ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

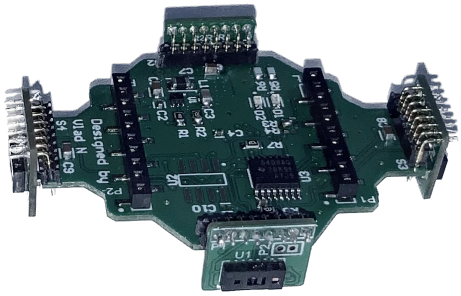
CENTER FOR PROJECT-BASED LEARNING

Department of Information Technology and Electrical Engineering

Spring Semester 2023

Nano-Drone autonomous navigation challenge: try to catch me!

Bachelor Thesis



Lukas von Briel
lvonbriel@ethz.ch

June 2023

Supervisors: Mr. Vlad Niculescu, vladn@iis.ee.ethz.ch
Dr. Tommaso Polonelli, tommaso.polonelli@pbl.ee.ethz.ch
Professor: Prof. Dr. L. Benini, lbenini@iis.ethz.ch

Acknowledgements

I would like to thank my supervisors Vald Niculesu and Dr Tommaso Polonelli for guiding me through this thesis. The whole process from getting to know the system over implementation until testing was very well organised and structured. From general design ideas to very concrete technical support, I always got a clever input whenever I asked for it. Furthermore, I want to thank Prof. Dr. Luca Benini to make it possible for me to work on such a great challenge.

Abstract

Nano unmanned aerial vehicles (UAVs) have gained significant attention due to their potential applications in various fields, including surveillance, search and rescue operations. However, the limited size and computational power of nano UAVs pose challenges for achieving efficient obstacle avoidance.

Previous studies have demonstrated the effectiveness of utilizing time of flight (ToF) sensors for implementing obstacle avoidance with low computational and electrical power requirements. However, so far only an one-directional ToF sensor has been used.

This bachelor thesis focuses on developing a moving obstacle avoidance algorithm on nano UAVs by utilizing four ToF sensors, each with an 8x8 pixel distance matrix and a 45 degree field of view (FoV), monitoring all four directions.

Firstly, the algorithm achieved a quite high robustness and secondly, it tackles the issues of 45° blind spots between sensors by focusing on the nearest moving object. In an empty indoor environment, it escaped a moving object with mean speed of 1m/s for 120s in 98% of the time. By addressing the challenges of navigating real-life environments, this research contributes to the overall development of efficient and robust nano UAV systems, by laying a foundation of basic moving obstacle avoidance in all directions.

Declaration of Originality

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. For a detailed version of the declaration of originality, please refer to Appendix B.

Lukas von Briel,
Zurich, 10 Juni 2023

Contents

List of Figures

List of Tables

Acronyms

AI artificial intelligence.

CNN convolutional neural network.

ETH eidgenössische technische hochschule.

FoV field of view.

I2C inter integrated circuit.

IEEE institute of electrical and electronics engineers.

PCB printed circuit board.

PID proportional integral derivative.

poc probability of collision.

SLAM simultaneous localization and mapping.

ToF time of flight.

UAV unmanned aerial vehicle.

ULP ultra-low-power.

VFH vector field histogram.

Chapter 1

Introduction

1.1. Motivation

Nano drones have gained significant attention due to their potential applications in crowded environments. They can not cause any injuries to humans because they are leightweight ($\sim$ 50 g) and low power devices. However, the limited size and computational power of nano drones makes it difficult to run advanced algorithms for mapping such as simultaneous localization and mapping (SLAM) or deep neural networks like convolutional neural networks (CNNs) for image classification. This poses a lot of challenges for achieving efficient and robust obstacle avoidance.
One way to tackle this problem are the novel VL53L5CX 8x8 Multi-Zone Time-of-Flight Sensors from STMicroelectronics. With their small size, leightweight and low power consumption, they fit perfectly for providing data to a basic obstacle avoidance algorithm.
This bachelor thesis focuses on developing this just mentioned real-time obstacle avoidance algorithm by utilizing four ToF sensors, each with a 45 degree FoV, covering all directions. Notably, there is a 45 degree blind spot between each sensor, which necessitates careful consideration in designing effective avoidance strategies.
Furthermore, the implications of this research extend beyond obstacle avoidance. The findings and methodologies presented in this thesis serve as a foundation for future advancements in nano drone technologies. They lay the groundwork for integrating moving obstacle avoidance capabilities with other advanced projects such as path planning algorithms and artificial intelligence (AI)-based object detection systems. The significance of this research lies in its contribution to overcoming the challenges of operating nano drones in real-life environments, where reliable and efficient obstacle avoidance is crucial.

1.2. Objective

1.2.1. Personal contribution

This thesis is based on the hardware implementation of the paper "Towards a Multi-Pixel Time-of-Flight Indoor Navigation System for Nano-Drone Applications" by the institute of electrical and electronics engineers (IEEE) at eidgenössische technische hochschule (ETH) Zürich. [1]
However, instead of only one ToF sensor, the nano UAV used here has four sensors.
This means that hardware, namely the nano UAV, as well as the ability to read in time of flight data via a inter integrated circuit (I2C) communication have already been designed and implemented by the just mentioned laboratory. The thesis only focuses on the implementation of the control policy, which has ToF Data as input and directional flying commands as output.

1.2.2. Research Goals

- main research goal:

  Design a control policy to autonomously fly the Crazyflie 2.1 in challenging situations with high-speed moving obstacles, relying only on onboard sensors and computational resources.

- sub-goals:

  The main research goal was divided in sub-goals, which have to be individually solved in order to achieve the overall goal.

  1. Detect a moving object approaching the drone.
  2. Try to focus on the moving object with one ToF sensor on the drone to cover up for blind spots.
  3. Choose the direction to fly to in case of a potential collision.

1.2.3. Outline

To set a baseline to what kind of UAV, sensors and algorithms this bachelor thesis is based on, each of them is explained in detail in Chapter 2. In Chapter 3 related work is being analysed and set into context. As the used sensors were released in Q2 of 2021 by STMicroelectronics, they are quite new and therefore, not much practical research has been done so far. Chapter 4 explains how the sensor data is processed in order to extract information about nearest moving objects and avoidance trajectories. To achieve quantitative results several indoor flying scenarios have been run through and are described in Chapter 5. Robustness, computational speed as well as the approaching

speed are derived to put the results in a measurable context. Finally, in Chapter 6, conclusions from the results are drawn and summarized to evaluate the effectiveness of the algorithm. Moreover, further ideas and improvements are pointed out.
In the Appendix, further information such as the detailed task description (Chapter A), the declaration of originality (Chapter B), as well as the github file structures (Chapter C) can be found.

Chapter 2

Theory

This chapter describes and categorizes the hardware used in the project. It also provides a short description of two basic real-time obstacle avoidance algorithms.

2.1. Nano-UAVs

Figure 2.1.: Crazyflie 2.1 (Image source: Foto)

The Crazyflie 2.1 (Figure 2.1) is a small (80 mm in diameter), lightweight (27 g) and agile drone developed by Bitcraze.[3] It thereby categorizes as a nano-drone, which means it is smaller than 10 cm and weighs less than 50 g.
The most important feature is its open-source nature, meaning its hardware and software are easily accessible and customizable. These features make it easy for developers to modify and extend the functionality according to their specific needs, for example by so called expansion decks. These are printed circuit boards (PCBs) that can be mounted on the drone and can thereby add additional sensors.
However, there is a downside resulting from the lightweight design: The maximum recommended payload is restricted to 15 g, which excludes many sensors by default.

2.2. Time of Flight Sensor

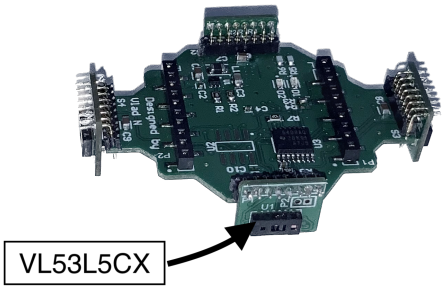


Figure 2.2.: ToF expansion Deck designed by Vlad Niculescu [1] (Image source: Foto)

Luckily, in Q2 of 2021 STMicroelectronics released their novel multi-zone ToF sensor VL53L5CX, which is the first miniaturized ToF sensor that features an 8x8 image resolution.[4] It has been incorporated four times into a expansion board which can be seen in figure 2.2. Due to its small size (6.4 x 3 x 1.5 mm) and its low power consumption even in continuous mode (313 mW), it is compatible with the light weight of the Crazyflie 2.1. The VL53L5CX has FoV of 45° vertical as well as horizontal. It can measure the distance reliably up to 3 m.

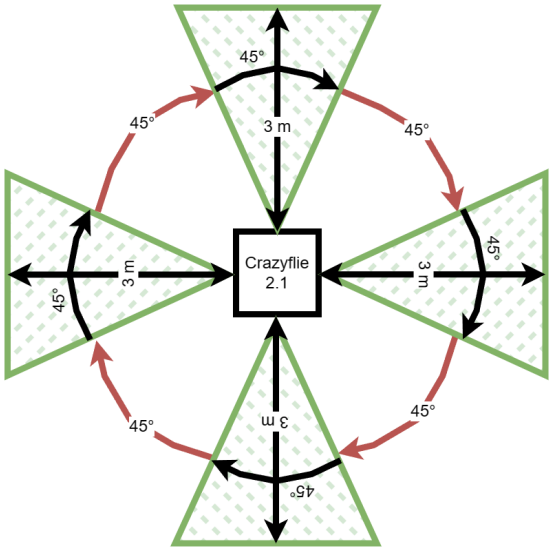


Figure 2.3.: 2D Depiction of ToF coverage (Image source: draw.io); Green: area covered by ToF sensor / Red: blind spots between ToF sensors

Consequently, when mounting the VL53L5CX sensors on the ToF Expansion Deck, we get a sensor coverage like depicted in figure 2.3. To sum it up, the sensor covers an area of

$36\,m^2$ with however 45 degree blind spots between each of them. This is very important as it contributes massively to the algorithm design described in Chapter 4.

2.3. Real-Time Obstacle Avoidance Algorithms for Mobile Robots

The Control policy has been fully invented and implemented by myself and is not directly based on any other paper. However, it is connected to general avoidance concepts that have already been developed. The following ideas are both used in the algorithm explained in this thesis.

2.3.1. The Vector Field Histogram Algorithm

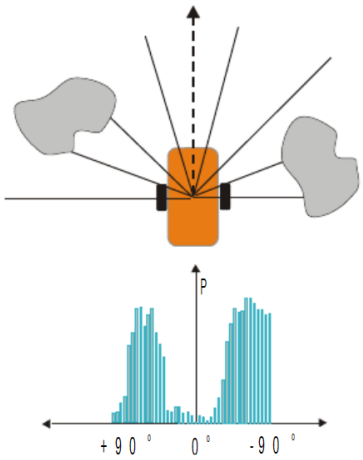


Figure 2.4.: The polar histogram used in VFH algorithm (Image source: [2])

As described in [2], the VFH algorithm creates a polar histogram of recent sensor readings as you can see in figure 2.4. The x-axis of the histogram shows the angles relative to the drone and y-Axis the probability of collision (poc). Above the histogram, you can see an autonomous vehicle facing two objects; one to the left at 45 degree and one to the right at 70 degree.
The resulting histogram is then used to identify passages large enough to allow the robot to pass through. The selection of the final path is computed by combining the robot's path to the goal and the identified open passages. Path planning is, however, not part of this thesis. That is why I will not go in more detail here.
The takeaway here is the idea of dividing the surrounding area in chunks that can have a poc assigned.

2.3.2. The Bubble Band Technique

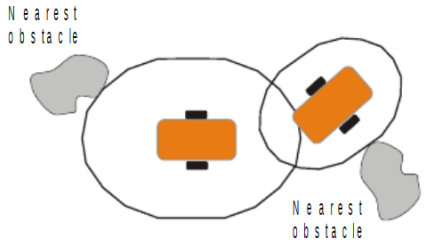


Figure 2.5.: Illustration of the bubble band concept (Image source: [2])

Another avoidance technique, also described in [2], is the Bubble Band Technique. In illustration 2.5, two different "bubbles" can be seen where each has the radius of the distance to the closest object. With this concept, a band of such bubbles can be used to plan a path to the goal.
This technique is not too much of a real-time planning algorithm, but the core concept of giving the minimum distance an important role in path planning, makes it important for the algorithm of this thesis.

Chapter 3

Related Work

This chapter describes the relevant aspects of related work. However, caused by the novelty of the previous described VL53L5CX sensor (Section 2.2), there are only very few closely related papers available.

The Crazyflie 2.1 has proven its versatile nature in various application. It can be combined with many different expansions decks such as the ToF deck explained in this thesis. Another way to measure the distance on autonomous vehicles lies in the use of mmWave radar sensors. [5] Relying on the Doppler effect, these sensors can even measure the speed of incoming objects and also work in bad lighting or complete darkness, such as the ToF sensors do. In order to use the radar sensors for object detection, the mainstream approach is to use any kind of vision based fusion mostly with the help of machine learning. [5] [6]
However, it is hard to deploy such networks on nano UAVs like the Crazyflie 2.1. On approach has been made by the paper [7] , which uses an energy-efficient ultra-low-power (ULP) parallel processor called GAP8, which is mounted on a separate expansions deck called the AI-Deck. This AI-Deck makes it possible to run small and optimized neural networks in real-time on nano UAVs with a classification accuracy of 90% and only 4 mJ/frame energy consumption. [7]
Instead of using radar sensor, recent papers try to include laser sensors for moving object detection, path planning as well as mapping. [8] [9] [10] [1] [11]. In [8], they are using the multi-ranger expansion deck, which has four tiny laser rangers that point to front, left, right and back. It is used in the context of autonomous exploration and homing. However, the low resolution and poor spatial covering of the laser sensors still demand camera data to make their approach work.
Because high resolution camera data is difficult to process on low power devices like the nano-drones, [10] and [1] tried to integrate a novel 8x8 image resolution ToF sensor from STMicroelectronics (VL53L5CX). This should enable the drone to perform obstacle avoidance and path-planning without vision-based data opposite to [8], [9] and [11].

[10] and [1] used one of the VL53L5CX, which faced in flying direction. From the incoming data, they computed the approach angle to walls and developed a basic obstacle avoidance algorithm. The algorithm uses object clustering by distance in the 8x8 matrix received from the sensor. These objects are then categorized depending on their distance into ceiling, floor, object in caution zone and object in danger zone, which is followed up by steering left or right around the object or even go backwards. Thereby, they could avoid dynamic objects with drone flying speeds up to 2 m/s. However, [10] had not included the possibility of avoiding objects coming from behind, left or right of the drone. Because of a default sideways drift of the Crazyflie 2.1, which made flying close to walls more dangerous, on of the autors of [10] designed a new expansion deck with 4 of the VL53L5CX sensors facing front, left, right and back.
Based on this deck (already described in section2.2), this thesis tries to implement a real-time obstacle avoidance control policy.

The following table shows a comparison between this thesis and other related work. It can be seen that the omnidirectional ToF sensing is the unique selling point of this thesis and thereby the biggest novelty.

Topic	[1]	[10]	[8]	[9]	[2]	[11]	This Thesis
Usage of Crazyflie 2.1	✓	✓	✓	-	-	✓	✓
Real-time obstacle avoidance algorithm	-	✓	✓	✓	✓	-	✓
Usage of VL53L5CX ToF Sensor	✓	✓	-	-	-	✓	✓
Usage of advanced methods (CNNs/SLAM)	-	-	-	-	-	✓	-
Omnidirectional ToF sensing	-	-	-	-	-	-	✓
Autonomous (fully onboard computation)	-	✓	-	-	-	-	✓

Table 3.1.: Comparison between this thesis and related work.

Chapter 4

Implementation of Real-Time Obstacle Avoidance Control Policy

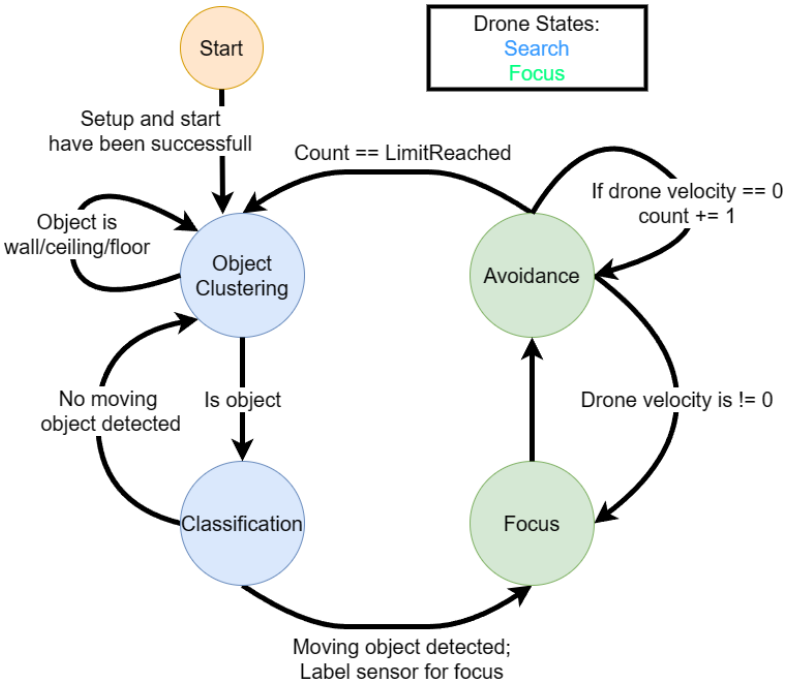


Figure 4.1.: StateMachine of Real-Time Obstacle Avoidance Control Policy (Image source: draw.io)

As it can be seen in figure 4.1, the control policy is divided in two states namely "search" and "focus". When starting the algorithm, the drone hovers at a certain height above the ground and is in the search-state. From here on, it performs object detection in all four

directions and classifies the resulting objects on manually crafted features whether if it is moving or not. This is an important step as moving objects have a higher probability of collision with the drone than not moving ones.
This part is so important because of the already mentioned blind spots in Chapter2 between the sensors. To cover up for the blind spots, the drone is constantly rotating around its own axis in search-state to stay updated of the full 360° around it.
Once a moving object has been detected in one of the sensor's vision, the drone state changes to "focus". In the focus-state, the sensor facing the moving object is now labeled as moving object sensor and the drone tries to keep focus on the object within that sensor. This is done by calculating the center column of the object and trying to align it with the center of the ToF sensor by the help of a simple PID controller, which acts on the yaw rate.
While regulating the yaw rate in such a way that the moving object is always in sight of a sensor, a basic avoidance algorithm tries to steer the drone in the most open direction by regulating forward and sideways velocity accordingly.
The combination of the two independent controllers "focus" and "avoidance" bares some problems in the stability of the drone. This is because having to many flying inputs can destabilize the drone and make it crash, which has been compensated by intensive tuning of each of the parameters of the controllers.
However, there are also two big advantages resulting from this approach. First of all, it copes with the problem of blind spots, because the moving object always stays insight of a sensor. Consequently, it is not possible to approach the drone from a blind angle. Second, it is possible to use different factors when converting distance in probability of collision. To illustrate that, it is helpful to think of an example: There is a moving obstacle 2 m in front of the drone and a wall 2 m behind. If the drone was not able to distinguish between wall and moving object, it would have no idea in which direction it is safe to fly and evaluate its position as optimal when staying right in middle. If the drone, however, knows where the moving obstacle is located, it can add an additional safety factor in that direction. This makes it possible to escape close to walls or static objects without risking any moving object getting to close, which is the research goal of this thesis (Section 1.2).
If the drone has not detected any moving object for a couple of frames, it switches back to search-state and starts again with object clustering and classification while rotating.

4.1. Object Clustering

This section takes a closer look on how the Object Clustering is performed in search-state.

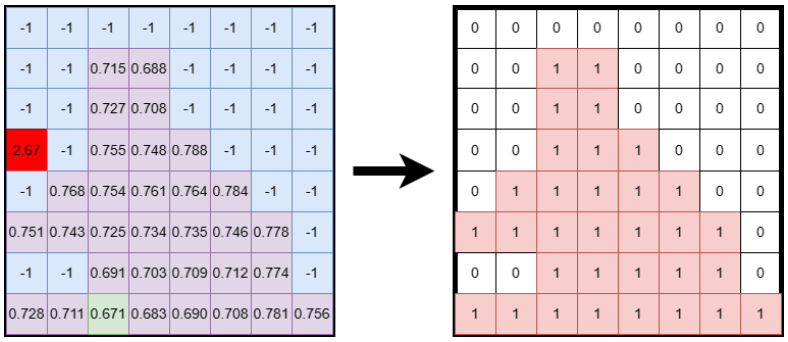


Figure 4.2.: Example of Object clustering: Person sitting 0.7 m in front of sensor. Distance values in [m] (Image source: draw.io)

As you can derive from in figure 4.2, the clustering is performed directly on the distance matrix, which can be seen on the left side. All pixels which invalid by the internal filtering of the ToF sensor have the value "-1". In this example, a person is sitting at a distance of 0.7 m in front of the sensor, which results in a distance matrix with many values close to 0.7 in the middle. The recursive algorithm, which can be seen below, starts off at the closest pixel (highlighted in green) and labels it as 1. Then it checks the neighbours recursively until no connected pixel can be found anymore. The result can be seen on the right side of figure 4.2. Especially for small objects, the recursive algorithm works faster than iterating through the whole matrix.

```
    """
    For a given starting pixel (i,j), Recursive Cluster finds all pixel
    that are connected to the starting pixel by roughly the same distance
    -threshold is the upper bound for the distance
    -solution is the matrix that is filled with 1s for all pixels that are part
        of the object
    -grid is the matrix that contains the distance values
    -(i,j) is the starting pixel for the recursive search it determined by the
        closest pixel
    """
    def RecursiveCluster(grid, solution, i, j, threshold):
        # check if pixel is out of bounds
        if i < 0 or i >= grid.shape[0] or j < 0 or j >= grid.shape[1]:
            return
        # check if pixel is already part of the object and if the distance is
            within the threshold
        if solution[i, j] == 0 and grid[i, j] < threshold:
            # mark pixel as part of the object
```

```
solution[i, j] = 1
# recursive call for all 4 neighbors
RecursiveCluster(grid, solution, i + 1, j, threshold)
RecursiveCluster(grid, solution, i - 1, j, threshold)
RecursiveCluster(grid, solution, i, j + 1, threshold)
RecursiveCluster(grid, solution, i, j - 1, threshold)
```

After clustering, features are extracted to distinguish between wall/ceiling/floor and potentially moving object.

4.1.1. Object Center

One of the most important features is the object center. In order to get the center, the center column and center row is computed. To get the center column/row, the algorithm iterates through the columns/rows and calculates:

int leftMostCol/Row = "Left most column/row which contains >=2 object pixels"
int rightMostCol/Row = "Right most column/row which contains >=2 object pixels"

$$return \quad getOuterCenter(\frac{leftMostCol/Row + rightMostCol/Row}{2}) \tag{4.1}$$

The getOuterCenter function handles the possibility of an uneven column/row and returns the outer value, meaning all values smaller 4 are getting rounded off and lager ones are getting rounded up. This will improve the reaction of the focus controller as the center column is thereby pushed outwards, which causes a faster reaction.
If the determined center row is the bottom/top row, the object is labeled as floor/ceiling.

4.1.2. Object Size

Another feature is the object size. This is just the sum of all as 1 labeled pixels. The object size in figure 4.2, for example, would be 32. If the object size is bigger than 90% of the ToF matrix size, the object is considered to be a wall.

4.1.3. Approaching Angle

Like already mentioned in the beginning of this chapter, during search-state, the drone needs to constantly rotate around its own axis to cover up for the blind spots. This rotation, however, makes the moving object detection more difficult because the drone's

movement has to be kept in mind when computing the speed or position of other objects.

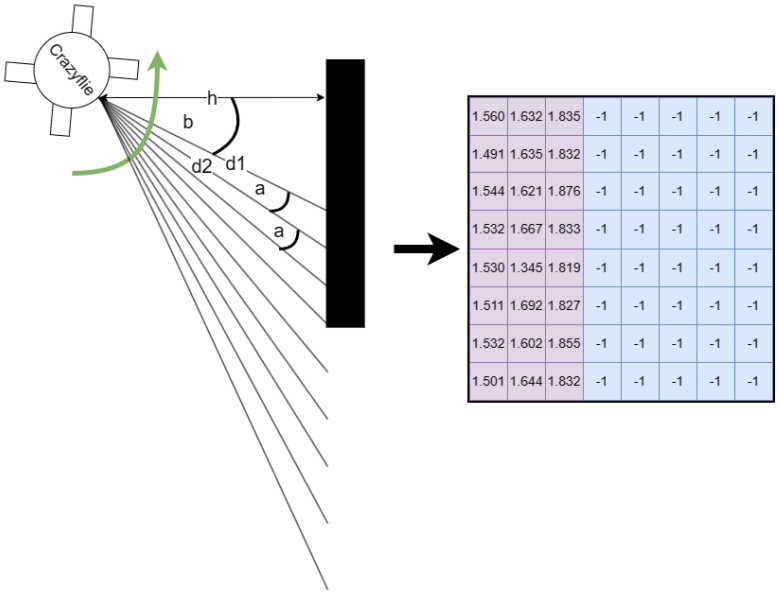


Figure 4.3.: Example of a wall sliding into the range of ToF sensor (Source: draw.io)

In figure 4.3 on the left, you can see the top down perspective of a Crazyflie drone close to a wall. The lines coming out of the drone and partially end at the wall represent the vision of the ToF sensor. The vertical distance to the wall is labeled with "h", the angle between the first ToF column and the vertical line is called "b", the angles between each column are "a" wide and the distance measured by the ToF sensor go from d1 to d8 (left to right). On the right side, you can see a possible distance matrix the ToF sensor could produce, if it is positioned like on the left.
If the drone is now rotating in the direction of the green arrow it would seem like there is object coming closer on the left of this ToF matrix.
To solve this issue, the drone needs to be able to detect walls not only when they are facing from the front but also with an angle. Therefore, the approaching angle "b" is calculated like the following:

$$h = d1 \cdot \cos(b) \tag{4.2}$$

$$h = d2 \cdot \cos(b + a) \tag{4.3}$$

These two equations are combined to get:

$$\cos(b + a) = \frac{d1}{d2} \cdot \cos(b) \tag{4.4}$$

When solving this equation for b with $c = \frac{d1}{d2}$ we get:

$$b = \arccos\left(\frac{\sin(a)}{\sqrt{-2c\cos(a) + c^2 + 1}}\right) \tag{4.5}$$

$$h = d1 \cdot \cos(b) \tag{4.6}$$

If there is an angeled wall insight of the ToF sensor the third column mostly has a distance of:

$$d3 = \frac{h}{cos(b + 2a)} \tag{4.7}$$

To sum it up, when the closest pixel is on the left or right edge of the ToF matrix, the approaching angle is calculated with the information of the first two columns. Then the distance the third column should have is computed. Finally, the measured distance and the calculated distance of the third column are compared to determine if the drone sees a wall.

4.2. Moving Object Classification

The next step in the control policy pipeline is the classification process. This step is also performed in search-state. If the potential object has been classified as an object by the clustering algorithm it is passed to the moving classification.
To determine whether, an object has moved during a period of time, the x and y position of the object to the drone is stored in a queue. Each ToF sensor has its own queue. The x and y coordinates are calculated as follows:

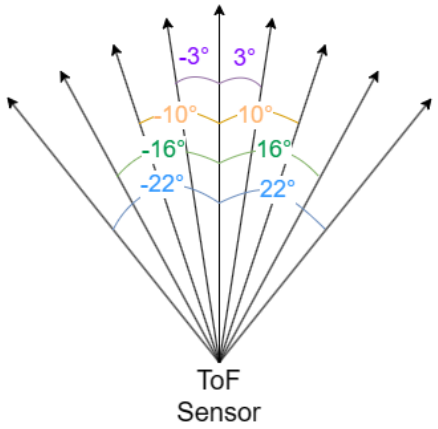


Figure 4.4.: Illustration of the "columnAngle" for each column of the ToF matrix (Source: draw.io)

Sensor	ToF Front	ToF Back	ToF Left	ToF Right
Sensor Angle	0 degree	180 degree	90 degree	270 degree

Table 4.1.: Relation of sensors and their Angle

$$x = \cos(droneYaw + sensorAngle + columnAngle) \cdot objectDistance \tag{4.8}$$
$$y = \sin(droneYaw + sensorAngle + columnAngle) \cdot objectDistance \tag{4.9}$$

As it can be seen in equation 4.8 and 4.9 first the angle of the object center relative to the drone's zero angle is computed. It consists of the droneYaw, which is the output of the internal yaw measurement of the Crazyflie firmware. Then the sensor angle is added, which depends on the ToF sensor currently selected, as it can be seen in table 4.1. Lastly, the angle within the ToF sensor is added (figure 4.4). To get the x-value/y-value, the cosinus/sinus is calculated and multiplied with the object's distance.

To get a better understanding of how this works, let's have a look at figure 4.5. In this figure, the drone's front (indicated with a triangle) points to the top right. Additionally,

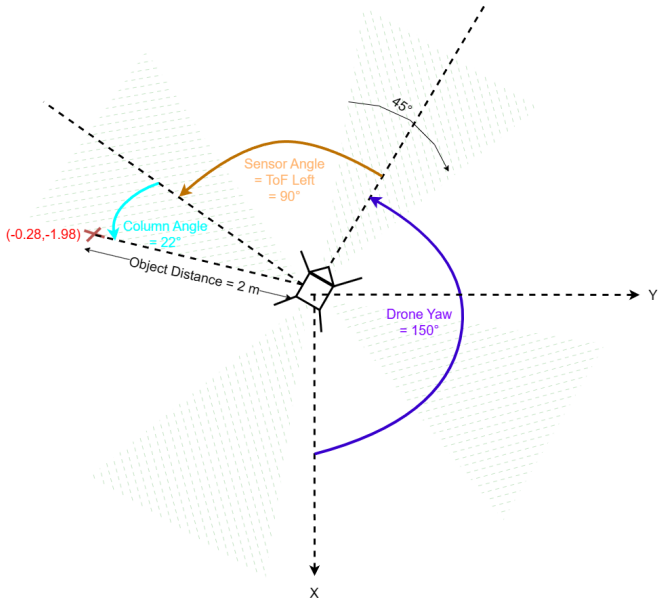


Figure 4.5.: Example of how a point is mapped (Source: draw.io)

the coordinate system is displayed with a downwards facing x-axis and a rightwards facing y-axis. Let's assume that an object has been detected on the left ToF sensor indicated with a red cross in figure 4.5. By adding up the highlighted angles and using the equations 4.8 and 4.9, x and y-coordinates are received. These are then stored in queue.
If a queue is full, the mean distance between the stored objects is calculated and combined with the frame rate, a mean speed is computed. If the mean speed exceeds a threshold, the according ToF sensor is labeled as moving object sensor and the drone state changes to focus-state.

4.3. Focus

After a moving object has been detected in the sight of a ToF sensor, the yaw angle is regulated in such a way that the center column of the object is always in the middle of the received ToF matrix.

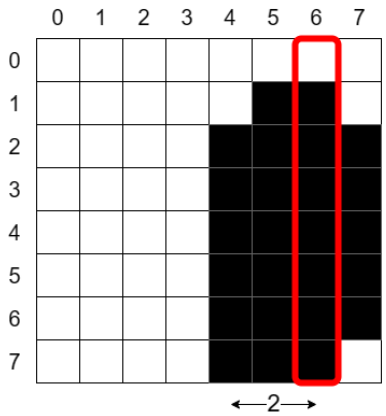


Figure 4.6.: Example of clustering with offset center column highlighted in red. Object is marked in black. (Source: draw.io)

This is done by using first the objects clustering like explained in section 4.1, where the object is clustered and the center column is computed (Example in figure 4.6). Second, the PID control sets the yaw rate to move the center column of the object towards the center column of the matrix (column 3 or 4), resulting ideally in the matrix in figure 4.7 when the controller has converged.

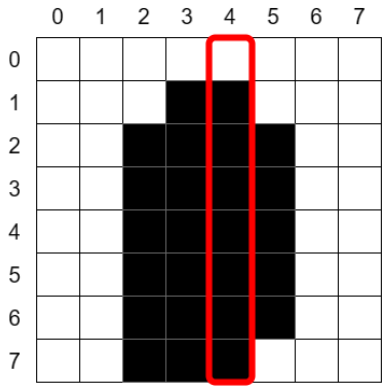


Figure 4.7.: Example of clustering with converged center column highlighted in red. Object is marked in black. (Source: draw.io)

4.4. Basic Avoidance

The heart of the real-time moving obstacle avoidance control policy is the basic avoidance part. It sets the forward and sideways velocity of the drone to outmaneuver nearby obstacles.

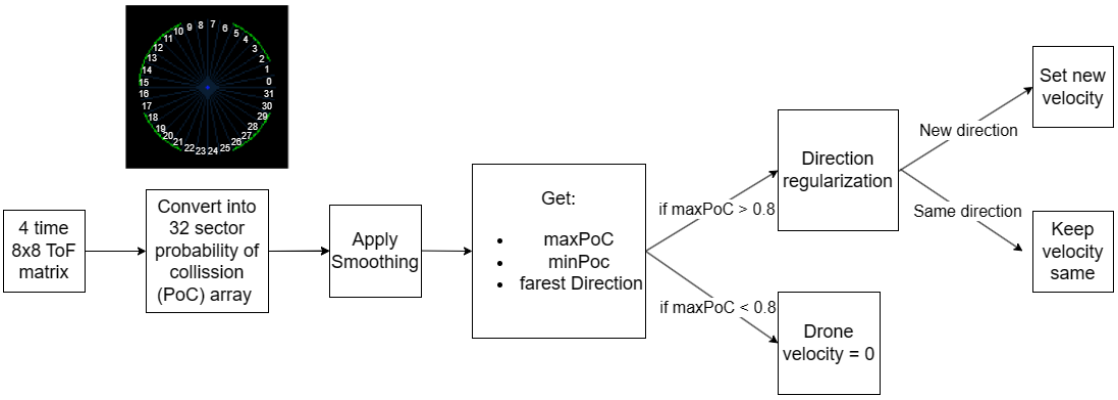


Figure 4.8.: Decision pipeline of avoidance control policy. (Source: draw.io)

The decision pipeline depicted in figure 4.8 works like following.
It starts off when all four ToF sensors have received a new measurement.
Then, the distances are converted into a probability of collision (PoC). This is done by calculating the poc of every column individually. To get the poc of a single column, the mean distance of all values as well as the minimum distance is calculated. After that, each distance is converted into a poc by applying one of the following functions, where "d" stands for distance:

$$f(x) = 1 - \frac{1}{9} \cdot d^2 \qquad if: -2.5 < d < 2.5 \tag{4.10}$$

$$f(x) = 0 \qquad else \tag{4.11}$$

$$f(x) = 1 - \frac{1}{4} \cdot d^2 \qquad if: -2 < d < 2 \tag{4.12}$$

$$f(x) = 0 \qquad else \tag{4.13}$$

The first function (4.10) is used if the ToF sensor currently has a moving object in sight. If that is not the case, the second function (4.12) is used. Both of these functions are displayed in figure 4.9. This differentiation makes it possible for the drone to fly close to walls but still keep a larger distance to the moving object.
After these functions have been applied, we now have two poc values. First, we got the

minimum distance poc value and second we got the mean distance poc value. These values are combined in a weighted average. This results to: "column poc value = 80%· minimum distance poc + 20%· mean distance poc".
After the conversion has happened, each column poc value is stored at its respective position. This position depends on which angle the individual column was facing relative to the front of the drone. Then this angle is discretized into 32 segments to improve the computation speed (4.8). In the following the column poc value is assigned to the according segment. If the segment is in a blind angle, it is assigned with the maximum poc value of one. The blind spots are not left out because they provide an easy way of including potential mapping data. The blind spot segments could filled up with previous mapping of the environment.

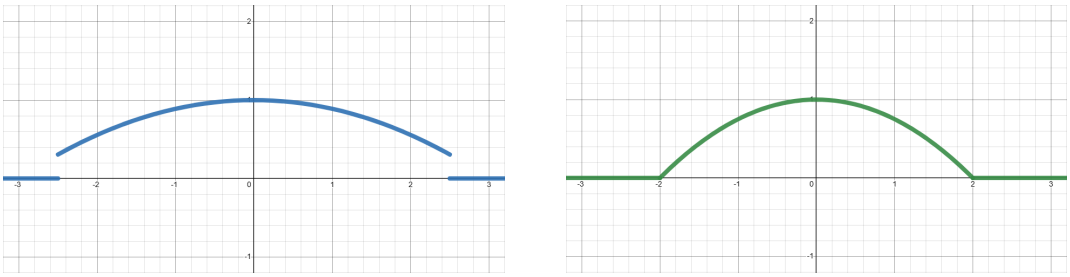


Figure 4.9.: Illustration of functions. Left: 4.10, Right: 4.12 (Source: desmos.com)

Once all segments have a value, smoothing is applied to the segments array. To smooth the array, a windows slides over the array to smooth values relative to their neighbours. Inside of these windows, the maximum poc value is determined. Then the maximum value is averaged with the current value to get the smoothed value, which is then inserted into the array.
An example can be see in figure 4.10. The current value is at position 2 with a poc value of 0.74, highlighted in red. The window, which has size 5, covers the two values left and right of the middle one. Inside the window, the maximum value is 0.85. By taking the average of those two, we get a new smoothed value of 0.8, which is then inserted. This can be seen on the right of the figure.

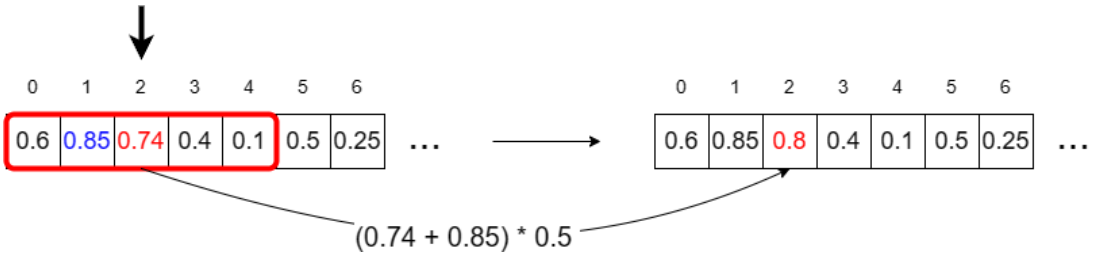


Figure 4.10.: Example of smoothing. (Source: draw.io)

Like shown in figure 4.8, after the poc array has been smoothed, three features are extracted from it. First, the maximum poc value is computed, which represents the nearest object to the drone. Second, the minimum poc value as well as the corresponding segment is computed, which represents the most open/clear area the drone can fly to.
Based on the maximum poc value, a decision is made whether the drone needs to change its current velocity. If there is an object coming to close, the drone sets a new velocity in the direction of the minimum poc value segment.
However, before actually setting the new direction, a regularization controller checks if the new direction is considerably better than the previous flying direction. This is a very important step, as it prevents oscillating between two very similar directions in terms of collision probability, which makes flying a lot more stable.

Chapter 5

Results

To evaluate the functionality of the system, different tests have been conducted. Firstly, each controller, which has been described in the previous chapter, has been tested individually. Secondly, full system tests in different environments have been done.

5.1. Moving Object Detection

First, the moving obstacle detection has been tested. The setup is displayed in figure 5.1. During the test, the drone is hovering at a constant position and rotates with a yaw rate of 25 degree/sec. Then a person approaches the drone at random moments. Each time the drone detects the incoming moving object the run is labeled as success. If the moving object crashed into the drone, the run is labeled as failure.

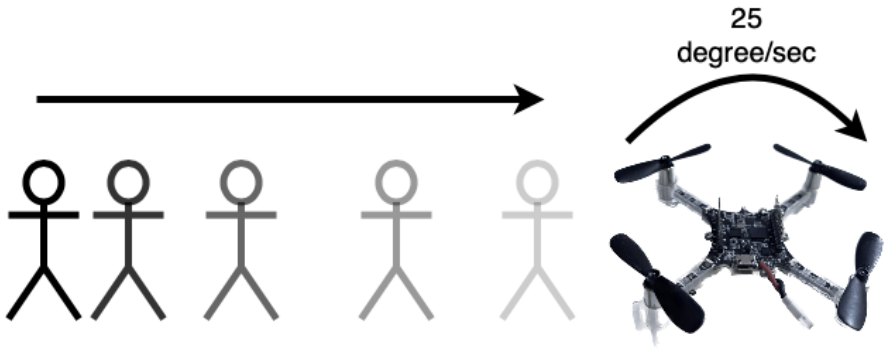


Figure 5.1.: Experimental setup for Moving Object Detection Test(Image source: draw.io)

5.2. Results of Moving Object Detection

The results of the just described setup can be quickly summarized. Out of 10 tries, the drone detected the moving object 8 times and crashed 2 times, which results in a success rate of 80%.

5.3. Focus

Next, the Focus and thereby the yaw rate PID controller was tested. As it can be seen in figure 5.2, a moving object rotates around the drone at a fixed distance. If the drone is able to keep the focus on the moving object by adjusting the yaw rate, the run is labeled as success. If it looses track, the run is labeled as failure. The walking speed which has been used during the test is about $2\,\frac{\mathrm{m}}{\mathrm{s}}$ and also fast direction changes have been included.

Figure 5.2.: Experimental Setup for Focus Test (Image source: draw.io)

5.4. Results of Focus

The results of the focus test are summarized in table 5.1.

Four different distances have been measured. It can be seen that at 3 m distance, there is 0% success rate. This is, however, caused by the ranging distance of the ToF sensor, which is also 3 m. Being so close to the maximum distance makes it impossible for the drone to even detect an object, let alone maintain the focus. For distance between 1 m and 3 m the focus works perfectly. When getting closer than half a meter, the success rate drops again. However, this is not caused by the implementation of the focus algorithm. Infact it is an issue of the internal yaw controller, which cannot handle fast changing yaw rates.

Table 5.1.: Focus test results.

Distance in [m]	$\frac{keeps\ focus}{n\ test}$ = success rate
3	$\frac{0}{5} = 0\%$
2	$\frac{5}{5} = 100\%$
1	$\frac{5}{5} = 100\%$
0,5	$\frac{1}{5} = 20\%$

5.5. Basic Avoidance

Also the basic avoidance controller has been tested individually. Therefore, three different setups have been used, which are displayed in figure 5.2. The avoidance path is indicated with a green arrow.

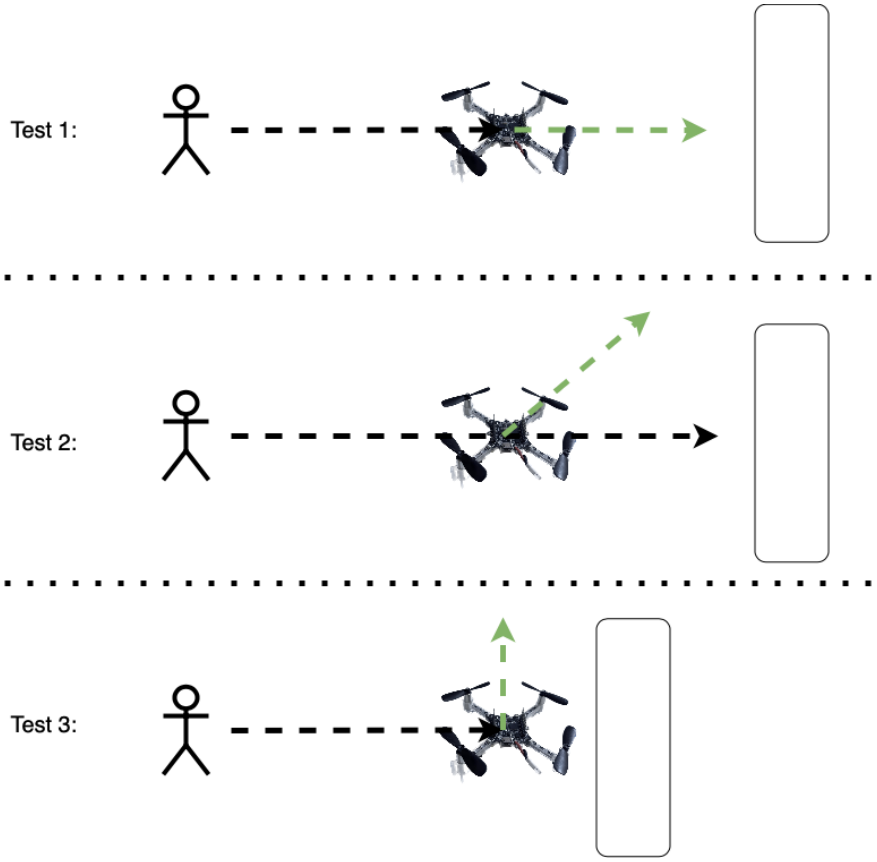


Figure 5.3.: Experimental Setup for Basic Avoidance Test (Image source: draw.io)

In the first test, the moving object approaches the drone until it reaches the drones starting position and stops there. The drone itself avoids the incoming object by moving backwards.
The second test has the same setup, however, the moving object does not stop at the drone position but keeps going all the way until a wall instead. This forces the drone to perform a more complex avoidance maneuver than in the first test by going back and to the left or right simultaneously.
For the last test, the drone is already hovering closely in front of a wall. This makes the drone move left or right only to escape the incoming object.

5.6. Results of Basic Avoidance

The results of the avoidance tests are displayed in figure 5.2. It can be seen that avoidance tests one and two have quite similar results. Up to a speed of about 2.2 m/s, the drone avoids the incoming moving object every time. For approaching speeds higher than 2.5 m/s the, drone starts crashing. The third test, however, does not perform quite as well as the others. Already at 1.9 m/s the drone starts crashing. Additionally, there is also an outlier at 1.6 m/s. This shows that with more complex avoidance paths, the drone gets more and more unstable.

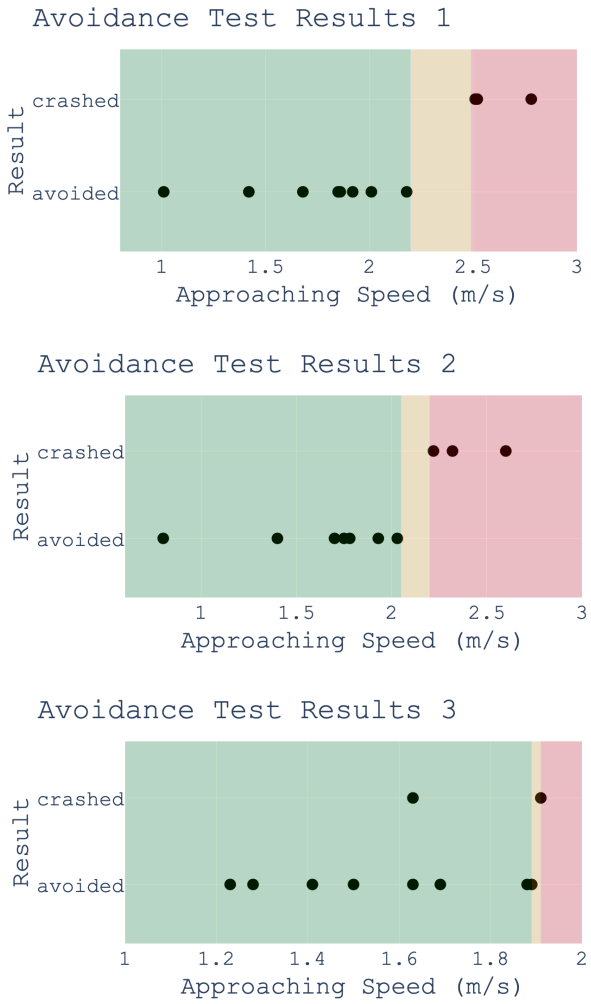


Figure 5.4.: Basic Avoidance Test Results.(Source: draw.io)

5.7. Full system in Open Space

The first full system test has been conducted in an indoor 7x7 m open space area without any additional obstacles except one person (figure 5.2). The person tries to chase the drone for 2 min. If the drone crashes, the flight is over and the time is stopped. The walking speed of the person is about 1.5 m/s in average. During this test, the complete control policy is activated, which means the drone starts in search-state and if the moving object is detected, it switches to focus-state.

Table 5.2.: Experimental Setup for Full System Test in Open Space

5.8. Results of Full System in Open Space

The just described test has been repeated 10 times. The flight time of each of those flights can be seen in table 5.3).
This results in a flight time of 19 min 30 sec out of a total possible flight time of 20 min. To have a meaningful evaluation metric, the success rate for the full system tests has been defined as $\frac{flight\ time}{total\ flight\ time}$ instead of only counting if the full 2 min are reached. By defining it this way also flights that did not reach the full flight time are considered.
Consequently, the success rate of this test is 97.5 %.

Table 5.3.: Open Space Test Results.

Test nr.	Flight Time
1	2 min
2	2 min
3	2 min
4	2 min
5	2 min
6	2 min
7	1 min 30 sec
8	2 min
9	2 min
10	2 min

5.9. Full system in Maze

To also evaluate the drones performance in a more difficult environment, a small maze has been built, which can be seen in figure 5.3). Apart from that, the test setup is the same as in section 5.7). Another important point is the controller used in this test. What came up during testing was that the avoidance part can also function on its own without the moving object detection and focus. Of course it lacks the blind spot compensation, which is, however, not a problem most of the time, when the approaching speed is slow. This is especially valid in narrow environments like this maze because with such close corners, it is not possible to keep someone in focus anyway with the ToF sensors only.

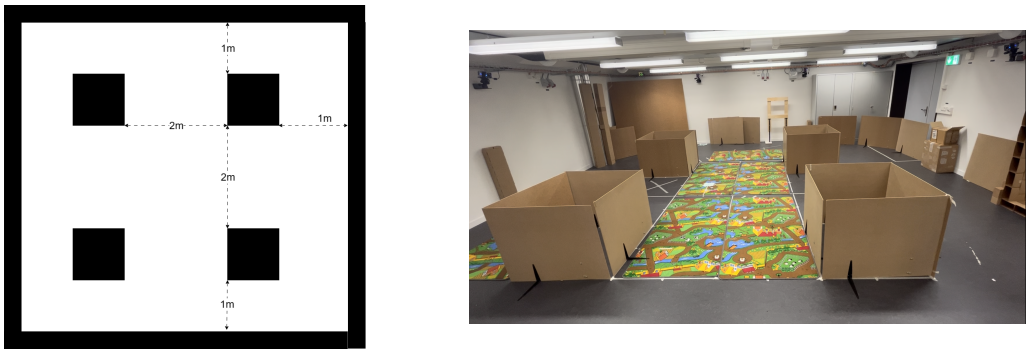


Figure 5.5.: Experimental Setup for Full System Test in Maze

5.10. Results of Full system in Maze

Like the open space tests, there have been also 10 flights of 2 min conducted. The results can be seen in table 5.4). When using the same definition of success rate as described in section 5.8), the success rate is 54.7 %.

Table 5.4.: Maze Test Results.

Test nr.	1	2	3	4	5	6	7	8	9	10
Flight Time	51 sec	2 min	1 min 30 sec	1 min 5 sec	50 sec	34 sec	1 min 35 sec	54 sec	1 min 9 sec	28 sec

5.11. Full system in Small Cage

Lastly, the system has been tested in small 3x3 m cage with only "artificial" cardboard walls to make the setup as ideal and clean as possible for the sensors and the drone itself. This setup is used to test the highest speed avoidance performance of the drone. Therefore, a person tried to catch the drone with an average walking speed of about 2.5 m/s and with peak walking speeds of 3 m/s, which is already close to a slow running speed. Like in the previous section, only the avoidance system is used again to get the maximum computation speed and thereby shortest reaction time of the drone. The setup can be seen in figure 5.5).

Table 5.5.: Experimental Setup for Full System Test in Small Cage

5.12. Results of Full system in Small Cage

The difference for this test compared to the other full system tests is that the flight time is only 1 min here. The results can be seen in table 5.6). When using the same definition of success rate as described in section 5.8), the success rate is 81.5 %.

Table 5.6.: Small Cage Test Results.

Test nr.	Flight Time
1	1 min
2	1 min
3	10 sec
4	34 sec
5	45 sec
6	1 min
7	1 min
8	55 sec
9	45 sec
10	1 min

Chapter 6

Discussion and Outlook

the goal of this thesis was to design a custom control policy to enable autonomous flight of the Crazyflie 2.1 nano drone in challenging situations with high-speed moving obstacles, relying solely on onboard sensors and computational resources. The objective was to develop a robust obstacle avoidance system that could effectively navigate real-life environments and pave the way for broader applications of nano drones.
To achieve this, a combination of four 8x8 multi-zone time of flight (ToF) sensors has been used, each with 45-degree field of view, covering all directions. The presence of 45-degree blind spots between each sensor necessitated careful consideration to ensure accurate obstacle detection and avoidance.
To evaluate if the thesis goals have been reached, I firstly evaluate step by step if every aspect of the goal is solved. Secondly, I compare the results to the paper [10], which is very closely related as explained in chapter 3.
The first part of the assignment reads: "design a custom control policy". Like explained in chapter 4, a custom control policy has been developed with various features like moving object detection, a PID focus controller and a fast reacting basic avoidance algorithm. Furthermore, the avoidance algorithm could be easily enhanced with additional mapping data to improve the performance during flight time.
The next sub-goal is: "enable autonomous flight ... relying solely onboard sensors and computational resources". All of the control policy has been fully implemented in c code, which can run autonomously on the nano drone without relying on any data or control input from outside. The execution time of $4\,\text{ms} - 10\,\text{ms}$ not only enables fast moving object detection and decision making and thereby fast avoidance, it also leaves computational capabilities for other applications, which could run simultaneously. That means the implementation makes it possible for other applications to collect data or avoid crowded areas in the first place.
Additionally, the drone should also be able to handle "challenging situations". To evaluate this part, the drone has been tested in many different environments. With success

rate of nearly 100% in the open space test described in section 5.7 the results prove that this part has been solved. Especially, the maze setup from section 5.9 shows that the drone can also handle more complicate real life environments with moving objects in narrow areas.
Another very important aspect is the avoidance of "high-speed moving obstacles". This part has been demonstrated in the results chapter section 5.5 and 5.11. Firstly, the avoidance control policy has been tested with single approaches. Here, a maximum speed between 1.9 m/s and 2.5 m/s has been reached. Secondly, the drone has been tested continuously chased in a small cage with fast walking speeds of 2 m/s in average and with peak walking speeds of 3 m/s. This speeds can be considered as high-speed moving obstacles having in mind previous work conducted with the crazyflie 2.1. For example in the paper [10], they reached a maximum drone avoidance velocity of the of 2 m/s on single obstacle avoidance test. However, in full system test with multiple static objects, they only had a 100% reliability with a drone velocity of 1 m/s. To sum it up, compared to that paper, the results can be called high-speed when talking of the Crazyflie 2.1.
The last aspect, which is not directly written in the challenge description but came up through the design of the ToF expansion deck is the "blind spot compensation". A relatively complex control policy with moving object detection has been developed, to provide a solution to that issue.
Looking back, it can be said that most of the goals have been successfully reached.

However, there is still a lot of room for improvement. For instance the avoidance part could be optimised to react even faster and more reliable in also more complex environments. Another issues is the integration of focus and avoidance. Because they function completely separate from each other (one handles sideways/forward velocity and the other yaw rate), they sometimes either crash the drone because too many flying commands are sent or the focus looses track because the avoidance system performs a too fast avoidance maneuver. This could be solved by combining the two in a way that they react on each other. For example, one gets restricted if the other controller uses the full motor actuation power. Lastly, the object clustering and classification could be improved by adding more features. This might improve the differentiation between potential moving obstacle and walls. However, the quite low resolution of the 8x8 multi-zone ToF sensors compared to a camera, for example, limits the classification possibilities of surrounding objects enormously.
To conclude, there are of course still possibilities to improve the control policy. However, from a mathematical/statistical point of view, the quite restricted information input from the ToF sensors also restrict the complexity of the avoidance algorithm. Therefore, it seems like the current implementation has already nearly maxed out the performance if you just look at the complexity of it. That means to improve this control policy significantly, additional sensor should be considered because they improve the information input the drone has for decision making.

Another aspect that could be improved is the testing of the avoidance system. Because the avoidance system has been directly tested in real life environments, many crashes had to be handled by the drone until the control policy was fully functioning. This caused a lot of damage to the drone. Repairing this damages took a lot of time and slowed down the development process. This could be solved by testing the avoidance algorithm in a virtual environment first in order to already have a rough basis working before fine tuning in a real life environment. Of course, it takes also time to learn how to work in such a virtual environment, but safes a lot of hardware material and soldering time afterwards.

To put it all into a nutshell, it can be said that many different approaches have been considered during this thesis and converged into one control policy, which works most of the time. However, there are still things that can be improved in the software, hardware as well as in the testing process.

Appendix A

Task Description

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CENTER FOR PROJECT-BASED LEARNING

Task Description for a Bachelor Thesis on

Nano-Drone autonomous navigation challenge: try to catch me!

at the Department of Information Technology and Electrical Engineering

for

Lukas von Briel
lvonbriel@student.ethz.ch

Advisors: Vlad Niculescu, vladn@iis.ee.ethz.ch
Dr. Tommaso Polonelli, tommaso.polonelli@pbl.ee.ethz.ch

Professor: Prof. Dr. Luca Benini, luca.benini@iis.ee.ethz.ch

Handout Date: 27.03.2023
Due Date: 13.06.2023

1 Project Goals

Nowadays, the industry is pushing the evolution of autonomous flying vehicles (UAV) toward rapidly decreasing form factors and increasing of computational capabilities. On the other hand, pioneering research has already demonstrated nano-scale aerial vehicles, i.e., pico-size UAVs. However, 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, in this project our nano-size UAVs is based on a Crazyflie 2.1 from bitcraze [1], which includes a STM32F4 from STMicroelectronics. If you accept to be part of this project, you will need to design your custom control policy to autonomously fly your Crazyflie 2.1 in challenging situations with high-speed moving obstacles, relying only on onboard sensors and computational resources.

The Crazyflie 2.1 features an indoor localization deck based on an array of four 8x8 pixel depth sensors, which will provide information about incoming obstacles from 360° at a rate of 15 Hz.

Following the example (*https://youtu.be/FyipTqjBGuM*), you will be asked to propose and implement your solution supporting high-speed (>1 m/s) moving obstacles coming from all directions, and… never crash the drone!

[1] https://www.bitcraze.io/products/crazyflie-2-1/

2 Tasks

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

Phase 1 (Weeks 1-5)

1. Investigate and study the state-of-the-art technology for obstacle avoidance on flying objects;
2. Getting started with the STM32 and bitcraze environment, IDE, SDK, and cross compilation/programming;
3. Getting started with the VL53L5CX sensor;
4. Study the reference design and previous related works.

2.1 Phase 2 (weeks 5-12)

1. Implement the obstacle avoidance demo using existing hardware and control algorithms;
2. Propose and discuss your own idea to avoid high-speed moving obstacles coming from 360° (in 2D) using a ToF deck with 4 VL53L5CX sensor facing at opposite diretions;
3. Implementation and optimization of your obstacle avoidance policy;
4. Functional test (demo).

2.2 Phase 3 (last 2 weeks)

1. Finalizing the tests and optimizations on the final version
2. Write the final document and prepare a presentation.

1 Milestones

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

- Good working experience with STM32 and the bitcraze platform;
- Good knowledge of the state-of-the-art technology for autonomous navigation and obstacle avoidance on UAVs.

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

- Working demo with obstacle avoidance capabilities

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

- Practical functional verification;
- Final presentation;
- Final report, including final results;

3 Project Organization

During the thesis, students will gain 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: Student effort, thoroughness and learning curve; Results in terms of quality and quantity; final presentation and report; documentation and reproducibility. All theses include an oral presentation, a written report and are graded. Before starting, the project must be registered in myStudies and all required documents need to be handed in for archiving by PBL.

3.1 Weekly Report

There will be a weekly report/meeting 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 document the progress of the project (required to be done by the student). Beside these regular meetings, additional meetings can be organized to address urgent issues as well. The weekly report, along with all other relevant documents (source code, datasheets, papers, etc), should be uploaded to a clouding service, such as Polybox and shared with the assistants.

3.2 Project Plan

Within the first month of the project, you will be asked to prepare a project plan. This plan should identify the tasks to be performed during the project and sets deadlines for those tasks. The prepared plan will be a topic of discussion of the first week's meeting between you and your advisers. Note that the project plan should be updated constantly depending on the project's status.

3.3 Final Report and paper

PDF copies of the final report written in English are to be turned in. Basic references will be provided by the supervisors by mail and at the meetings during the whole project, but the students are expected to add a considerable amount of their own literature research to the project ("state of the art").

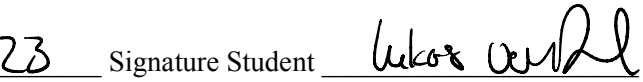
3.4 Final Presentation

There will be a presentation (15 min presentation and 5 min Q&A for BT/ST and 20 min presentation and 10 min Q&A for MT) at the end of this project in order to

present your results to a wider audience. The exact date will be determined towards the end of the work.

References:

Will be provided by the supervisors by mail and at the meetings during the whole project.

Place and Date 8.03.23 Signature Student 

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-Drone autonomous navigation challenge:
Try to catch me!

Authored by (in block letters):

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

Name(s):	**First name(s):**
von Briel	Lukas

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)**
Zürich, 30.5.2023	Lukas von Briel

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

There are two different repositories where the code can be found.

The first repo which can be found under this link:
url: https://gitlab.ethz.ch/lvonbriel/quad-tof-crazyflie-example-app.git
It stores the c implementation of the algorithm and has the following file structure:

```
/
├── README  A README with some general information of how to flash the drone
├── Makefile ..... Makefile for compiler, which includes the firmware
├── src ..... The source and header files of the control policy
│   ├── App defines.h ..... general struct definitions
│   ├── App parameters.h ..... general parameter definitions
│   ├── I2C expander.c/h ..... I2C driver implementation
│   ├── ToFplatform.c/h ..... time of flight driver implementation
│   ├── avoidance.c/h ..... avoidance controller
│   ├── commander interface.c/h ..... sends flying orders to firmware control
│   ├── focus.c/h ..... focus controller
│   ├── helper.c/h ..... general math functionality implemented
│   ├── mapping.c/h ..... moving object detection controller
│   ├── objects.c/h ..... objects clustering and labeling
│   ├── queue drone.c/h ..... personalized implementation of queue
│   ├── test tof.c ..... main file
│   ├── vl53l5cx api.c/h ..... tof driver
│   └── vl53l5cx buffers.h ..... tof buffer
└── .vscode ..... vs code configuration for debugging on st-link
```

The second repo can be found under this link:
url: https://gitlab.ethz.ch/lvonbriel/tof-camera-logger.git
It stores the python implementation of the algorithm which can be run via the Crazyradio. It also contains the plotter script for plotting data collected by the drone or by the vicon system.

```
/
├── README ..... A README with built and run instructions
├── python-app-imaage-tof-logger ..... control policy in python
│   ├── basicAvoidance.py ..... avoidance controller
│   ├── cache.py ..... tranfers data between different threads
│   ├── camera logger.py ..... camera logger (not used in thesis)
│   ├── configs.py ..... configs for cam logger (not used)
│   ├── controller.py ..... sends flying commands to drone
│   ├── crazyflie commander.py ..... not used in thesis
│   ├── crazyflie control.py ..... keyboard control (not used)
│   ├── crazyflie manager.py ..... reads in time of flight data
│   ├── custom logger.py ..... logs all sensor data (not used)
│   ├── focus.py ..... focus controller
│   ├── helper.py ..... additional functions
│   ├── main.py ..... main file
│   ├── mapping.py ..... moving object classification controller
│   └── objects.py ..... object class
└── Plotter ..... plotter script for displaying data
```

Bibliography

[1] V. Niculescu, H. Müller, I. Ostovar, T. Polonelli, M. Magno, and L. Benini. (2022) Towards a Multi-Pixel Time-of-Flight Indoor Navigation System for Nano-Drone Applications. DOI: 10.1109/I2MTC48687.2022.9806701. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9806701

[2] I. Susnea, V. Minzu, and G. Vasiliu, *Simple, Real-Time Obstacle Avoidance Algorithm for Mobile Robots*, 2009.

[3] Bitcraze. Crazyflie 2.1. Datasheet. [Online]. Available: https://www.bitcraze.io/documentation/hardware/crazyflie_2_1/crazyflie_2_1-datasheet.pdf

[4] STMicroelectronics. (2023, Mar.) Time-of-Flight 8x8 multizone ranging sensor with wide field of view. Datasheet. [Online]. Available: https://www.st.com/en/imaging-and-photonics-solutions/vl53l5cx.html#st-also-like

[5] A. Pandya, A. Jhaa, and L. R. Cenkeramaaddi. (2021, Sep.) A Velocity Estimation Technique for a Monocular Camera Using mmWave FMCW Radars. [Online]. Available: https://doi.org/10.3390/electronics10192397

[6] Z. Wei, F. Zhang, S. Chang, Y. Liu, H. Wu, and Z. Feng. (2022, Mar.) MmWave radaar and Vision Fusion for Object Detection in Autonomous Driving: A Review. [Online]. Available: https://doi.org/10.3390/s22072542

[7] V. Niculescu, L. Lamberti, F. Conti, L. Benini, and D. Palossi. (2021, Dec.) Improving Autonomous Nano-Drones Performance via Automated End-toEnd Optimization and Deployment of DNNs. In IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 11, no. 4, pp. 548-562, Dec. 2021, doi: 10.1109/JETCAS.2021.3126259.

[8] K. N. McGuire, C. D. Wagter, K. Tuyls, H. Kappen, and G. de Croon. (2019, Oct.) Minimal navigation solution for a swarm of tiny fyling robots to explore an unknown environment. [Online]. Available: https://doi.org/10.1126/scirobotics.aaw9710

[9] K. N. McGuire, G. de Croon, and K. Tuyls. (2019, Aug.) A comparative study of bug algorithms for robot navigation. [Online]. Available: https://doi.org/10.1016/j.robot.2019.103261

[10] H. Müller, V. Niculescu, T. Polonelli, M. Magno, and L. Benini. (2022, Aug.) Robust and Efficient Depth-based Obstacle Avoidance for Autonomous Miniaturized UAVs. ArXiv:2208.12624v1 [cs.RO] 26 Aug 2022. [Online]. Available: https://arxiv.org/pdf/2208.12624.pdf

[11] C. Brander. (2023, Jan.) Investigating the Deep Sensor Fusion of Camera and ToF Sensors for Obstacle Detection with Nano-UAVs. Bachelor Thesis.