaw)
  - In some cases, other parameters like battery voltage

  All in a row with their corresponding time stamp.
- *state_ Crazyflie_ group01.dat*: Same content with the previous file, but as a binary file. It is encoded with Pickle (a Python library).
- *state_ ToF.csv*: This file is a text-based table including ToF data. Each 65 row is a different frame. First row, first column element is time stamp. other 64 rows, are made of (in order):

  - Distance
  - Number of targets
  - Pixel status

  Each corresponds to ToF 64 pixels.

- *state_ ToF.dat*: Same content with previous file, but as a binary file. It is encoded with Pickle (a Python library).
- *state_ Vicon.csv*: This file is a text-based table includes Vicon positioning data. Each row consists of:
  - Object name
  - Time stamp
  - 3D position (posX, posY, posZ)
  - 3D quaternions orientation (qW, qX, qY, qZ)
- *state_ Vicon.dat*: Same content with previous file, but as a binary file. It is encoded with Pickle (a Python library).

Table D.1.: ToF data-set content in detail

Approach	
	foward backward .75
	Move forward and crash
	move forward_backward vx = .5
	move forward, rotate, backward vx =.75
	vx .75 forward
	vx 1 forward backward not good
	vx 1 forw back
	vx 1 forw back 2
	vx = .75
	vx = .75 forward backward not good
Rotate	
	rotate yawr 50
	rotate yawr 70
	rotate yawr 90
Object movement	
	Human Movement
	apppreoach narrow hole
	avoid big object stable
	Avoid Object
	Avoid Object 2
	Avoid Object 3
	Avoid Object_not complete3
	Avoid Object_not complete10
	Corridor test
	cross hole
	cross hole 2
	cross hole 3
	cross hole with narrow bands
	cross narrow door
	cross the open door
	Height increase with object in front
	Height increase with object in front 2
	Human Movement
	lab room survay
	move in y direction to see 2 object
	move through two object
	object avoid angle
	object avoid surface
	object avoid two object
	object avoid with vicon
	route and object avoid on top
	route with obstacle
	route 1
	route 2
	simple take off and land

Bibliography

[1] E. Balestrieri, P. Daponte, L. De Vito, F. Picariello, and I. Tudosa, "Guidelines for an unmanned aerial vehicle-based measurement instrument design," *IEEE Instrumentation & Measurement Magazine*, vol. 24, no. 4, pp. 89–95, 2021.

[2] J. Matos-Carvalho, R. Santos, S. Tomic, and M. Beko, "Gtrs-based algorithm for uav navigation in indoor environments employing range measurements and odometry," *IEEE Access*, vol. 9, pp. 89 120–89 132, 2021.

[3] T. Lee, S. Mckeever, and J. Courtney, "Flying free: A research overview of deep learning in drone navigation autonomy," *Drones*, vol. 5, no. 2, p. 52, 2021.

[4] V. Iyer, A. Najafi, J. James, S. Fuller, and S. Gollakota, "Wireless steerable vision for live insects and insect-scale robots," *Science robotics*, vol. 5, no. 44, p. eabb0839, 2020.

[5] V. Niculescu, L. Lamberti, F. Conti, L. Benini, and D. Palossi, "Improving autonomous nano-drones performance via automated end-to-end optimization and deployment of dnns," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2021.

[6] K. McGuire, C. De Wagter, K. Tuyls, H. Kappen, and G. C. de Croon, "Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment," *Science Robotics*, vol. 4, no. 35, 2019.

[7] Y. Fu, D. Tian, X. Duan, J. Zhou, P. Lang, C. Lin, and X. You, "A camera–radar fusion method based on edge computing," in *2020 IEEE International Conference on Edge Computing (EDGE)*, 2020, pp. 9–14.

[8] M. T. Lázaro, L. M. Paz, P. Piniés, J. A. Castellanos, and G. Grisetti, "Multi-robot slam using condensed measurements," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 1069–1076.

[9] W. Zhao, A. Goudar, J. Panerati, and A. P. Schoellig, "Learning-based bias correction for ultra-wideband localization of resource-constrained mobile robots," *arXiv preprint arXiv:2003.09371*, 2020.

[10] Z. Wei, F. Zhang, S. Chang, Y. Liu, H. Wu, and Z. Feng, "Mmwave radar and vision fusion based object detection for autonomous driving: A survey," *arXiv preprint arXiv:2108.03004*, 2021.

[11] A. Pandya, A. Jha, and L. R. Cenkeramaddi, "A velocity estimation technique for a monocular camera using mmwave fmcw radars," *Electronics*, vol. 10, no. 19, p. 2397, 2021.