UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Industrial

Development of an autonomous navigation system in highly dynamic environment

Trabajo Fin de Máster – Treball Fi de Màster

Máster Universitario en Ingeniería Industrial-Màster Universitari en Enginyeria Industrial

AUTOR: Vigara Puche, Víctor

Tutores: Martínez Turégano, Jaime; Fumagalli, Matteo

CURSO ACADÉMICO: 2023/2024

DTU

DTU Electro

Development of an autonomous navigation system in highly dynamic environment

Master Thesis

Development of an autonomous navigation system in highly dynamic environment

Master Thesis
Auguts, 2024

By
Victor Vigara Puche

Cover photo: Vibeke Hempler, 2012
Published by: DTU, , Elektrovej, Building 326, 2800 Kgs. Lyngby Denmark
www.electro.dtu.dk
ISSN: [0000-0000] (electronic version)
ISBN: [000-00-0000-000-0] (electronic version)

ISSN: [0000-0000] (printed version)
ISBN: [000-00-0000-000-0] (printed version)

Approval

This thesis has been prepared over six months at the DTU Electro department at the Technical University of Denmark, DTU, in partial fulfillment for the Master's degree in Autonomous Systems

It is assumed that the reader has a basic knowledge of autonomous systems.

Victor Vigara Puche - s222929

Victor Vigara Puche

Signature

09/08/2024

Date

Abstract

This thesis explores the development and application of an autonomous navigation system in dynamic environments. The primary focus is leveraging contact information to overcome challenges such as GNSS-denied environments, sensor degradation, and other common failures, enabling collision recovery.

The thesis begins by addressing the theoretical and technical challenges of autonomous navigation in obstacle-dense environments and the existing approaches to contact detection techniques. After comprehensively evaluating various contact detection alternatives, we propose a novel collision platform and contact detection algorithm integrated into an existing drone frame. These enhancements enable contact-based autonomous navigation capabilities in challenging environments.

The designed collision system was evaluated independently and as part of an autonomous contact-based mission. The findings demonstrate the system's remarkable ability to accurately predict contact events, even integrated within an autonomous mission. This showcases the system's versatility, performing well in static conditions and during real navigation in flight.

Conclusively, this research highlights the potential of contact information to enhance autonomous navigation significantly. The study suggests avenues for future research, particularly in improving collision platform robustness for more challenging environments and integrating additional sensor modalities to further enhance reliability.

Acknowledgements

First, I would like to thank my supervisor, Matteo Fumagalli, who agreed to engage with me with guidance and support throughout this project. I would also like to thank Manuel Jesús Fernández González, a PhD from DTU, and Laura Santas Moreu, an Academic employee from DTU, for his help during the experiments. Finally, I would like to thank my family and friends, whose support has been a constant source of motivation for me throughout this journey.

Contents

1 Introduction

1.1 Context

Aerial robots are not constrained by the morphology of the ground, being able to navigate over obstacles more efficiently than ground robots. That is why flying robots have a unique advantage in various applications, such as exploring dangerous or inaccessible environments for humans, search and rescue, monitoring, and surveillance, among others [1, 2].

1.1.1 The problem

Autonomous navigation in obstacle-dense environments poses significant challenges for flying robots. The 3D scenario, while offering freedom of movement, also increases the risk of collisions. A collision during a mission could lead to mechanical damage to the platform or, even worse, the mission's failure. These challenges underscore the need for robust collision detection and avoidance systems.

Conventional approaches try to solve collision avoidance by discarding the collision events. Most of these approaches are based on vision [3], IR range sensors [4], or lasers, coupling the output of these sensors with localization and mapping algorithms. These approaches are mainly based on exteroceptive sensing, achieving reliable pose estimation and mapping of the environment, and unlocking collision-free navigation solutions to execute autonomous missions.

Collisions are also more probable in GNSS-denied environments, where localization relies on perception sensors such as cameras or lidars. This difficulty level requires advanced sensors, precise mapping, and powerful resources to execute everything in real-time. These requirements sometimes are not available for aerial platforms because of weight restrictions.

There are also strategies to mitigate challenges of degraded sensing [5]; however, reliable solutions to localization and mapping problems under extreme conditions cannot always be taken for granted.

1.1.2 Overview of autonomous robots

An autonomous flying robot is capable of navigating and operating without the help of any external control, making decisions based on the hardware and software architecture. To achieve this autonomy, the typical architecture can be described as follows.

Firstly, exteroceptive and proprioceptive sensors capture the environment and the robot states. These measurements estimate and correct the robot states and represent the environment through maps. That representation enables planning by linking points in the space. Finally, the trajectory created is followed by using a control strategy. The autonomous navigation stack described can be shown in Fig. 1.1.

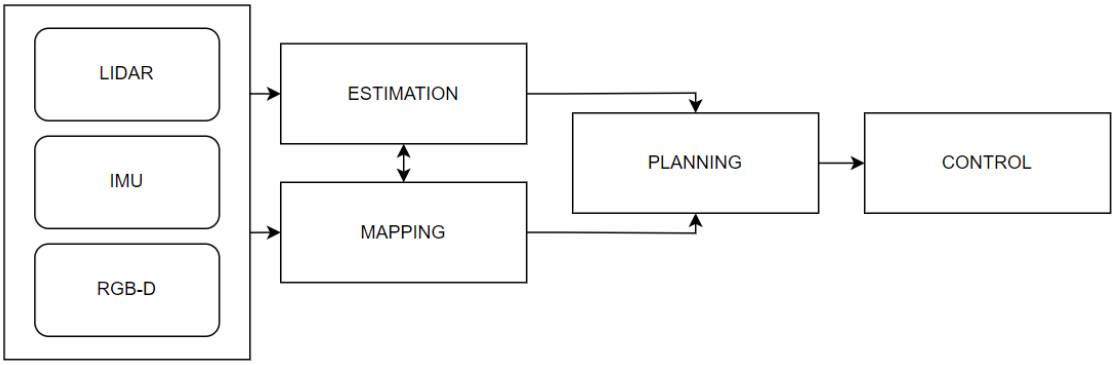


Figure 1.1: General Autonomous Navigation Stack

1.2 Structure of the thesis

The thesis presents the following structure:

- **Theoretical Background** analyzes the previous work related to the thesis topic. It also explains the methodologies implemented in the Methods chapter.
- **Problem Definition** explains the problem solved in the thesis, the starting point, and the contributions to solving the problem.
- **Methods** covers each method developed to achieve the thesis goal.
- **Experiments** describes the experiments to evaluate the methods implemented.
- **Results and Analysis** shows the experiment results and their analysis.
- **Discussion** puts the results into perspective considering other studies.
- **Conclusion** with an overall analysis and proposes future work to improve the solution.

2 Theoretical background

This chapter will delve into several technologies that enable autonomous navigation in challenging environments. It will provide a brief theoretical background and a literature review on various topics relevant to the thesis's contributions described in the following Chapter 3, specifically in Section 3.3.

2.1 PX4 Offboard Control

The control strategy chosen for the Unmanned Aerial Vehicle (UAV) defines how it can navigate. During this project, the main control over the UAV is given by the real-time operating system (RTOS) PX4. The control diagram used by PX4 can be shown in Fig. 2.1.

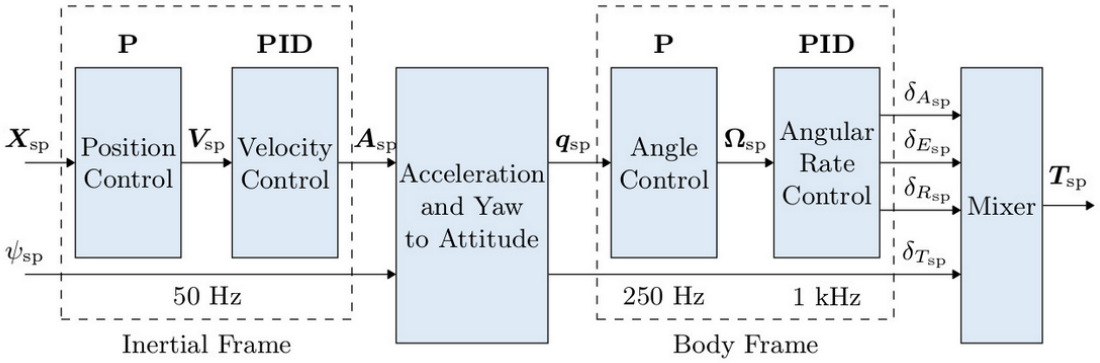


Figure 2.1: Control diagram used by PX4

This is a standard cascade control architecture combining P and PID controllers. Control commands must be sent to the PX4 to control the UAV in offboard mode. These commands will vary depending on the control mode desired, which can be setpoints in position, velocity, acceleration, attitude, attitude rate, or thrust/torque denoted respectively as X_{sp}, V_{sp}, A_{sp}, q_{sp}, Ω_{sp}, T_{sp}, and τ_{sp} [6]. The estimation of the different robot states is performed by the Extended Kalman Filter fusing onboard sensor measurements [7].

2.2 System Frames

In this section, the frames used in this project are described. Usually, for autonomous navigation, the Robotic Operating System (ROS) is used as a middle framework to connect all the subsystems. However, ROS and PX4 do not share the frame convention, so frames presented in Table 2.1 must be considered.

World and Body frames can be visualized in Fig. 2.2

This project will use the following ROS frames:

- The **World frame**: X East, Y North, Z Up.
- The **UAV Body frame**: Attached to the center of the UAV. X Forward, Y Left, Z Up.
- The **Lidar frame**: Attached to the center of the lidar. X Forward, Y Left, Z Up

Frame	PX4	ROS
Body	FRD (X Forward, Y Right, Z Down)	FLU (X Forward, Y Left, Z Up)
World	FRD or NED (X North, Y East, Z Down)	FLU or ENU (X East, Y North, Z Up)

Table 2.1: Frames conventions in PX4 and ROS

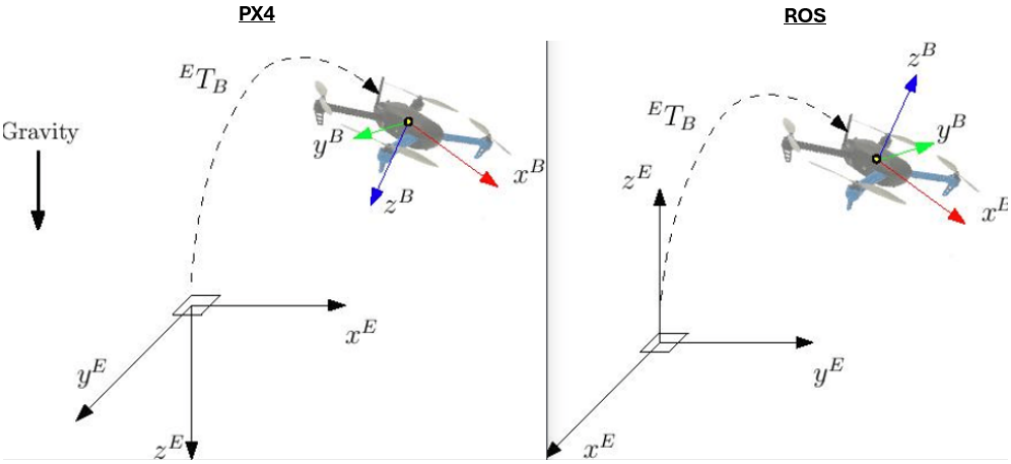


Figure 2.2: World and UAV Body frames conventions in PX4 and ROS

2.3 Autonomous navigation in obstacle-dense environments

Autonomous navigation in obstacle-dense environments, such as forests, presents significant challenges due to the complexity of maneuvering through numerous and often unpredictable obstacles. Various algorithms have been developed to address these localization, mapping, and path planning challenges. This section will cover some approaches, highlighting their methodologies and common failures.

2.3.1 Localization

Localization involves determining a vehicle's pose relative to a mapped environment based on sensory inputs. Several approaches exist to address this problem.

Monte Carlo Localization [8], known as particle filtering, employs particles to represent potential robot states, updating their likelihood based on sensor data. This method is particularly effective in environments with non-linear characteristics and noisy sensors. The Kalman Filter [9] continuously refines the system's estimated state by integrating new sensor information in an optimal statistical manner, making it ideal for scenarios with well-modeled robot dynamics and Gaussian noise.

Scan Matching is essential for robots with LiDAR or similar sensors, aligning scan data with a map or previous scans to determine movement. Visual-inertial odometry (VIO) and Simultaneous Localization and Mapping (SLAM) are commonly used in GNSS-denied environments. VIO combines camera visual data with inertial data from IMUs to estimate the UAV's position and orientation. SLAM algorithms build and update a map of the environment while simultaneously keeping track of the UAV's location within that map.

For example, the system proposed in [10] utilized VIO combined with a LiDAR-based SLAM to correct for drift in VIO estimates during autonomous flights in North American pine forests. This approach showed good localization accuracy but struggled with drift in dense forest environments, which could lead to navigation errors.

2.3.2 Mapping

Mapping involves creating a representation of the environment that the UAV can use for navigation. Standard mapping techniques include occupancy grid maps, voxel maps, and point clouds. Stereo cameras and LiDAR sensors are often used to generate these maps. [11] used a point cloud from an active stereo depth camera to create a voxel map, focusing on computational efficiency for real-time processing with a low-power CPU. While effective in sparse environments, this method faced challenges in dense forests where small obstacles like branches were often missed, leading to potential collisions.

OctoMap [12] is another powerful tool for mapping 3D environments. It represents the environment with an efficient 3D occupancy grid, allowing for the detection and avoidance of obstacles. OctoMap uses probabilistic occupancy estimation, which provides a compact and flexible representation of the environment, making it suitable for dynamic and cluttered environments like forests. It integrates well with various sensors, offering a robust framework for autonomous navigation.

2.3.3 Path Planning

Path planning algorithms generate a collision-free trajectory for the UAV to follow. Several approaches have been explored, including decomposition hierarchic methods, EGO-Planner, and geometric planners.

In [10], the decomposition hierarchic method is used. It separates path planning and trajectory generation. An A* search algorithm plans a collision-free path in a 3D grid map, serving as the path planner. The maneuver automation method generates a feasible trajectory for the UAV. This approach has been effective in environments like forests and urban canyons but struggles with real-time implementation due to computational complexity. It also faces challenges in dynamic environments where obstacles may move unpredictably.

In [11], uses EGO-Planner-v2 [13] for path planning and VINS-Fusion [14] for localization, integrating stereo-depth camera-based mapping. Initially tested in simulators, the algorithms were later deployed on custom drone hardware for real-world testing, showing promising results in boreal forests. However, the reliable detection of small and leafless branches remained challenging, leading to collisions, and only about half of the test flights succeeded.

Geometric planners, such as Rapidly-exploring Random Tree (RRT) [15] and its variants like RRT*, are widely used for their ability to efficiently explore large spaces and find feasible paths. RRT* improves upon RRT by optimizing the path to make it shorter and smoother. These planners are popular due to their simplicity and effectiveness in various environments. However, they can be computationally intensive and may not always find the most optimal path in complex environments. Additionally, their performance can degrade in real-time applications due to the high computational demands.

In [16], a dynamic path planning method is implemented using RRT*. As implemented in the paper, it involves the robot detecting obstacles during path execution and triggering a replanning event when an obstruction is encountered. The robot invalidates the obstructed nodes and identifies a new replan goal location. It then modifies the existing RRT* tree by sampling new nodes and rewiring it to ensure a collision-free path around the obstacle. This process allows the robot to dynamically adapt its path, maintaining optimal navigation in environments with moving obstacles.

2.3.4 Common Failures

Despite advancements, several common failures still occur in autonomous navigation systems:

- **Obstacle Detection**: Small obstacles, such as thin branches, often go undetected, especially by depth cameras. This can lead to collisions, as seen in [11], where only nine out of nineteen test flights were successful due to poor detection of needleless branches and leafless vegetation.
- **Real-Time Processing**: Many algorithms struggle with the computational demands of real-time processing. The decomposition hierarchic method, for example, can be too slow for dynamic environments, leading to delayed responses and collisions.
- **Localization Drift**: In GNSS-denied environments, drift in VIO can cause significant localization errors over time, leading to navigation failures. This issue was noted in several studies, where prolonged flights resulted in accumulating errors despite SLAM corrections

2.4 Contact detection

Collision events, often unpredictable and undetectable when exteroceptive sensing is compromised, pose a challenge to reliable localization and mapping. Some studies demonstrate that collision-tolerant flying robots can encounter obstacles, react, and self-recover after the collision. This approach, particularly appealing in complex environments, complements other sensing modalities, such as perception, and reduces the complexity of obstacle avoidance and control, among other applications.

The sense of touch is commonly used by ground robots for navigation [17] in several tasks such as area coverage, for example, vacuum cleaning [18] or obstacle following [19]. Other strategies include detecting unpredictable or moving obstacles that other sensors may not detect [20].

An approach to improving the robustness of flying platforms in cluttered environments is introduced in [21]. This approach is inspired by insects, which cannot avoid all obstacles but can recover after collisions [22]. Thanks to the platform's robustness and ability to crash, autonomous navigation can be performed without the caution and low speed required for sensing and mapping to avoid obstacles; these would not need to be ideally detected, drastically reducing the sensing system's complexity. In addition, this sensing modality is helpful in dynamic environments.

2.4.1 Collision protection approaches

A comparison between insects and flying robots when discussing protection mechanisms can be made. Flying animals have improved their robustness against collisions, providing some inspiration for collision protection mechanisms. Due to their low mass, insects do not achieve high-impact forces during free fall. That is why their rigid exoskeletons are enough to absorb the forces. However, as animals grow, their mass increases cubically, whereas their surface area only increases quadratically, so the air drag can no longer limit the impact force. This mass increment led to heavier animals and humans protecting the skeleton by soft tissue enhancing impact energy absorption.

Comparing animal's endo- and exoskeletons, collision protection platforms for flying robots can be categorized into two types of structures:

- **Stiff protection** is commonly used in commercial drones and is enough to protect propellers from low-energy contact

- **Soft protections** are typically built with foam or deformable cages with thin components like carbon fiber rods.

Different protection approaches can be seen in Fig. 2.3.

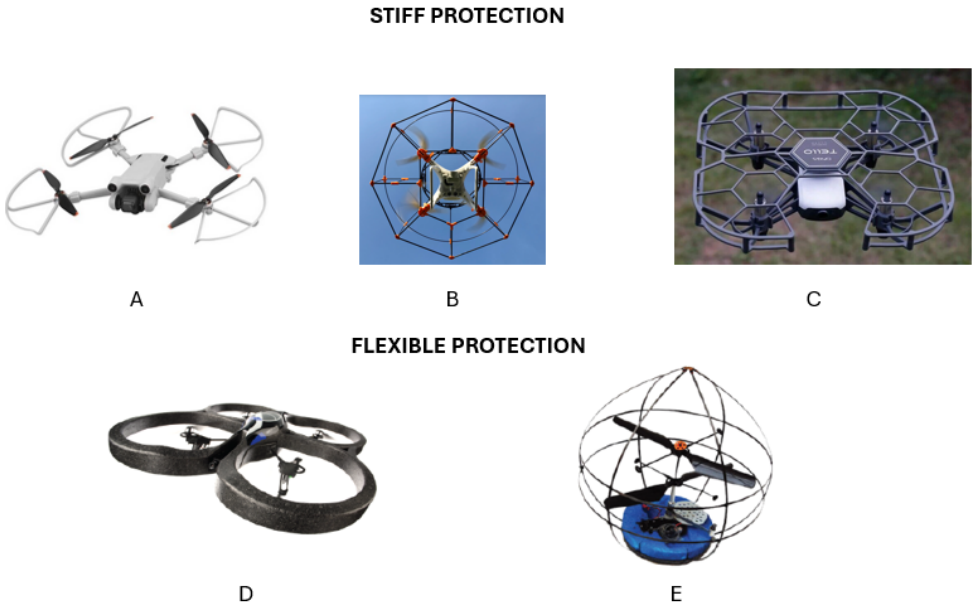


Figure 2.3: Collision protection platforms used in drones

As shown in Fig. 2.3, we can find different protection levels. Fig. 2.3A shows propeller protection of a *DJI mini 3 Pro* made of 3D printed PLA. On a higher protection level, Fig. 2.3B shows a 2D protection around the drone platform made of carbon fiber rods. Finally, we can also find 3D protection cages that protect from impacts in all directions, as shown in Fig. 2.3C. The highest drawback of stiff protection is the direct force transferred from the impact point to the drone's frame, decreasing the robot's stability.

On the other hand, several solutions can be found for flexible protections. Fig. 2.3D shows a drone frame protected with foam. This material distributes the force over a large area. However, it presents a significant drawback because when absorbing the energy, the foam is crushed, and repeated impacts can reduce the energy it can absorb. Another option for flexible protection is using one or several flexible carbon rods, as seen in Fig. 2.3E. Carbon fiber is stiffer compared to foam, and it does not lose energy absorption ability with repeated contact.

In [23], a method is presented for designing protective structures that transfer the lowest possible amount of force to the platform's frame while simultaneously minimizing weight and thus their effect on flight performance. After designing collision-tolerant platforms, it focuses on self-recovery, which means the UAV returns to flight without requiring external intervention. However, in that study, the collision was not detected, and there was no reaction during the collision event, just after the crash when the UAV fell. In the following Section2.4.2, we will focus on collision detection approaches.

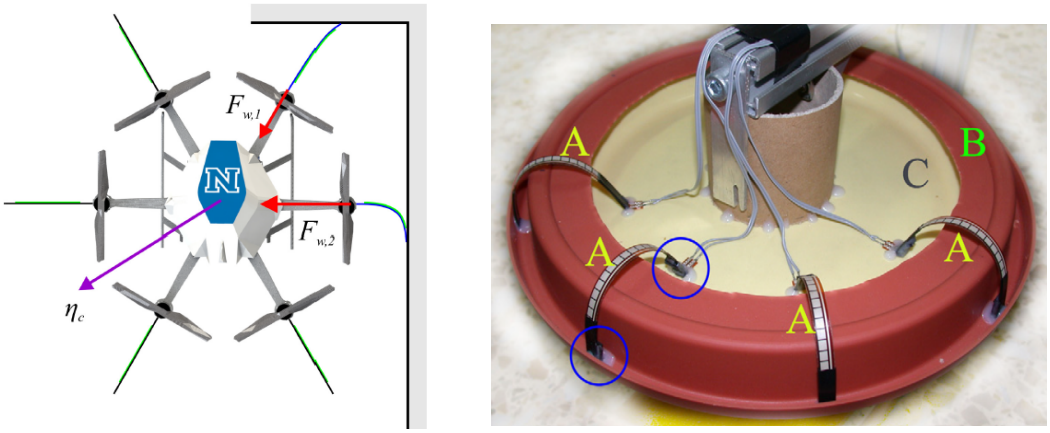
2.4.2 Contact detection approaches

Several approaches can solve this problem when discussing contact or collision detection. In this section, an overview of these approaches, which have been of inspiration for this project, is shown.

The sense of touch can be implemented with a variety of sensors. Basic contact sensing has been implemented in the past using accelerometers. Accelerometers have a drawback because they can only detect high-impact forces. So, it could get stuck if the robot collides at low speed or with a soft obstacle. It is also possible to detect contacts by comparing the expected motion of the system and the current motion [19] where collisions are detected when odometry does not match the expectation. The main advantage of these approaches is that these kinds of sensors are almost always implemented on an aerial platform, so no new implementations are needed.

On the other hand, several sensors can be used to know if a platform is in contact with an obstacle and where the obstacle is. The typical sensors found mainly in ground robots are tactile sensors [24], whiskers [25], and proximity sensors [26].

Regarding whiskers, many solutions can be implemented to simulate these mammals' hair to sense the environment. One option to simulate whiskers is Flex sensors [27]. The working principle of a flex sensor is based on the concept of variable resistance due to mechanical deformation. This kind of sensor has been used in [28] as whiskers integrated perimetrically into the robot's body as shown in Fig. 2.4a. Each robotic antennae, as they call it, provides force feedback estimates through bending angles during physical interaction. The interacting antennas are calibrated against force and bending angles. Given the estimates of the interacting forces, they implement a reactive control strategy to ensure collision-tolerant flight. In Fig. 2.4a, the reactive force η_c is calculated as the resultant of $F_{w,1}$ and $F_{w,2}$. However, this approach of perimetrically distributed antennas would only detect collisions with planar surfaces, so small or thinner objects could collide with the robot platform between each antenna because there are blind areas.



(a) Haptic Feedback-based Reactive Navigation [28]

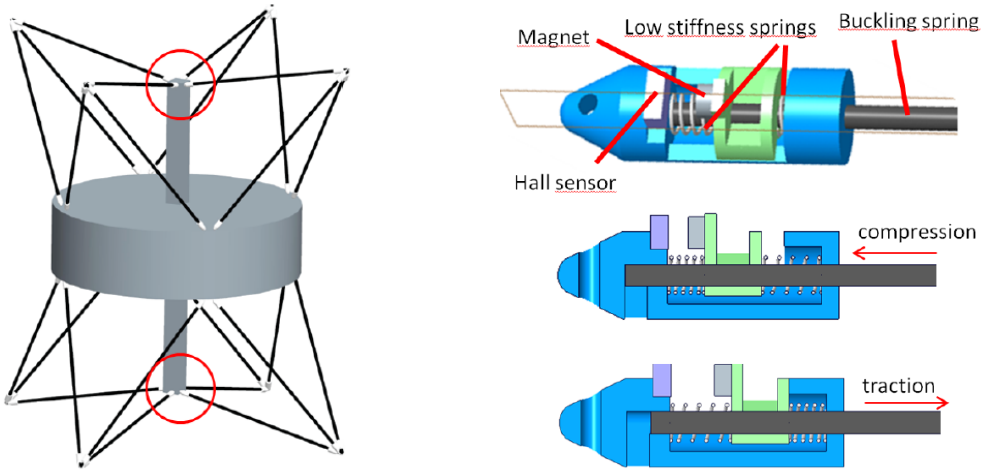
(b) Obstacle contact sensor [29]

Figure 2.4: Two different contact detection approaches using Flex sensors

Another interesting flex sensor approach is presented in [29]. Instead of detecting collisions directly with the flex sensors colliding with the obstacles as done in the previous approach, flex sensors are used to measure displacements between two platforms. As shown in Fig. 2.4b, the sensors are placed around the mobile system's perimeter, a sliding platform on top of a second fixed platform. It is explained that only two sensors are required to measure two orthogonal directions. Still, the rotation could disturb the measurements, so two sets of the basic two-element system are installed to diminish the effect through redundancy. A fuzzy inference system is implemented to predict the contact zone,

classifying the contacts into eight zones.

Another approach accounting for 3D collision detection is presented in [30]. This study uses the AirBurr protective structure [23]. This structure is formed by 24 buckling springs arranged in 8 tetrahedral configurations, as shown in Fig. 2.5a. The eight pivot joints indicated in Fig. 2.5a have been replaced by the sensors pictured in Fig. 2.5b. These sensors measure the deformation of low-stiffness springs thanks to hall sensors.



(a) AirBurr tetrahedral platform (b) Force sensor based on hall sensor

Figure 2.5: Force sensors integrated in the AirBurr platform [30]

The force sensors implemented in Fig. 2.5b provide a force range of 1N, so it could detect low impacts that accelerometers could not differentiate between an impact or a normal flight maneuver.

Torque sensors have also been used to design contact detection platforms in [31]. They equipped a quadrotor with a disc-shaped shell protecting the drone frame from collisions. The shell surface is made of low-friction fiberglass, allowing obstacles to slide along its surface. A six-axis load cell connects the quadrotor's frame and the shell to enable force feedback control. So, the haptic sensor measures the net wrench resulting from contacts occurring across the entire shell surface. The mechanism integrating the shell, the load cell, and the quadrotor frame is shown in Fig. 2.6.

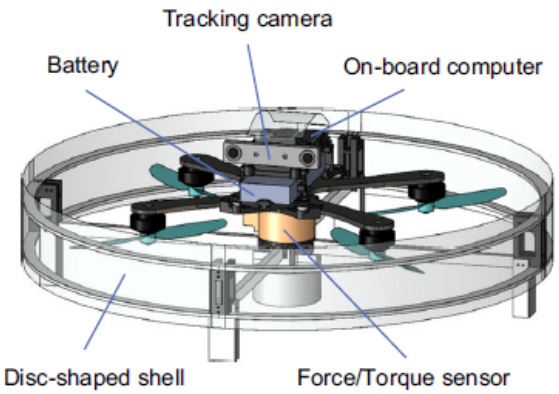


Figure 2.6: Shell disc integrated with torque sensor for contact detection [31]

The disc-shaped shell protection and detection mechanism have been tested in push and slide tasks.

2.5 Deep learning foundamentals

Artificial intelligence is a broad field of computer science aimed at creating systems capable of performing tasks that require human intelligence. These tasks include decision-making,e.g., autonomous driving [32]; problem-solving,e.g., medical diagnosis [33]; and learning, e.g., language translation [34], among others.

Deep learning is a Machine Learning subset that uses Deep Neural Networks to discover patterns in large datasets. These datasets can be labeled so that the ground truth for each data point is known, and it can be used for the neural network training or unlabeled so patterns are recognized without sharing any label with the model. Based on labels in the dataset, deep learning can be divided into the following categories:

- **Supervised learning**: This is defined by using labeled datasets to train algorithms to classify data or predict outcomes accurately. As input data is fed into the model, the model adjusts its weights until it has been fitted appropriately. It is typically categorized into:
  - Classification: Predicting a discrete label.
  - Regression: Predicting a continuous value.
- **Unsupervised learning**: Involves training a model on data without labeled responses, aiming to find hidden patterns or intrinsic structures in the input data. Common techniques include:
  - Clustering: Groups similar data points together
  - Dimensionality reduction: Simplifies the data by reducing the number of features while retaining important information.
- **Semi-seupervised learning**: During training, a smaller labeled data set guides classification and feature extraction from a larger, unlabeled data set. Semi-supervised learning can solve the problem of not having enough labeled data.
- **Reinforcement Learning**: Involves training an agent to make sequences of decisions by rewarding desired behaviors and punishing undesired ones. This technique is particularly effective in dynamic environments where the agent must adapt its strategy based on feedback.

This section will focus on the supervised learning category, in which the dataset is labeled to learn patterns based on the existing data.

2.5.1 Feedforward Neural Network

A neuron, also known as an information processing unit, is the fundamental building block of neural networks. A single neuron is the simplest form of a neural network model, which can be described mathematically by the function $f(x, w)$. This function takes a set of inputs x and learning parameters called weights w and produces an output that can be either positive or negative.

Each input x_i is multiplied by its corresponding weight w_i. These products are then summed to produce a single value:

$$z = \sum_i x_i w_i + b \tag{2.1}$$

where b is the bias term, which allows the activation function to be shifted to the left or right. The weighted sum z is typically passed through an activation function $\phi(z)$.

A perceptron is an artificial neuron that includes an activation function, usually a step function, which converts the weighted sum of the inputs into a binary output. The perceptron can classify input data into one of two classes. The operation of a perceptron can be expressed as:

$$f(x) = \begin{cases} 1 & \text{if } \sum_i x_i w_i + b > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

The perceptron's decision boundary is linear, meaning it can only solve problems where the data is linearly separable.

To handle non-linearly separable data, a more complex structure called a Multilayer Perceptron (MLP) is used, which is a type of Feedforward Neural Network (FFNN) [35]. An MLP consists of multiple layers of neurons:

- **Input Layer**: This layer receives the input data. Each neuron in this layer represents one feature of the input data.
- **Hidden Layers**: These layers are positioned between the input and output layers. Each hidden layer consists of multiple neurons connected to every neuron in the previous and next layers. The purpose of hidden layers is to capture complex patterns and interactions in the data. The more hidden layers a network has, the deeper it is, allowing it to learn more abstract features. Including non-linear activation functions in these layers allows the network to model non-linear relationships in the data.
- **Output Layer**: This layer produces the network's final output. In a classification problem, the output layer might use a softmax activation function to produce a probability distribution over the classes. In a regression problem, it might use a linear activation function to produce a continuous output.

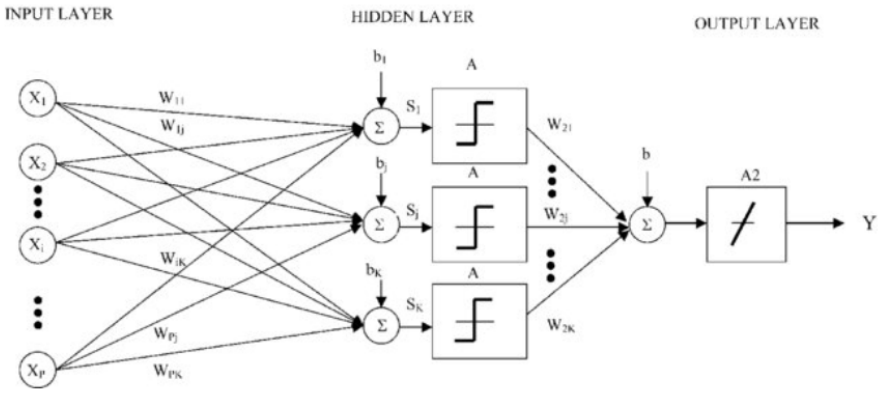


Figure 2.7: Feed forward neural network explained in [36]

2.5.2 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a class of artificial neural networks proposed in the 1980s [37] designed to recognize patterns in data sequences, such as time series, speech, text, financial data, and more. Unlike FFNNs, RNNs have connections that form directed cycles, creating an internal network state that allows them to exhibit temporal dynamic behavior, i.e., the last output is influenced not only by the previous input but by

all the inputs fed in the network. RNNs are particularly well-suited for tasks where context and sequential information are crucial.

One of the key ideas behind RNNs is parameter sharing. This concept allows RNNs to maintain consistent parameters across different time steps, enabling the network to generalize effectively across sequences of varying lengths. In contrast, ordinary FFNNs learn separate parameters for each input position, which limits their ability to handle sequences of different lengths. Parameter sharing in RNNs ensures that the same set of weights is applied to each element in the sequence, which helps the model capture temporal dependencies and patterns irrespective of the input sequence length. This property is crucial for tasks involving sequential data, where generalizing across different sequence lengths is essential [38].

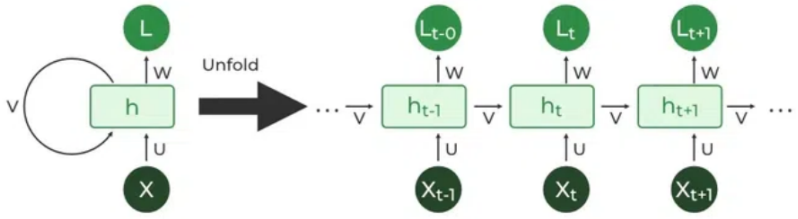


Figure 2.8: RNN architecture

Fig. 2.8 shows a typical RNN architecture. The variable *x* is the input of length t+1, which could be easily decreased or increased thanks to parameter sharing. These shared parameters are *U*, defining the weights connecting the input to the hidden units *h*; *V*, defining the weights from the hidden layer *h*; and *W*, defining the weights from the hidden layer to the output neurons.

However, simple RNN architectures, such as the one shown above, can lead to vanishing/exploding gradient problems while training when the architectures become deeper. The vanishing gradient problem occurs when the gradients of the loss function become extremely small, making it difficult for the model to learn. On the other hand, the exploding gradient problem occurs when the gradients become extremely large, leading to unstable training and convergence issues.

Other RNN types, such as Long-Short Term Memory (LSTM) modules, are used to tackle the gradient problem. In that case, the simple hidden units from RNN are replaced by more complex units called LSTM blocks.

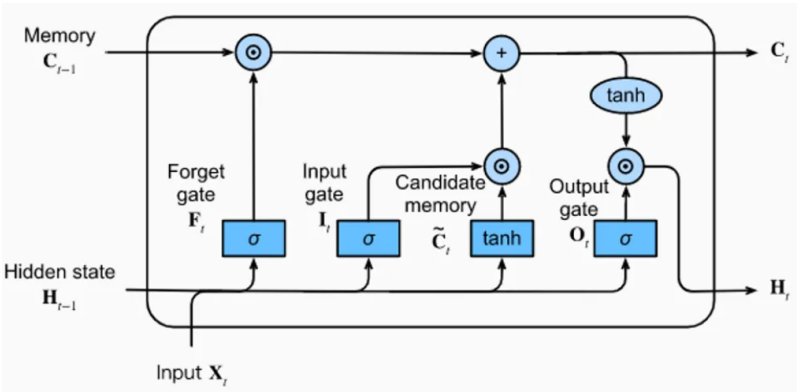


Figure 2.9: LSTM block explained in [39]

An LSTM unit consists of a cell Memory C_t and three gates: the input gate I_t, the forget gate F_t, and the output gate O_t. These gates control the addition and removal of information from the cell Memory, as depicted in Figure 2.9.

The forget gate determines what portion of the cell Memory from the previous time step should be carried forward to the current time step. It uses a sigmoid function to generate values between 0 and 1, where 0 means completely forgetting the information and 1 means keeping it entirely.

$$F_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{2.3}$$

The input gate I_t controls how much of the new information from the current input x_t and the previous hidden state h_{t-1} should be added to the cell Memory. The Candidate memory cell $\tilde{C}_t$ is created using the tanh activation function, which produces values between -1 and 1. The input gate then scales this candidate cell Memory using the sigmoid function before adding it to the current cell Memory.

$$I_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{2.4}$$

$$\tilde{C}_t = \tanh(W_C[h_{t-1}, x_t] + b_C) \tag{2.5}$$

The cell Memory C_t is updated by combining the information preserved by the forget gate and the new information scaled by the input gate.

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{2.6}$$

The output gate O_t determines which parts of the cell Memory should be output as the new hidden state h_t. This hidden state is then used for predictions and input for the next step.

$$O_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{2.7}$$

$$H_t = o_t \odot \tanh(C_t) \tag{2.8}$$

Where σ is the sigmoid activation function, $\odot$ represents element-wise multiplication, W_f, W_i, W_C, and W_o are weight matrices, b_f, b_i, b_C, and b_o are bias vectors.

2.5.3 Activation Functions

An activation function is an element-wise differentiable operation applied to the output of a neuron, required to calculate hidden layer values, helping neural networks perform complex tasks such as image recognition, natural language processing, and speech recognition. The purpose of an activation function is to introduce non-linearity into the model, allowing the network to learn and represent complex patterns in the data. Without non-linearity, a neural network would behave like a linear regression model.

Non-linear activation functions are defined based on their range or curves. The following activation functions are used in the project within the FFNN or the LSTM block:

- **Sigmoid Function**: The sigmoid activation function is a continuously differentiable and non-linearly separable function that squeezes the output values in the range [0,1].

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.9}$$

  It is especially used for models where the required output is a probability.

- **Rectified Linear Unit (ReLU)**: One of the most popular non-linear squashing functions, ReLU generates the maximum between 0 and input x as output.

$$\text{ReLU}(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.10}$$

- **Hyperbolic Tangent (tanh)**: The tanh activation function is another popular non-linear function that outputs values in the range [-1, 1]. It is defined as:

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.11}$$

2.5.4 Loss Functions

A loss function, a cost or objective function, is a crucial component in training neural networks. It quantifies how well the predicted outputs of the model match the true targets. The ultimate goal of the training process is to minimize the loss function, thereby significantly improving the model's accuracy. This underlines the pivotal role of the loss function in the field of machine learning.

Two of the main loss functions used for regression and classification tasks, respectively, are described below:

- **Mean Squared Error (MSE)**: Commonly used in regression tasks, MSE measures the average of the squares of the errors between the predicted and actual values. It is given by:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2.12}$$

  where y_i is the actual value and $\hat{y}_i$ is the predicted value.

- **Binary Cross-Entropy Loss**: Commonly used in binary classification tasks. It measures the difference between two probability distributions: the label's true distribution and the model's predicted distribution. For binary classification, it is given by:

$$\text{Binary Cross-Entropy} = -\frac{1}{n} \sum_{i=1}^{n} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)] \tag{2.13}$$

  where y_i is the actual label and $\hat{y}_i$ is the predicted probability.

However, there are also cases where custom loss functions need to be defined. This is the case when predicting angles. Angles have a circular topology, so the MSE loss function would not work properly when calculating the loss between 0 and 359 degrees; breaking the continuity between 359 and 0 MSE would account for the maximum error. One way to solve this issue is to encode angle values as pairs of sine and cosine. With that representation, a custom loss function could be defined to minimize the Euclidean distance between predicted and true sine/cosine pairs, as shown below:

$$\text{Sine Cosine Distance Loss} = \sqrt{(\sin(\hat{\theta}) - \sin(\theta))^2 + (\cos(\hat{\theta}) - \cos(\theta))^2} \tag{2.14}$$

where $\sin(\hat{\theta})$ and $\cos(\hat{\theta})$ are the predicted sine and cosine values, and $\sin(\theta)$ and $\cos(\theta)$ are the target sine and cosine values.

The loss function's choice directly impacts gradient descent, as it influences the gradients calculated during back-propagation, thereby affecting the direction and magnitude of parameter updates. The design of the loss function is task-specific. Therefore, selecting the appropriate loss function is critical for effective model training, ensuring the neural network is optimized correctly and performs better.

2.5.5 Gradient Descent Optimization

Gradient descent is a fundamental optimization algorithm that minimizes the error function in machine learning tasks by iteratively updating learning parameters based on gradients computed through partial derivatives. In each iteration, the parameters (weights and biases) are adjusted to reduce the loss, gradually converging towards a minimum of the cost function. The algorithm is crucial for training deep neural networks due to the non-convex nature of their cost functions. Non-convexity means that the cost function can have multiple local minima, making it difficult to find the global minimum.

The basic gradient descent update rule is given by:

$$\theta = \theta - \alpha \nabla J(\theta) \tag{2.15}$$

where θ represents the model parameters, α is the learning rate, $J(\theta)$ is the cost function, and $\nabla J(\theta)$ is the gradient of the cost function for the parameters.

The learning rate is a hyperparameter that determines the step size at each iteration while moving toward a minimum of the cost function. The choice of learning rate is critical for the convergence of gradient descent. If the learning rate is too high, the algorithm may overshoot the minimum and fail to converge. The convergence can be very slow if the learning rate is too low.

2.5.6 Metrics for Regression

In this section, we will delve into popular metrics for evaluating regression models and how to calculate them.

Mean Squared Error (MSE)
It is a widely used error metric for regression problems. The MSE is calculated as the mean of the squared differences between predicted and actual target values in a dataset:

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{2.16}$$

where y_i is the i-th actual value and $\hat{y}_i$ is the i-th predicted value. Squaring the differences ensures that the error values are positive and penalizes larger errors more heavily, making highlighting significant prediction errors useful.

Root Mean Squared Error (RMSE)
It is obtained by taking the square root of the MSE. This adjustment restores the units of the RMSE to be the same as the original target values, making it more interpretable. RMSE can be particularly useful for understanding the magnitude of prediction errors in the same units as the target variable.

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{2.17}$$

Mean Absolute Error (MAE)
It is another common metric for regression. Unlike MSE and RMSE, which penalize larger errors more, MAE treats all errors equally by averaging the absolute differences between predicted and actual values:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{2.18}$$

This linear error metric provides an intuitive measure of model accuracy, making it easier to understand how the model performs across different prediction ranges.

2.5.7 Metrics for Binary Classification

This section will explore the commonly used metrics for evaluating binary classification models and the methods to calculate them.

Accuracy
Accuracy is the most straightforward metric for classification problems. It is calculated as the ratio of correctly predicted instances to the total instances in the dataset:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{2.19}$$

where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN is the number of false negatives. Accuracy provides a general measure of how well the model is performing but can be misleading if the dataset is imbalanced.

Precision
Precision, also known as the positive predictive value, measures the proportion of positive predictions that are correct. It is calculated as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.20}$$

Precision is particularly useful in scenarios where the cost of false positives is high. It indicates the reliability of the model when it predicts the positive class.

Recall
Recall, or sensitivity, measures the proportion of actual positives that the model correctly identifies. It is given by:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.21}$$

Recall is crucial in applications where the cost of false negatives is high, as it captures the model's ability to detect all positive instances.

F1-Score
The F1-Score is the harmonic mean of precision and recall, providing a balance between the two metrics. It is calculated as:

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.22}$$

The F1-Score is useful when you need to balance precision and recall, particularly in situations where there is an uneven class distribution.

Area Under the Curve (AUC)
The Area Under the Receiver Operating Characteristic Curve (AUC-ROC) is a metric that evaluates the ability of the model to distinguish between the positive and negative classes. The ROC curve is plotted with the true positive rate (recall) against the false positive rate (1 - specificity):

$$\text{AUC} = \int_0^1 \text{ROC}(x)\, dx \tag{2.23}$$

AUC ranges from 0 to 1, with a value closer to 1 indicating better model performance. It provides a single measure of overall model performance across all classification thresholds.

2.6 Sensor pattern prediction

Pattern recognition is crucial in analyzing data from multiple sensors to predict various outcomes. Various methods have been developed and applied successfully in different domains, demonstrating their capabilities and limitations.

One commonly used pattern recognition method is decision trees and random forests. These algorithms can handle non-linearities and interactions within the data, making them suitable for complex sensor data analysis. For instance, in the study [40], random forests were utilized to predict crop yields using multiple environmental and satellite data inputs.

Another method that has shown promise is fuzzy logic. Fuzzy logic systems use linguistic variables and rules to manage non-linearity and ambiguity in sensor data. This approach is beneficial in scenarios where the data is imprecise or noisy. In the paper [29], fuzzy logic was employed to enhance the accuracy of flex sensor measurements. The study demonstrated how fuzzy logic could handle the non-linear characteristics of flex sensors, making it a viable approach for integrating contact angle prediction. The complexity of integrating multiple outputs, such as displacement, angle, and contact prediction, can be managed by defining new sets of rules for each additional output and sensor, leveraging the flexibility and robustness of fuzzy logic systems.

Neural networks offer significant advantages for pattern recognition in sensor data. They can model complex, non-linear relationships and handle multiple outputs by adding neurons to the output layer. This scalability makes neural networks highly suitable for simultaneously predicting numerous variables. In the study [41], FFNN and CNN were applied to classify arrhythmias in real time using Electrocardiography (ECG) data. Additionally, in another study [42], an LSTM-based approach was used for tilt angle monitoring. The authors proposed a Sparse Residual LSTM network combined with grid search to optimize hyperparameters, demonstrating significant improvements in accuracy and robustness for dynamic tilt angle prediction. This shows the applicability of neural networks in processing and predicting outcomes from complex sensor data.

The k-nearest neighbors (KNN) algorithm is a simple yet effective method for pattern recognition. KNN classifies data points based on their proximity to training data points. This approach can be helpful for sensor data analysis, where similar patterns are expected to yield similar outcomes. In the paper [43], KNN was applied to classify activities based on data from wrist-worn inertial sensors.

These methods, each with their strengths and limitations, offer a range of tools for effective pattern recognition in sensor data. By leveraging the appropriate algorithm based on the specific characteristics and requirements of the data, it is possible to achieve accurate and reliable collision event predictions. The literature demonstrates the successful application of these methods across various domains, providing valuable insights and methodologies that can be adapted to solve similar problems in sensor data analysis.

3 Problem Definition

This chapter defines the problem that this thesis addresses and outlines the initial resources and tools available for tackling this challenge. As stated in Section 1, aerial robots offer significant advantages in performing complex tasks due to their 3D freedom of movement. However, this increased maneuverability also raises the risk of collisions in dense obstacle environments.

The primary objective of this thesis is to manage and mitigate the risk of collisions during autonomous aerial missions. To achieve this, we will leverage the hardware and software components that are specifically chosen to address this issue effectively.

This chapter will detail the elements used to develop the project, including the hardware setup and the software tools that will facilitate the solution. Finally, the specific contributions of this thesis will be outlined.

3.1 Hardware

This section describes the hardware used to develop the thesis.

3.1.1 Holybro X500 v2

The UAV used for this project is the quadrotor X500 v2 from the PX4 Development Kit [44]. This is the frame over which the different components (PX4, lidar, optitrack markers, and collision platform) will be placed.

Figure 3.1: Holybro X500 v2 PX4 Development Kit

3.1.2 LattePanda 3 Delta

As the onboard computer, the LattePanda 3 Delta 864 is used. It is mounted in one of the Holybro frame legs. The computer specifications are shown in Table 3.1

In addition, this computer also incorporates an Arduino pin interface, which is useful for connecting sensors with analog outputs.

Components	Latte Panda 3 Delta
Processor	Intel Celeron N5105
Core	2.0-2.9GHz Quad-Core, Four-Thread
Graphics	Intel UHD Graphics (Frequency: 450 – 800MHz)
Memory	LPDDR4 8GB 2933MHz
Storage	64GB eMMC

Table 3.1: LattePanda 3 Delta specifications

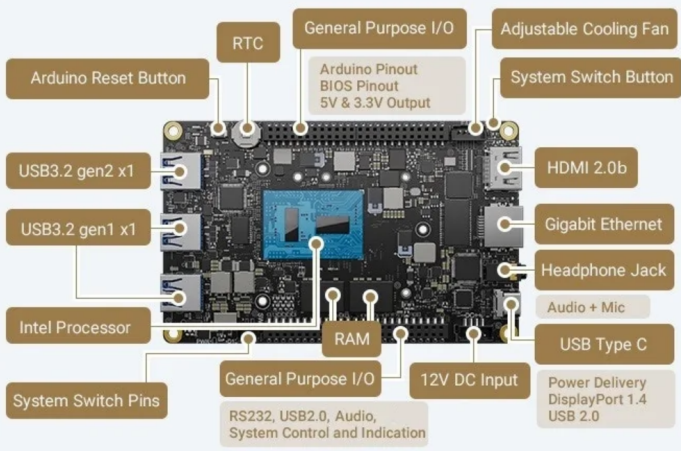


Figure 3.2: LattePanda 3 Delta

3.1.3 Livox Mid-360

For sensing the environment, the 3D lidar Livox Mid-360 has been chosen because of its high resolution, as well as its low weight compared with other 3D lidars as shown in the specifications in Table 3.2.

Specs	Livox Mid-360
Detection Range (@ 100 klx)	40 m @ 10% reflectivity - 70 m @ 80% reflectivity
Close Proximity Blind Zone²	0.1 m
FOV	Horizontal: 360°, Vertical: -7°52°
Range Precision³ (1σ)	≤ 2 cm (@ 10m)
Angular Precision1σ	< 0.15º
Point Rate	200,000 points/s (first return)
Frame Rate	10 Hz (typical)

Table 3.2: Livox Mid-360 specifications

(a) Livox MID360 sensor (b) Livox MID 360 sensor FoV

Figure 3.3: Livox MID 360 sensor and its FoV

3.1.4 Creality 3D Printer

To fabricate the necessary components for this project, a 3D printer is employed due to its ability to rapidly and reliably bring 3D CAD models to life. The specific model used is the Creality Ender-3 V3 KE. This printer offers the flexibility and precision to create custom parts tailored to the project's requirements.

Figure 3.4: Creality Ender-3 V3 KE 3D Printer

The maximum printing volume of the Ender-3 V3 KE is 220 x 220 x 240 mm, so all designed pieces must fit within these dimensions.

3.2 Software

The experiments conducted in this study were executed on a system running Ubuntu 22.04 LTS. The following sections introduce the main software modules used for the project development.

3.2.1 ROS2

The middleware used to interconnect all the subsystems is the Humble distribution of the Robot Operating System 2 (ROS2). ROS 2 is a middleware based on a strongly typed, anonymous publish/subscribe mechanism that allows message passing between different processes [45].

Additionally, other ROS tools were employed. The ROS TF package is used to handle transformations between different coordinate frames. For visualization purposes, RVIZ, which allows the visualization of 3D environments with robot position and sensor data as lidar, and *plotjuggler* for time series sensor data visualization, is used.

Finally, Gazebo is used as an integrated simulation environment within ROS 2. This tool allows for implementing a drone configuration similar to the real-life setup, supports sensor integration, and enables the testing of autonomous missions in a simulated environment before deployment in the real world.

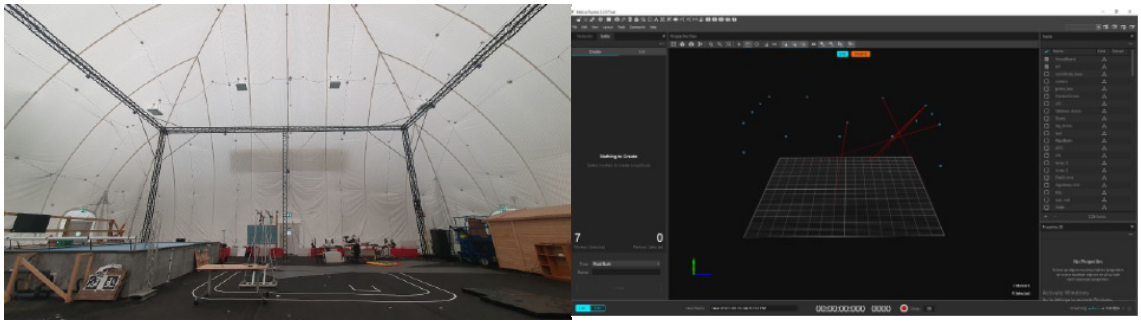
3.2.2 Onshape

To create computer-aided design (CAD) models for integration into the current drone frame, we used the browser-based CAD software system Onshape. Onshape provides a flexible and efficient platform for designing and modifying components, allowing seamless integration with the existing hardware setup.

3.2.3 Optitrack setup

To record contact data and conduct autonomous navigation missions in a controlled environment, we utilized DTU's Autonomous Systems Test Arena (ASTA). ASTA is a confined

space equipped with an OptiTrack motion capture system. The OptiTrack system records information from the drone's position, enabling the position loop for the PX4 to be closed.



(a) ASTA OptiTrack structure (b) OptiTrack software

Figure 3.5: ASTA facilities and OptiTrack system

3.3 Goal

As detailed in Section 1.1, the autonomous navigation stack of an autonomous flying robot is a complex system comprising various interconnected modules. However, these modules are prone to failures, which can occur due to factors such as sensor degradation or improperly fine-tuned algorithms, as demonstrated in Section 2.3.4. Such failures can lead to severe consequences, including mission failure due to collisions. This underscores the critical importance of addressing the problem outlined in this thesis.

The primary objective of this master's thesis is to ensure collision-tolerant navigation for flying robots. To achieve this, the thesis aims to analyze different contact detection methods and design and implement a novel contact detection system that can be integrated into an autonomous robot system stack, ensuring reliability in highly dynamic and obstacle-dense environments. The following contributions have been made to achieve this goal:

- Analysis of existing contact detection techniques.
- Mechanical design of a collision detection platform integrated into a flying robot.
- Design of a contact detection algorithm to detect collision events, including contact events, orientation, and impact magnitude.
- Integration of collision detection into a path planning and mapping algorithm.
- Design and implement a complete autonomous pipeline, incorporating the aforementioned contributions to enable collision-tolerant autonomous navigation.

The development of these contributions is explained in the following Section 4.

4 Methods

This section will explain the different methodologies developed to achieve the thesis goal of performing collision-tolerant autonomous navigation in obstacle-dense environments.

First, a collision platform is completely designed from scratch to be integrated into the drone frame, providing a platform to locate sensors that can help to detect collisions.

Then, the sensors integrated into the collision platform are used to develop an algorithm that provides information from the contact/collision events.

Finally, an autonomous navigation stack comprised of a mapping, planning, contact detection, and control module is defined to perform autonomous contact-based navigation.

4.1 Collision platform design

This section will explain the different approaches taken during the collision platform design. Initially, a test prototype of the platform is designed to validate the idea behind the integration between the sensors to detect collisions and the platform itself. Once the concept is validated and deemed feasible, the full-scale prototype is designed based on the parameters established in the initial prototype.

Based on the literature review in Section 2.4, some requirements must be defined to have a design guideline. The approach shown in Fig. 2.4a, where several antennas are located around the robot, has the disadvantage of leaving blind spots between antennas, which means that thin obstacles could cause the drone not to detect the collision. On the other hand, the approaches shown in Fig. 2.5a, Fig. 2.6, and Fig. 2.4b can detect contacts in the 3D and 2D space, respectively. So, regarding covered space, at least an entire 360-degree perimeter should be covered to detect all kinds of contacts in the 2D space.

As explained in Section 2.4.1, when talking about collision protection platforms, two kinds of approaches can be considered: rigid or flexible protection. One of the thesis goals is to integrate collision recovery into an autonomous navigation stack so that flexible protection makes more sense due to impact absorption and time to react to the collision while the impact is absorbed.

Considering the output variables that should be expected from the contact detection algorithm is crucial. The primary variable is the presence or absence of contact. However, this information alone may not be sufficient to recover from a collision. Therefore, understanding the contact orientation and the magnitude of impact absorption is equally important. This underscores the significance of our work in designing a more reliable collision recovery system.

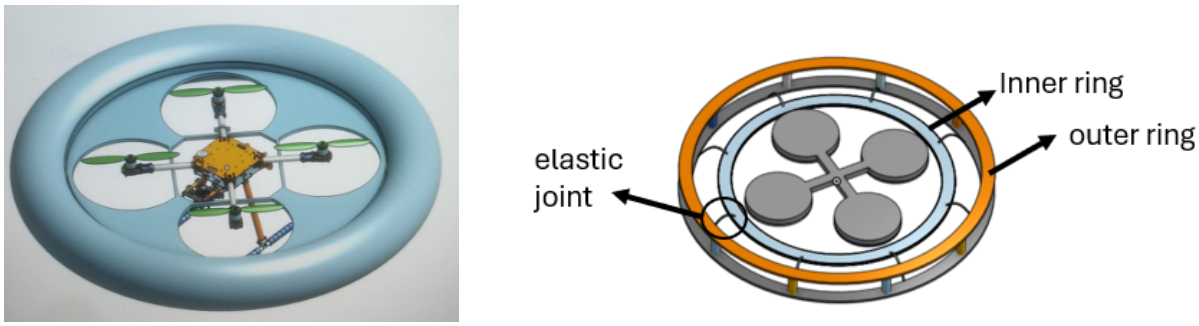
From the previous discussion, a requirement list can be defined, to sum up the guidelines for the concept design:

- The platform must cover a 360-degree perimeter to ensure no blind spots.
- The collision platform must incorporate an elastic structure to absorb impacts, preventing direct force transfer to the drone frame.
- The platform must reliably detect contact/collision occurrences. Additionally, it should provide the contact orientation and an impact magnitude.

4.1.1 Collision platform concept

Based on the guidelines defined in the previous section, some concepts have been analyzed before the platform was designed. Regarding 360º perimeter coverage, a continuous external platform would be necessary.

Several approaches can be implemented when talking about elasticity. One idea would be to have an external elastic platform (e.g., a toroid) with sensors integrated around it to measure the deformations, similarly to the approach shown in Fig. 4.1a simulating a float.



(a) External flexible toroid approach

(b) Inner and outer ring platform concept

Figure 4.1: Collision platform possible concepts

Another idea analyzed is to have an inner ring attached to the drone frame and an outer ring that can move radially freely over the inner ring so that some elastic joint could connect the inner and outer rings as done by [29]. A concept of the last idea is shown in Fig. 4.1b.

On the one hand, the toroid concept has the advantage that a collision from one side would not affect sensors located on the other side because the impact and surrounding zones would absorb the impact. However, that concept seems much more complex to integrate, and maintenance and repair might be challenging due to its continuous nature.

In contrast, the inner and outer ring concept would involve a collision causing a radial displacement of the external ring, which refers to the movement of the ring away from the point of impact. All the sensors would read the collision in different magnitudes based on the sensor position and displacement. Those magnitudes could be mapped to know the required contact, orientation, and displacement variables. The ring concept would be easier to integrate and maintain because components can be individually replaced if damaged. Additionally, regarding weight, the toroid would potentially be heavier than the ring design due to the continuous surface.

Given the advantages of the inner and outer ring concept in terms of integration, maintenance, and weight, this concept is chosen as a design guideline.

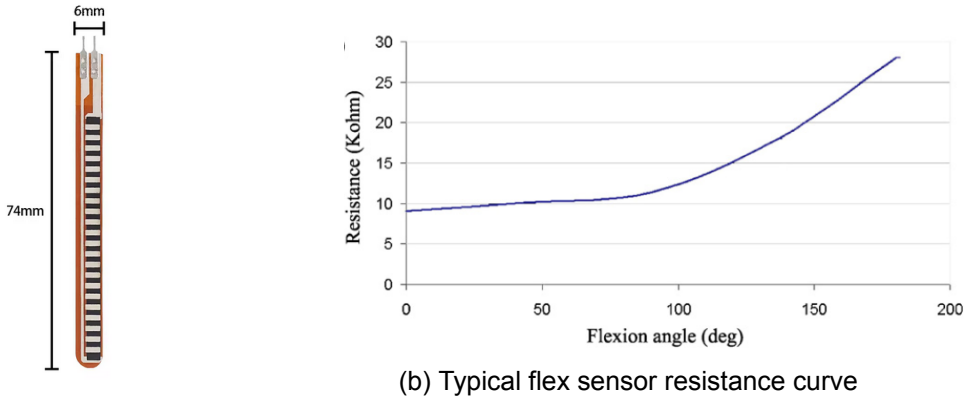
4.1.2 Collision platform sensors

Once the platform concept has been decided, knowing which sensor will measure the outer ring displacements is important for the design phase. Some sensor options can be used to measure displacements. Hall sensors, used in [23], are combined with a buckling spring mechanism to measure the spring's deformation. Another type of sensor used for contact detection has been the flex sensor, whose bending indicates some kind of contact. Another option would be to use optical sensors, which imply non-contact measurement, but it would need a clear line of sight and careful alignment.

After evaluating the available sensor options, flex sensors from Spectra Symbols [27] have been selected as the optimal choice for measuring outer ring displacements in the collision detection platform.

Flex sensors provide several key advantages:

- Simplicity and Cost-Effectiveness: Flex sensors are straightforward to integrate into the platform design and are relatively inexpensive compared to other sensor types.
- Reliable Contact Detection: These sensors effectively measure bending and stretching caused by collisions, providing accurate and immediate data.
- Ease of Installation and Maintenance: Flex sensors are easier to install and replace if needed.
- Compatibility with the Inner and Outer Ring Concept: The nature of flex sensors aligns well with the radial displacement mechanism of the inner and outer ring design.



(a) Spectra Symbol flex sensor [27]

(b) Typical flex sensor resistance curve

Figure 4.2: Flex sensor and its resistance curve

Finally, it is essential to determine the minimum number of sensors required for the platform design. One of the design objectives is to fulfill the requirements with a low-cost and functional contact detection system, so there is a trade-off between mechanical, electronic, and software complexity.

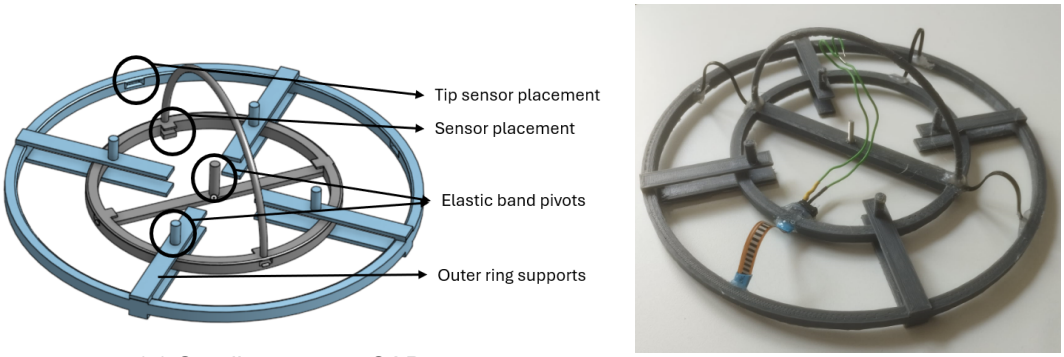
Each sensor measures displacement along its longitudinal axis. Therefore, at least two perpendicular sensors must capture displacement measurements in the 2D space. However, a configuration of four sensors is preferred to optimize platform dynamics and ensure symmetrical data collection. Given that the drone is a quad-rotor, four sensors evenly spaced around the circumference, connecting the inner and outer rings, will be employed.

The flex sensor can only measure bending in a specific direction. So, the neutral position of the flex sensors between the inner and outer rings should be bent. Then, the distance between the inner and outer ring will be half the sensor length, allowing it, by bending or stretching from that position, to measure within the whole operating range

4.1.3 Test prototype

To validate and enhance the inner and outer ring concepts explained in the previous section, several tests have been performed by designing a test platform. Fig. 4.3 shows the

test platform where the inner ring is assumed to be fixed to the drone frame while the outer ring moves when receiving a contact.



(a) Small prototype CAD

(b) Small prototype 3D printed

Figure 4.3: Small prototype design

As seen in Fig. 4.3, supports have been added to the outer ring to keep coplanarity with the inner ring. In Fig. 4.3b, one flex sensor is implemented with plastic ties to test sensor integration. However, the flex sensor and the plastic ties have different elasticities, so the flex sensor pushes the platform toward its axis more strongly than the ties.

Each sensor is bent when placed between the inner and outer rings in the neutral position, creating a force to restore them to their unbent state. Ideally, the resultant of these forces should balance each other out, keeping the outer ring centered relative to the inner ring. However, due to a slight misalignment in sensor placement, the resultant force might not be perfectly balanced, leading to the outer ring not being centered with the inner ring.

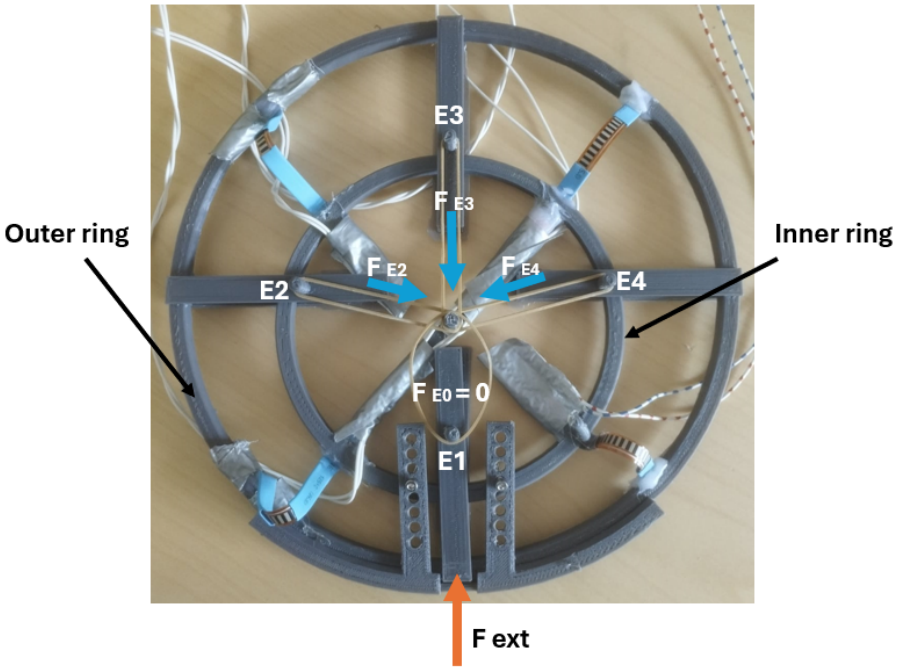


Figure 4.4: Small prototype elastic band forces due to an external force

To address this misalignment problem, an additional elastic joint between the inner and

outer rings must be implemented to ensure that the outer ring is always centered relative to the inner ring when recovering from a collision. Elastic bands have been selected as the elastic joint for several key reasons:

- Range of Displacement: Elastic bands can accommodate the necessary displacement range of 3.5 cm between the inner and outer rings and neutralize the forces generated by the flex sensors, maintaining the ring's alignment.
- Ease of Integration: The simplicity makes them a practical choice.
- Cost-Effectiveness: Elastic bands are relatively inexpensive.
- Consistency in Force Application: Elastic bands provide a consistent force proportional to their displacement, accurately maintaining the alignment of the outer ring.

The elastic bands are integrated using the pivots shown in Fig. 4.3 joining the inner and the outer ring as shown in Fig. 4.4.

As seen in Fig. 4.4, when an external force is applied to the outer ring in the E1 direction, and the inner ring is fixed, the elastic bands E2, E3, and E4 are stretched. This stretching creates a resultant force that restores the outer ring to its centered position relative to the inner ring once the external force is removed. At the same time, the elastic band E0 is not working.

With the test platform design performed in this section, there are some key features to consider for the full-scale prototype:

- **Coplanarity**: Ensure that the outer ring remains coplanar with the inner ring when suspended in the air. Use supports or additional structures as demonstrated in the test prototype.
- **Sensor Alignment and Placement**: Precisely align sensors to minimize misalignment and resultant force imbalances.
- **Elastic Joint Implementation**: Implement additional elastic joints (e.g., elastic bands) to ensure the outer ring remains centered relative to the inner ring.

4.1.4 Real prototype

Once the test prototype concept has been designed, tested, and solved the issues, the full-scale platform can be designed following the guidelines learned from the small platform design.

Sensor location

It is important to define the sensors' location before designing the inner ring attachment to the UAV frame. Having decided to use 4 flex sensors, there are 2 options to consider:

- Flex sensors aligned with the frame arms and propellers.
- Flex sensors located between frame arms.

The first approach could have several drawbacks. Locating the flex sensor closer to the propeller could cause measurement disturbances due to direct propeller airflow. In addition, a displacement of the sensor that is close to the propeller could cause damage to the sensor. So, the flex sensor will be located between the frame arms, keeping them as far as possible from the propellers.

Structure design

Once the sensor location has been decided, it is possible to design the ring platform.

The larger UAV radius is measured to define the inner platform's radius. This radius corresponds to the one from the drone center until the end of the propeller. To use the inner ring as propeller protection, the inner ring's radius is defined as the larger radius (from the center to the propeller end) plus one centimeter. The inner ring is divided into two pieces as shown in Fig. 4.5:

- **Inner ring arm**: Attached to the drone arm frame and connected through a screw union to the outer ring support.
- **Outer ring support**: Extension of the inner ring arm containing the sensor placement and a sliding slot to let the outer ring move radially in the 2D space.

Two inner ring arms are joined to the outer ring support by using two screws, as shown in Fig. 4.6.

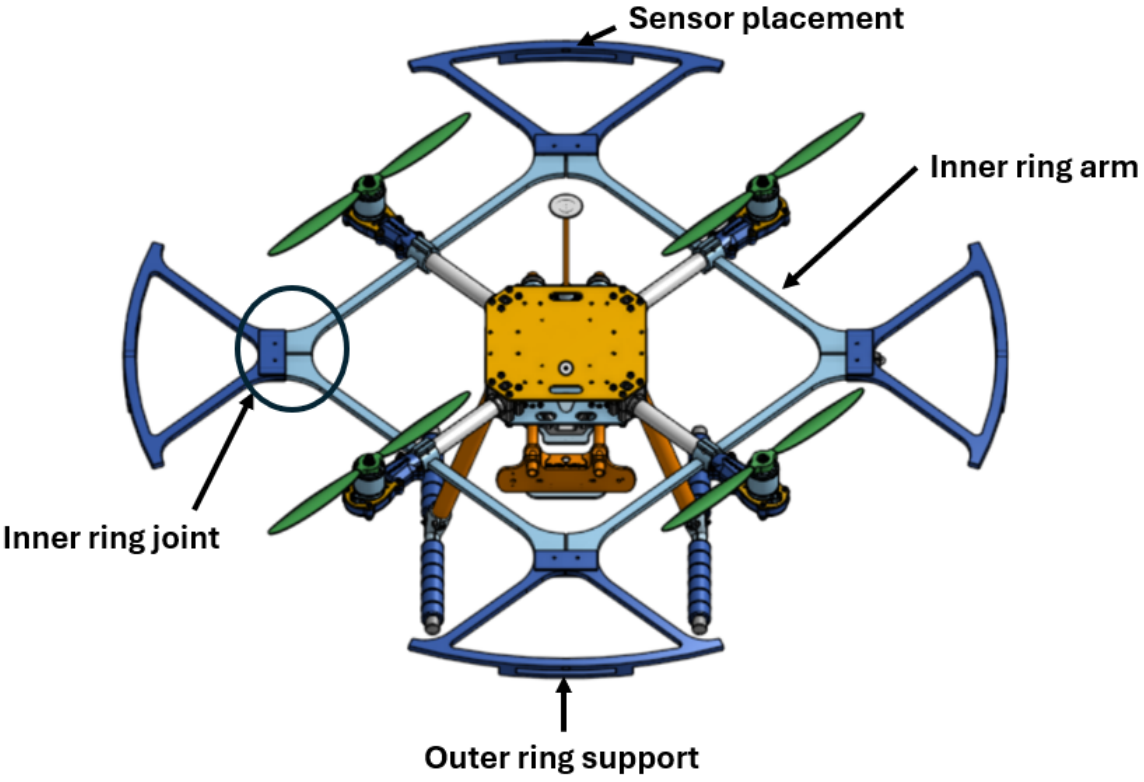


Figure 4.5: Collision platform inner ring

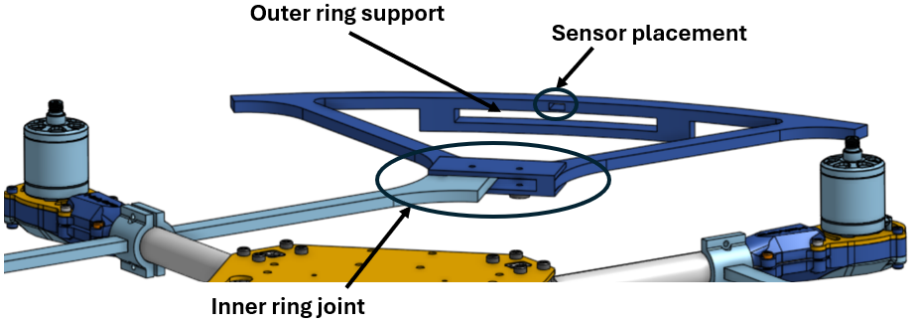


Figure 4.6: Collision platform inner ring joint

The outer ring is divided mainly into three pieces:

- **Outer ring sensor part**: Contains the sensor tip and the sliding support.
- **Outer ring no sensor part**: Joins the outer ring sensor parts by completing the full ring.
- **Outer ring joint**: Joins all the parts explained above.

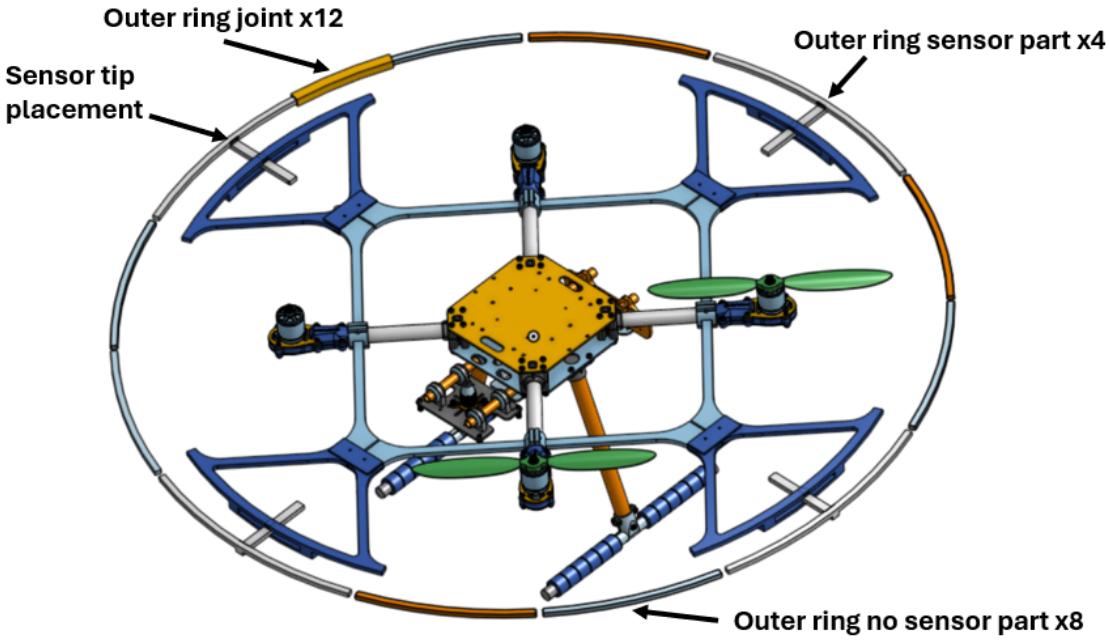
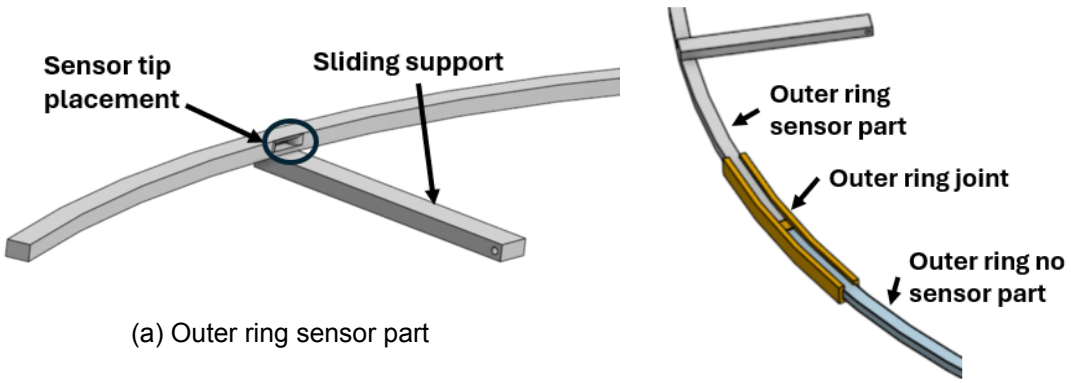


Figure 4.7: Collision platform outer ring



(a) Outer ring sensor part

(b) Outer ring joint

Figure 4.8: Outer ring parts detailed

Fig. 4.7 shows the different parts that create the outer ring. Fig. 4.8 illustrates a detailed view of the outer ring sensor part where the tip sensor is placed, the outer ring sliding support, and the outer ring joint.

The outer ring has a free 2d space movement over the inner ring with the current con-

figuration. To enable the outer ring to maintain a centered position when no contact is present, the elastic bands have to connect both rings so that a force is applied to keep the outer ring centered relative to the inner ring. An elastic band attachment has to be defined in both rings as shown in Fig. 4.9. The attachments are connected to the inner and outer ring, creating an elastic joint.

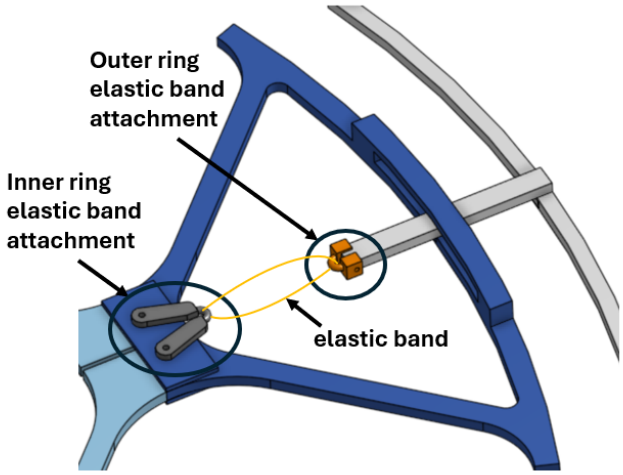


Figure 4.9: Elastic band joint between inner and outer ring

Once all the collision platform parts have been explained, it can be shown in Fig. 4.10 the final full-scale prototype assembly on the Holybro X500 frame.

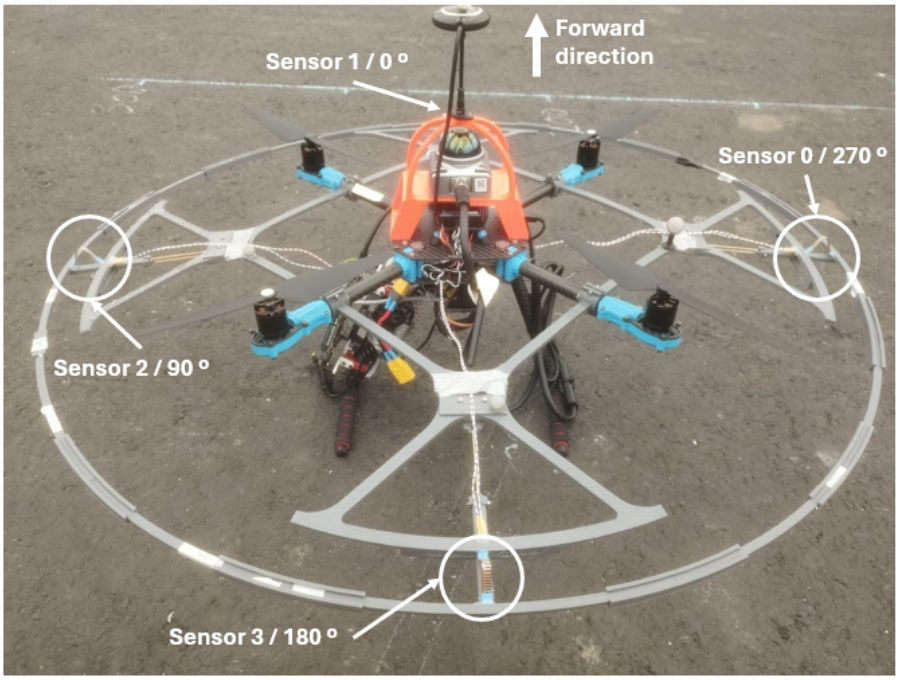


Figure 4.10: Collision platform integrated on the Holybro X500 frame

Sensor and platform integration
As explained in the previous section, the inner and outer rings contain slots to place the flex sensor joining both rings. Silicone has been used to keep the flex sensor fixed at

the slots. Heat-shrinkable shields are placed at the beginning and at the sensor's tip to protect the sensors from silicone.

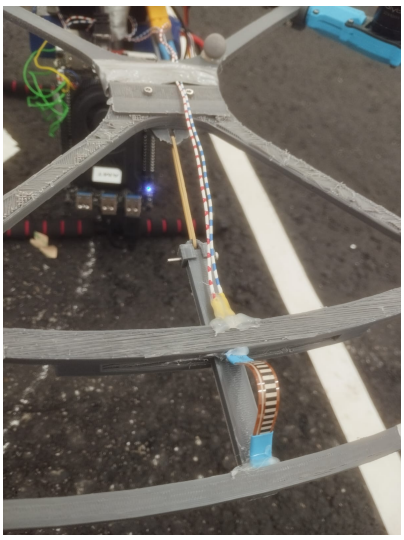


Figure 4.11: Flex sensor integrated into the collision platform

As can be seen in Fig. 4.11, the neutral sensor position is bent so that any outer ring radial displacement would bend or stretch the sensor, producing different sensor measurements

Sensor calibrator

As defined in Section 4.1, the output expected from the contact detection algorithm is the presence of contact, the orientation, and the outer ring displacement to account for the impact magnitude. A calibrator tool was designed to get data from different orientations and displacements, and it is shown in Fig. 4.12.

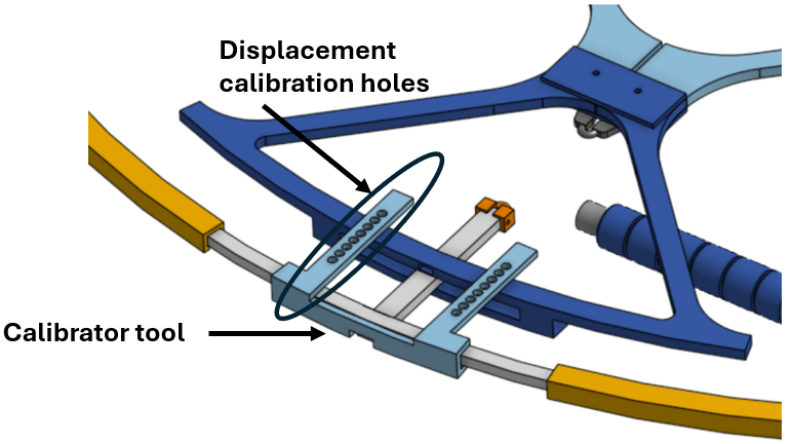


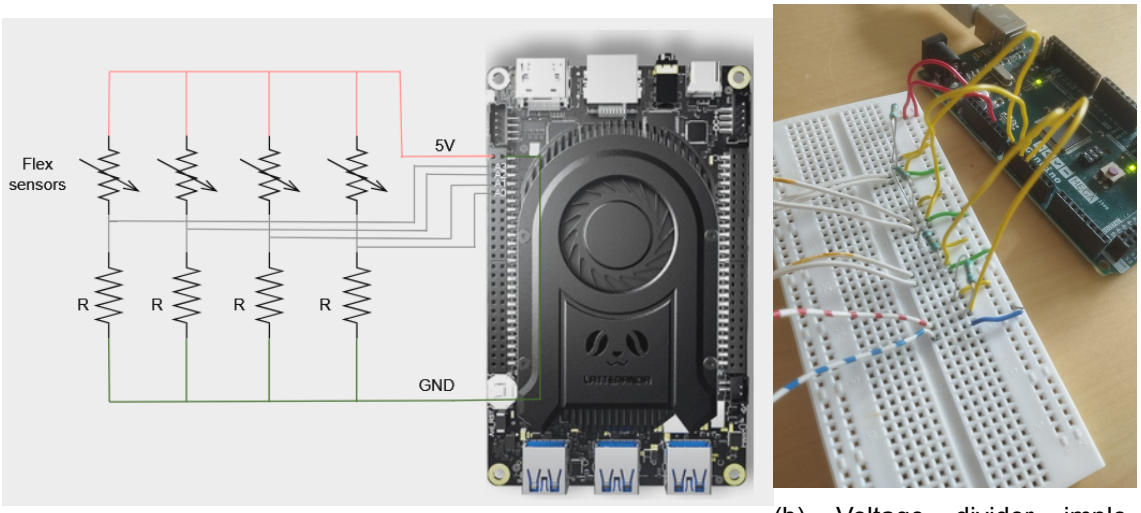
Figure 4.12: Angle and displacement calibrator tool

Data acquisition

The implementation for data acquisition consists primarily of an Arduino microcontroller integrated into the LattePanda platform. This microcontroller is responsible for capturing and processing the analog signals from the flex sensors.

Each flex sensor is connected in a voltage-divider configuration, as shown in Fig. 4.13a. In this setup, each flex sensor is paired with a resistor in series, forming a voltage divider. One end of the flex sensor is connected to the positive voltage supply, and the other is

connected to the ground. The point between the flex sensor and the resistor serves as the output of the voltage divider, which is fed into the Arduino's analog inputs.



(a) Flex sensor voltage divider schematic

(b) Voltage divider implemented

Figure 4.13: Flex sensor data acquisition system

The electronic implementation is shown in Fig. 4.13b. Under software control, the Arduino's built-in Analog-to-Digital Converter (ADC) reads these analog inputs sequentially. The ADC converts the analog voltage levels from the flex sensors into digital values.

A developed Python script on the LattePanda reads these digital values from the Arduino via a serial connection. The script is responsible for further processing and analyzing the data collected from the flex sensors. This setup ensures that the flex sensor data is captured accurately and efficiently for subsequent processing tasks.

4.2 Contact detection

In this section, the platform and data acquisition system described previously are utilized to derive insights from the data and to implement a contact detection technique for the collision platform. The main goal of the contact detection algorithm is to provide the following information from a contact or collision event:

- **Contact detected**: Boolean indicating if the platform is or is not in contact with an obstacle.
- **Contact angle**: Contact point orientation in the collision platform.
- **Contact displacement**: Displacement produced by the contact in the outer ring.

Data recording methods have been implemented to get the previous information. Then, the recordings have been organized into different datasets that will be used to train the contact detection algorithms.

4.2.1 Recordings

To ensure proper data processing and usage, it is crucial to define the methods for recording and saving data. This section outlines two primary data recording methods: using a calibrator tool to obtain ground truth contact events, orientations, and displacements within a static environment and employing an OptiTrack system to record flight collisions.

In both cases, recordings are saved within a CSV file containing the columns shown below:

sensor1	sensor2	sensor3	sensor4	collision_pred	angle_gt	displacement_gt
672.5	726.29	829.90	709.70	1	87.42	1.07

Table 4.1: Contact data recordings data structure from one sample

Each column is defined as:

- sensorx (x: sensor number): ADC sensor measurements.
- *collision_pred*: 1 when a contact event occurs and 0 when there is no contact.
- *angle_gt*: collision orientation in degrees.
- *displacement_gt*: outer ring displacement during the collision in centimeters.

Calibrator tool recordings

The calibrator tool generates the initial dataset in a static environment required to fit the first approach of the contact detection algorithm. Specifically, it is used to record data at various sensor orientations, including angles of 0°, 45°, 90°, 135°, 180°, 225°, 270°, and 315°. This comprehensive coverage ensures the algorithm can detect contacts from multiple directions.

For displacement measurements, the calibrator tool is designed to measure displacements ranging from 0 to 3.5 centimeters. However, due to the specific placement of the sensors, the effective displacement range is between 0 and 2.5 centimeters. Consequently, the recorded displacements are at intervals of 0.5 cm, 1 cm, 1.5 cm, 2 cm, and 2.5 cm.

So, by combining angle and displacement measurements, 40 combinations are recorded. In Fig. 4.14 is shown the methodology to record data for each position and each orientation.

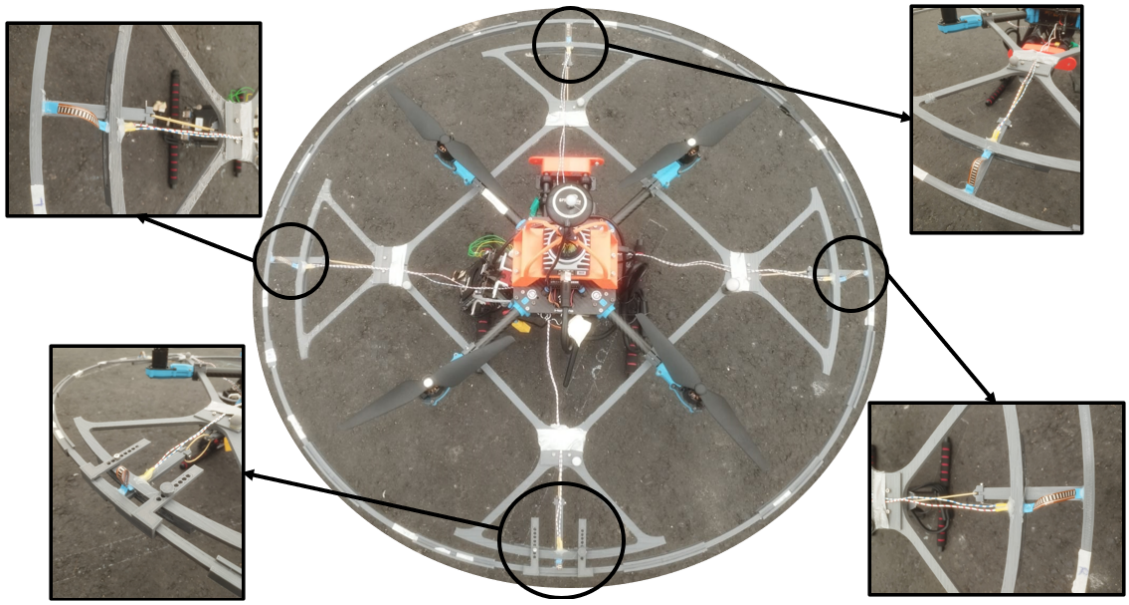


Figure 4.14: Collision platform calibration method: 180º orientation / 2.5cm displacement.

Fig. 4.15 shows the plots of the recorded data corresponding to each of the orientations and displacements with the calibrator tool. For each plot, the vertical red dashed line

divides the sections for each displacement (0.5, 1.0, 1.5, 2.0, 2.5 centimeters).

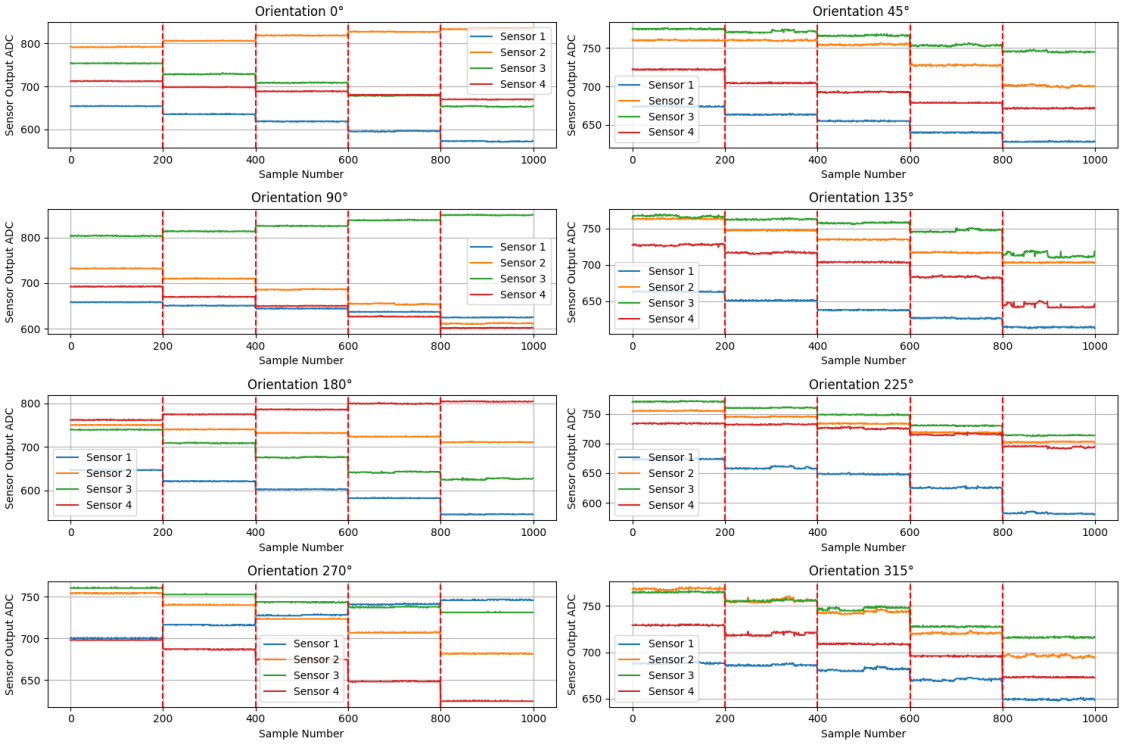


Figure 4.15: Calibration tool data recorded

OptiTrack System Recordings

The data obtained with the calibrator tool provides valuable initial insights and facilitates the development of an initial version of the contact detection algorithm. However, since this data is recorded in a static environment, it does not accurately represent real flight conditions. The OptiTrack system has been configured to capture orientations and displacements during manual flight collisions to address this limitation.

A planar surface is the collision target for the drone equipped with the collision platform. Both the drone's position and orientation, as well as the planar surface's position, are tracked using the OptiTrack system.

Since there is no direct implementation to capture ground truth contact events using the OptiTrack system, we utilize predicted contact events derived from the initial contact detection algorithm. This algorithm, fitted with calibration data, performs satisfactorily detecting contact events under flight conditions. The results of this algorithm and its performance are described in Section 5.2. However, as the calibration displacements start at 0.5 cm, there is a slight delay in detecting contacts during flight collisions. Therefore, manual fine-tuning of the contact predictions was performed on the dataset to align the predicted contact events as accurately as possible with the actual events.

To calculate the contact orientation, the following steps are performed:

1. Calculate planar surface normal: Using the positions of 2 OptiTrack points.
2. Calculate UAV Yaw: Extracted from the orientation quaternion.
3. Determine Contact Angle: The angle difference between the UAV's yaw vector and the planar surface's normal vector is calculated to represent the orientation of the contact.

To calculate the displacement during a collision, the following steps are performed:

1. Initial Contact Detection: When a contact event is predicted, the initial distance between the drone and the planar surface is calculated and stored.
2. Continuous Displacement Calculation: The displacement is the difference between the initial distance (from the first contact event) and the current distance. It is calculated until there is no contact prediction.

Fig. 4.16 shows the data recorded from a manual flight colliding the drone against the test wall. The first plot shows the ADC measurements from the sensors; the second plot shows the contact (1) and no contact (0) events; the third plot shows the drone orientation relative to the wall over the flight; and the last plot shows the outer ring displacement during contact events.

4.2.2 Dataset creation

Once the flight data have been recorded by using the calibrator tool or the OptiTrack, as explained in the previous sections, three different datasets have been created.

Calibration dataset

The calibration dataset is the one created with the recordings using the calibrator tools as explained in Section 4.2.1. The platform's static and flight data without collisions is also recorded and added to this calibration dataset to contain contact (with angle and displacement) and no contact events.

The data structure used in this dataset is shown in table 4.1.

This dataset was created to train the first contact detection model and to label contact events for the flight dataset. The calibration dataset will also compare models trained with calibration data versus models trained with real flight data.

Flight Discrete Dataset

This dataset was created from recordings using the OptiTrack system, as explained in Section 4.2.1. All flight recordings were merged into a single dataset comprising 11 manual flights. Each flight includes several collisions distributed around the drone, as illustrated in Fig. 4.16.

The data was split to train and evaluate the models to ensure that the evaluation used unseen data. Of the 11 flights, 9 were used for training the model, while the remaining 2 were reserved for evaluation.

Due to the predominance of non-contact data in the training set, the dataset was balanced to ensure an equal number of contact and non-contact data points. The training dataset and the ground truth labels are depicted in Fig. 4.17.

The data structure used in this dataset is shown in table 4.1.

Flight Time-Series Windows Dataset

In addition to the discrete dataset, another dataset was created specifically for training models with time-series information. Each training sample in this new dataset consists of a time-series window containing the last x sensor measurements (where x represents the window size). The label for each time-series sample corresponds to the last data point within the series. As well as for the Flight Discrete Dataset 4.2.2, the same 9 flights were used for training and the remaining 2 for evaluation.

The data structure used in this dataset is the same as the one described in Table 4.1, but each sensor consists of an array with the last sensor measurements of the length of the window size.

Figure 4.16: OptiTrack data recording

4.2.3 Inside the data

Fig. 4.15 shows data obtained in a controlled environment using the calibrator tool. As observed in the left column plots, when a collision occurs in the orientation where the sensor is placed (0, 90, 180, or 270 degrees), the sensor increases its ADC value (indicating bending) as the collision displacement increases. Simultaneously, the other sensors re-

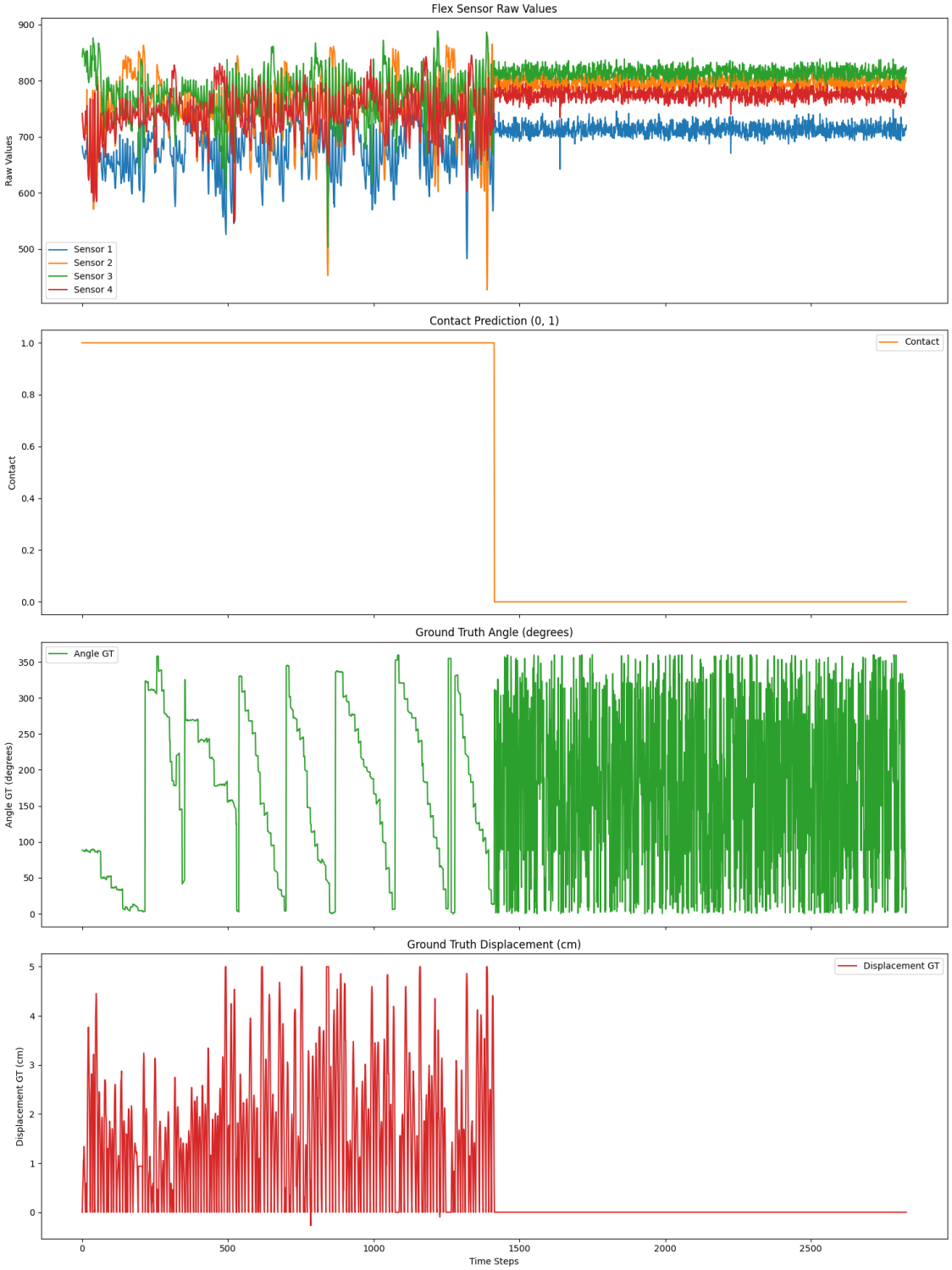


Figure 4.17: Training dataset recorded with OptiTrack from 9 manual flights

duce their ADC values because they are being stretched. This indicates that the torsion movement experienced by sensors placed perpendicularly to the collision orientation acts as if they are being stretched. A clear pattern emerges from these plots, allowing the definition of a simple logic to detect contacts based on sensor orientations during collisions.

However, the right column plots in Fig. 4.15 show collisions in the diagonal orientations

between sensors. In this case, all sensors experience torsion movements (because the force is not aligned with their longitudinal axes), reducing their ADC values as the collision displacement increases. This torsion movement increases the non-linearity characteristic of the flex sensor.

In contrast, Fig. 4.16 presents recordings from a manual control flight captured by the OptiTrack system. The most significant difference between the recordings from OptiTrack and the calibration tool is the time-series nature of the contact data. In the OptiTrack recordings, contacts can be visually identified as peaks in the sensor data. Notably, larger peaks correspond to greater displacements.

4.2.4 Modeling

Flex sensors inherently exhibit non-linear behavior due to the variability in their temperature resistance coefficient with changing operating temperatures. Additionally, the platform's design induces torsional movements, further contributing to the non-linearity.

Given the unprecedented nature of this particular problem—predicting contacts, angles, and displacements from four evenly spaced flex sensors, an approach using Feed Forward Neural Networks has been chosen. The selection of FFNNs is driven by their simplicity and effectiveness in handling various data types, including sensor inputs. By leveraging the simplicity and adaptability of FFNNs, this approach aims to establish a baseline solution for predicting contacts, angles, and displacements from flex sensor data. In addition, as FFNN does not leverage time-series information, a simple RRN consisting of an LSTM block has also been implemented for the angle and displacement model.

Due to the nature of the different variables to predict (contact event, angle, and displacement), two separate networks will be designed.

- **Contact detection model**: It will predict contact events using a binary classifier trained with contact and no-contact data
- **Angle and Displacement model**: It will focus on predicting angles and displacements, but only when a contact event is detected. This specialized network will be trained on contact event data.

By separating these tasks, the models can be more specialized and potentially more accurate in their predictions.

Contact detection model

For contact detection, a simple Feed Forward Neural Network has been implemented. The size of the input layer is determined by the data used to train the model, ensuring it appropriately captures the features from the flex sensors. Subsequently, one or more hidden layers with a specified number of units are selected, each followed by a ReLU activation function (as described in Eq. 2.10). ReLU helps introduce non-linearity into the model, allowing it to learn complex patterns in the data.

Given that the task is a binary classification problem (contact vs. no contact), the output layer uses a sigmoid activation function (as described in Eq. 2.9). This function maps input values to a range between 0 and 1, representing the probability that the prediction belongs to one class or the other.

The Binary Cross Entropy Loss function (as described in Eq. 2.13) is employed to train the model. This loss function is commonly used in binary classification tasks, as it effectively measures the difference between the predicted probabilities and the actual binary labels, guiding the optimization process to improve model accuracy.

Angle and displacement model

For angle and displacement modeling, the first approach is the implementation of a simple Feed Forward Neural Network, shown in Fig. 4.18. The input layer size is determined by the data used to train the model, ensuring it effectively captures the features from the flex sensors. The network architecture comprises a series of hidden layers divided into two modules.

The first module contains a hidden layer that extracts general features relevant to angle and displacement predictions. Subsequently, three sets of hidden layers are defined to specialize in predicting angles (sine and cosine) and displacements, respectively. Each hidden layer is followed by a ReLU activation function, introducing non-linearity and enabling the model to learn complex patterns.

For the angle and displacement predictions, the output layer is linear and does not use any activation functions. This allows the network to produce continuous output values directly corresponding to the predicted angles and displacements.

In contrast, a more sophisticated architecture leverages time-series information using Long Short-Term Memory networks. This second approach maintains a structure similar to the FFNN, with the primary modification replacing the general hidden layer with an LSTM block. The input to the LSTM block will be a time series from each sensor instead of discrete data points for the FFNN. The output of the LSTM block is then fed into the three sets of specialized hidden layers, as described in the first approach, to predict angles (sine and cosine) and displacements.

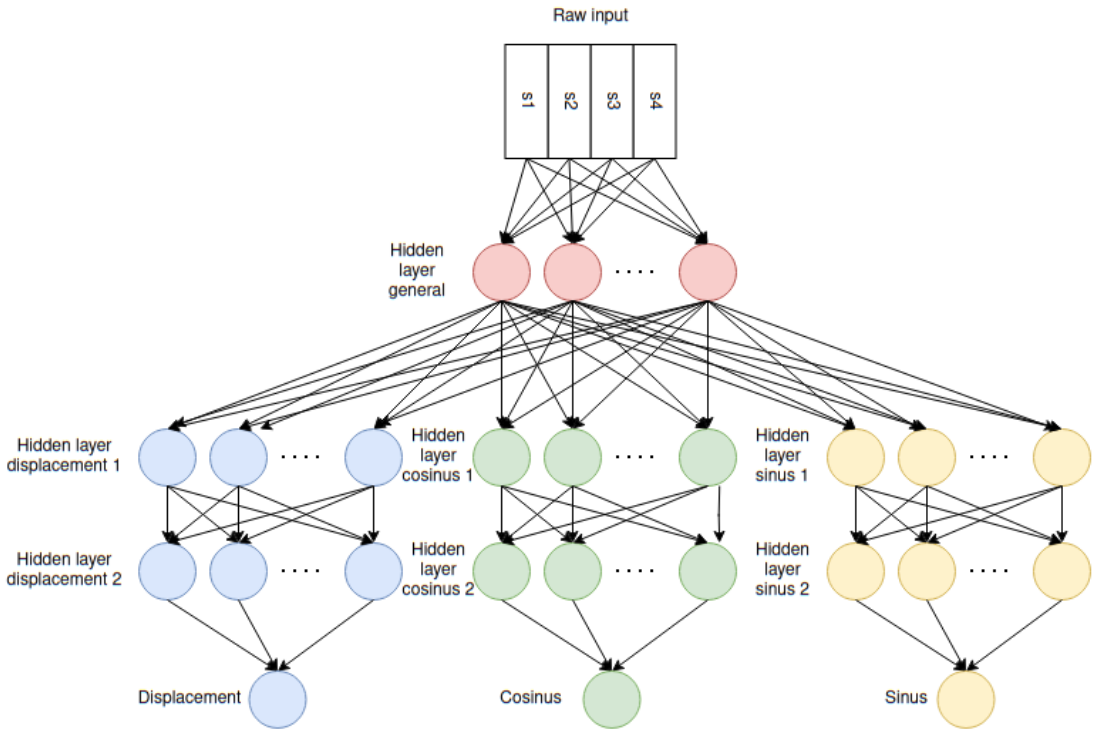


Figure 4.18: FFNN architecture for angle and displacement prediction

The training process employs a combined loss function to optimize the model. The displacement loss is calculated using the Mean Squared Error (MSE), as described in Eq. 2.12. The angle loss is determined using a custom Sine Cosine Distance loss, as detailed in Eq. 2.14. The final loss function is the sum of these two components:

$$\text{Angle Displacement Loss} = \text{MSE (displacement)} + \text{Sine Cosine Distance (angle)} \quad (4.1)$$

This combined loss function ensures that the model accurately predicts both angles and displacements by minimizing the respective errors in a balanced manner.

4.3 Autonomous navigation

This section presents the modules involved in the autonomous navigation mission. The mission consists of sending a goal waypoint to the autonomous system, and it has to reach it. Along the path to the goal, there will be some obstacles.

For the autonomous mission, the collision platform designed in Section 4.1 is integrated into the drone frame as shown in Fig.4.10. The contact detection algorithm implemented in Section 4.1 outputs contact variables such as contact detection, contact angle, and contact displacement. These variables are fed into the autonomous navigation stack to enable contact-based navigation.

Two main missions are implemented in this project. On the one hand, blind contact-based navigation does not use any perception sensor; it uses information from the collision platform to react to obstacles colliding. On the other hand, lidar and contact information are combined to navigate through lidar visible and not visible obstacles.

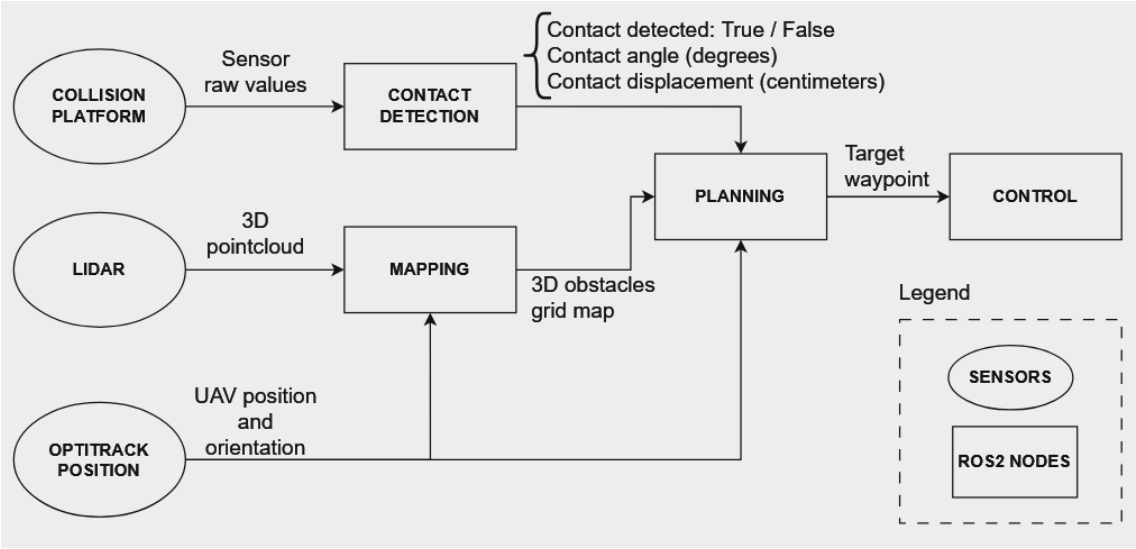


Figure 4.19: Autonomous pipeline proposed to solve contact-based navigation

To perform the autonomous mission, the following ROS2 nodes are involved:

- **Contact detection** node: Takes the sensor raw values, runs the model designed in Section 4.2.4, and publishes the contact information: contact detected, contact angle, and contact displacement.
- **Mapping** node: Takes the point cloud from the lidar and generates a 3D grid map containing occupied cells.
- **Planning** node: Takes the map and contact information to plan a path to the goal waypoint.

- **Offboard control** node: Takes the waypoint from the planning node and sends it to the PX4 controller to control the drone in position mode.

A summary of the autonomous pipeline implemented is shown in Fig. 4.19.

4.3.1 Mapping

As the mapping subsystem is not the focus of this project, the existing ROS2 package *Octomap Server* [12] has been used to create the 3D grid map based on the point cloud generated by the lidar sensor.

From the tunable parameters, the map resolution has been set to 0.88 m as the collision platform diameter. A lower value would involve a higher resolution map, leading to higher computation time. A higher value would involve losing resolution from obstacles.

4.3.2 Planning

This subsystem has been created to provide insights about the collision platform performance when integrated into a real autonomous mission. The reason for the simple solution taken is to evaluate the collision platform integration with the simplest planning setup.

The planning algorithm used is the *RRT** geometric path planning algorithm, which is supported by environment information provided by the Mapping subsystem (explained in Section 4.3.1) in the form of a 3D grid map. This 3D map is stored internally in the planning node using a search space algorithm. This algorithm enables the integration of the lidar-generated grid map with information from the contact detection algorithm, forming a comprehensive search space. The search space facilitates the RRT* algorithm finding obstacle-free paths and ensures that calculated paths are continually evaluated and updated as new lidar or contact data is received. This dynamic updating mechanism allows for efficient and safe navigation through changing environments.

The parameters from the planning algorithm are the explained in Table 4.2

Parameters	Explanation
Contact Map Resolution	Square box side length map contact obstacle [m]
Octomap resolution	Square box side length map lidar obstacle [m]
RRT Search Space Range	3D volume to search for planning solutions [m]
RRT q	Distance between RRT waypoints [m]
RRT Goal	Mission goal waypoint coordinates [m]
Step Back	Waypoints number to recover from collision [m]

Table 4.2: Path Planning parameters

Fig. 4.20 shows the logic to plan paths to a goal waypoint.

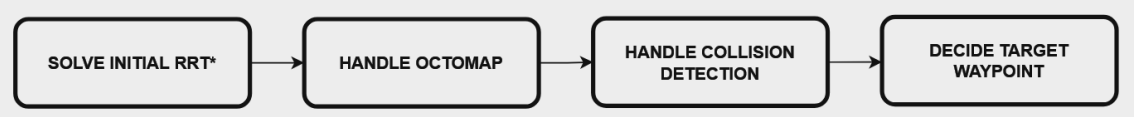


Figure 4.20: Path planning submodules proposed combining lidar 3D grid map and contact information for obstacle avoidance.

First, we will go through the ROS2 subscriber functions, which receive the input variables to the planning node:

- The ***octomap_pc2_callback*** function receives the grid maps created by the mapping subsystem. Every time a new grid map is received, this function updates the search space with the new obstacles detected by the lidar sensor. The flag *octomap_received* is set to True every time a new map is received.
- The ***contact_callback*** function receives information from the contact detection node. This information contains information about whether there is a collision, the collision angle, and the collision displacement. Whenever a collision is received, the initial predicted angle includes the contact in the map. The contact obstacle world frame position is calculated as follows:

$$\mathbf{o}^{uav} = \begin{bmatrix} \cos(\theta_c) \cdot r_d \\ \sin(\theta_c) \cdot r_d \\ 0 \end{bmatrix} \tag{4.2}$$

$$\mathbf{o}^{w} = \mathbf{R}^{w}_{uav} \cdot \mathbf{o}^{uav} + \mathbf{d}^{w} \tag{4.3}$$

  where θ_c is the collision angle in degrees, r_d is the radius of the drone, $\mathbf{o}^{uav}$ is the position of the obstacle in the drone frame, $\mathbf{d}^{w}$ is the position of the drone in the world frame, $\mathbf{R}^{w}_{uav}$ is the rotation matrix from the drone frame to the world frame, and $\mathbf{o}^{w}$ is the position of the obstacle in the world frame. Once the contact is detected, the *collision_platform* flag is set to True to activate the collision recovery method from the planning logic.
- The ***ekf_px4_pose_callback*** function receives the drone position and orientation from the Extended Kalman Filter implemented in the PX4 controller.

These input variables are used in the main loop shown in Fig. 4.20, which runs at a fixed rate. The logic implemented in the main loop is sequential, meaning several conditional statements decide which target waypoint should be sent to the offboard control node.

The following sections will explain each submodule's functionality:

SOLVE INITIAL RRT*
Every time the path planning node is launched, it is initialized by calculating the initial path from the starting point to the goal coordinates. If the lidar grid map is available, it will be used to plan a free-obstacle trajectory; if no grid map is available, it will find a solution in an empty search space.

HANDLE OCTOMAP
When a new lidar obstacle grid map is received from the Mapping node, this submodule verifies if any obstacles intersect with the previously calculated path. If such collisions are detected, the submodule recalculates an obstacle-free path using the RRT* algorithm from the current drone position, incorporating the latest obstacle information

HANDLE COLLISION DETECTION
When a contact detection message is received, this module handles the collision, guides the drone to a collision-free state, and recalculates a new path. The collision response logic involves returning to previous waypoints within a no-contact state.

Here's how the collision handling process works:

1. **Collision Detection**: Upon detecting a collision, the target waypoint is immediately decremented by the *Step Back* parameter, directing the drone to move backward to the previous waypoints. At this point, the *collision_recovering* variable is set to True, indicating that the drone is in a collision recovery state.

2. **Collision Recovery**: In the collision recovery state, there are two possible scenarios:
    - **Persistent Contact**: If the drone continues to detect contact while moving backward, the target waypoint will keep decreasing, sending the drone further back to previous waypoints until a no-contact state is reached.
    - **No-contact State**: Once the drone moves to a waypoint where no contact is detected, the *collision_recovering* variable is set to False.
3. **Post-Collision Handling**: After resolving the contact event, which involves detecting the collision, recovering from it, and reaching a no-contact state, the detected contact is saved on the map. Subsequently, a new path is calculated to avoid the obstacle that caused the collision.

DECIDE TARGET WAYPOINT
Once the lidar grid map and contact detection information have been processed, this submodule determines the appropriate target waypoint to send to the offboard control node.

- Normal Operation: When no collision events are detected, the submodule increments the target waypoint once the current waypoint is reached, ensuring continuous progression toward the goal.
- Collision event handling: If a collision event is detected, the submodule selects the target waypoint based on the output from the HANDLE COLLISION DETECTION submodule, which manages the drone's recovery from collisions.
- Mission Completion: Upon reaching the goal waypoint, the submodule recognizes that the mission is complete and ceases further waypoint updates.

4.3.3 Offboard control

The control strategy selected for the experiments is position control mode from PX4, explained in Section 2.1. This can be achieved by using the drone state given by the OptiTrack system. This simple control approach has been selected because the focus of the thesis has been the design of the collision platform and its integration into an autonomous mission.

To control the drone, a ROS2 node has been written to receive the target trajectory waypoint in the ROS2 world frame from the path planning algorithm, explained in Section 4.3.2, converted to the PX4 world frame, and sent to the PX4 controller at a fixed rate.

5 Experiments

In this section, various experiments and metrics are explained to evaluate the methods implemented in the previous Chapter 4.

First, we evaluate the collision platform and the contact detection models developed to predict contact events, contact orientation, and contact displacement.

Next, we implement the collision platform and contact detection algorithm into an autonomous navigation stack to perform contact-based navigation. This implementation is done in two stages: initially without any perception sensor and subsequently by combining contact data with lidar mapping.

5.1 Durability Assessment of the Collision Platform

In this experiment, we evaluate the durability of the collision platform through a series of rigorous tests. The primary objectives are to determine how well the platform withstands repeated impacts and to identify any components needing replacement after extensive testing. This experiment includes all contact flight recordings and autonomous mission contact-based experiments.

5.1.1 Assessment Criteria

The criteria for evaluating platform durability are as follows:

- **Structural Integrity**: After each set of tests, the collision platform was inspected for any signs of damage or wear.
- **Component Replacement**: The experiment tracked whether any components of the collision platform required replacement or repair throughout the testing phase due to fatigue or fracture.

5.2 Contact detection

In this experiment, we evaluate the performance of different contact detection models in predicting contact events, contact orientation, and contact displacement. The table summarizing the experiments uses abbreviations to define the parameters for better fit within the page. These abbreviations are explained below.

Regarding the datasets used in the training, the following abbreviations are used:

- Calibration Dataset (C)
- Flight Discrete Dataset (FD)
- Flight Time-Series Windows Dataset (FW)

Two different types of architectures are evaluated:

- Feed-Forward Neural Network (F)
- Long Short Term Memory + Feed-Forward Neural Network (L+F)

Three input vectors have been used along the experiments:

- Raw values (R): Sensor raw values stack in a vector of size [4 x 1].

- Differences (D): Differences between the raw values of each sensor are calculated and stacked into a vector of size [6 x 1].
- Time-series windows (W): Sensor raw values stacked in a vector of size [4 x windows size].

The following sections explain the experiments done for both the Contact detection model and the Angle and Displacement model. In both cases, the following training parameters are used:

- Learning Rate = 0.001
- Batch Size = 32

5.2.1 Contact model

The model used to predict contact events is detailed in section 4.2.4. The architecture includes only one hidden layer, with the number of hidden units being a parameter for experimentation (H units).

Initial contact detection models are trained using the calibration dataset (C) obtained in a static environment, as described in Section 4.2.2. Subsequently, additional contact detection models are trained using the flight discrete dataset (FD) described in Section 4.2.2.

For training the model, given that these datasets do not include time-series information, the raw values (R) and differences (D) inputs are used.

The table below summarizes the experiments conducted for the contact detection model:

Model	1	2	3	4	5	6
Dataset	C	C	FD	FD	FD	FD
Architecture	F	F	F	F	F	F
Input	R	D	R	R	D	D
H units	16	16	16	32	16	32

Table 5.1: Contact model experiments

5.2.2 Angle and Displacement model

Two models are evaluated to predict angles and displacements. The first model, described in section 4.2.4, is a Feed-Forward Neural Network with several adjustable parameters. For these experiments, the network will have a single general hidden layer, with the number of units in this hidden layer varied as an experimental parameter (G Units). Additionally, specific modules for predicting cosine, sine, and displacement values will each have one hidden layer. The number of hidden units in these specific modules will also be varied as an experimental parameter and consistent across all branches (S Units).

The FFNN model will be trained using both the calibration dataset (C) 4.2.2 and the flight discrete dataset (FD) 4.2.2. This model does not utilize time-series information, so the raw values (R) and differences (D) inputs will be used for training.

The second model, described in section 4.2.4, comprises an LSTM module and an FFNN to output the cosine, sine, and displacement predictions. For the LSTM module, one hidden layer will be used in all experiments, with the hidden size varied as an experimental parameter (G Units). In the FFNN part, the specific modules for predicting cosine, sine,

and displacement values will each have one hidden layer, with the number of hidden units set to match those of the LSTM module and consistent across all branches (S Units).

The LSTM plus FFNN model will be trained exclusively using flight data, as it requires time-series information. The input to this model will be the time-series data (FW) from each sensor, with the last measurements defined by the window size (W size), which will be varied as an experimental parameter.

The following table summarizes the angle and displacement model experiments:

Model	1	2	3	4	5	6	7	8	9	10	11	12
Dataset	C	C	FD	FD	FD	FD	FW	FW	FW	FW	FW	FW
Architecture	F	F	F	F	F	F	L+F	L+F	L+F	L+F	L+F	L+F
Input	R	D	R	R	D	D	W	W	W	W	W	W
W size							10	10	20	20	30	30
G Units	16	16	16	32	16	32	16	32	16	32	16	32
S Units	32	32	32	64	32	64	16	32	16	32	16	32

Table 5.2: Angle and Displacement model experiments

5.2.3 Evaluation metrics

All models will be evaluated using the two flight test sets not included in the training data. This approach ensures that the model's performance is assessed on unseen data. Also, models trained with calibration data will be evaluated using a separate calibration test set. This dual evaluation allows us to analyze the impact of the training dataset on the model's generalization capabilities.

Regarding contact events predictions, as a binary classification task, the metrics explained in section 2.5.6 will be used (Accuracy, Precision, Recall, F1-Score, and Area Under the Curve).

In contrast, angle and displacement predictions are regression tasks, so the metrics explained in section 2.5.6 are used (MSE, MAE, RMSE). In addition, other metrics such as standard deviation, maximum error, median, and interquartile range are included.

5.3 Autonomous Contact-based Navigation

In this experiment, we integrate the collision platform and contact detection algorithm, previously tested flying in manual mode as described in Section 5.3, into an autonomous mission. There will be two different missions. The first mission relies solely on contact information for navigation, while the second mission integrates LiDAR data and contact information to enhance obstacle detection and avoidance.

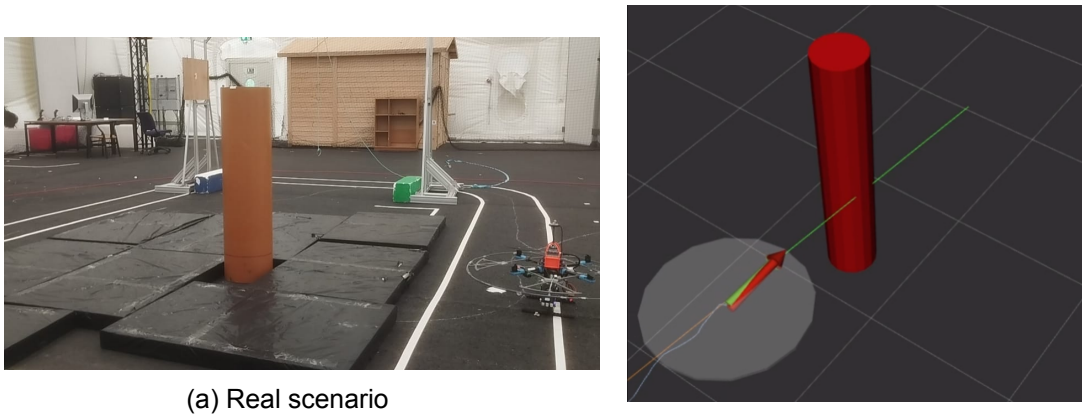
The autonomous navigation stack is tasked with reaching a designated waypoint in both missions. An obstacle is deliberately placed midway along this path, undetectable in advance by the drone. The primary objective is to assess the collision platform's ability to detect and recover from unexpected collisions, incorporate the detected contact information into its map, and replan its trajectory to avoid obstacles.

From the collision detection algorithm, the planning and mapping module will utilize only the contact prediction and the predicted contact angle. Although displacement data is not used in these experiments, it holds potential value for fine-tuning the collision recovery logic. By leveraging displacement information, the system could adjust its recovery behavior to be smoother or more aggressive, depending on the collision event.

For both missions, planning parameters are configured as follows:

- **Contact Map** and **Octomap Resolution**: Both are set to 0.88. This resolution matches the collision platform diameter, ensuring the minimum collision area is accurately represented in the 3D grid map. This prevents the drone from colliding again with the same point.
- **RRT q**: Set to 0.1 to ensure slow and controlled navigation.
- **Step Back**: Set to 3, meaning that after a collision, the drone is sent back to three waypoints. This ensures the drone moves at least 30 centimeters away from the obstacle to reach a non-collision state.
- **Search Space Range** and **RRT Goal**: These parameters are adjusted for each experiment scenario.

5.3.1 Blind Contact-based Navigation



(a) Real scenario

(b) Visualization scenario

Figure 5.1: Blind Contact-based Navigation experiment scenario

As explained, this experiment will show the ability to perform blind navigation (no perception sensors available) based on contact information from the collision platform.

The object used in the experiment is a pipe with a diameter of 0.35 centimeters and a height of 1.5 meters. It will be located in the middle of the initial path, causing a collision.

Fig. 5.1a shows the scenario used to test blind contact-based navigation. The drone is at the starting point (right in the image) and it is sent to go forward to the left in the image. The brown pipe is placed in the middle of the path to cause the collision. Fig. 5.1b shows the mission visualization. The drone is defined by the red arrow describing the drone position and orientation, and the white circle defines the collision platform diameter. The red cylinder is the pipe obstacle. Then, the green line is the RRT* calculated path, the blue line is the drone's real path, and the orange line is the target waypoint.

The path planning parameters used for the experiment are shown in Table 5.3.

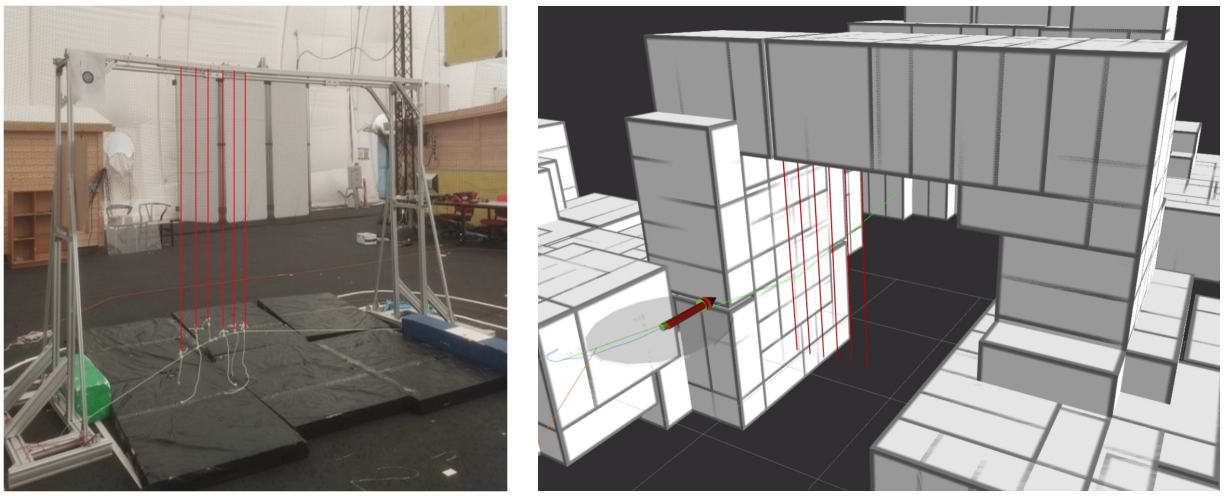
Planning parameters	Values
RRT search space range [x, y, z]	[[-1, 5], [-2, 2], [0.5, 1.0]]
RRT goal [x, y, z]	[3, 0, 1]

Table 5.3: Planning parameters for the Blind contact-based navigation experiment

5.3.2 Lidar and Contact-based Navigation

In this experiment, we integrate the collision platform and contact detection algorithm into an autonomous mission that utilizes a lidar sensor. The lidar sensor, a key component in many autonomous navigation systems, provides the drone with a detailed map of its surroundings. Still, it has limitations in detecting thin obstacles, such as thin wires.

The collision obstacle is a goal-shaped object with a height of 2 meters and a width of 3 meters. This goal's left half is obstructed by thin tensioned vertical wires, which the lidar sensor cannot detect. The objective is to demonstrate how the collision platform can compensate for the lidar's lack of resolution. For that purpose, the initial trajectory will be defined to send the drone against the wires while the lidar maps the environment.



(a) Real scenario

(b) Visualization scenario

Figure 5.2: Lidar and Contact-based Navigation experiment scenario

Fig. 5.2a shows the real scenario used to test the lidar and contact-based navigation. The tensioned wires in the left part of the goal-shaped obstacle can be seen highlighted in red. Fig. 5.2b shows the mission visualization. The drone is defined by the red arrow describing the drone position and orientation, and the white circle defines the collision platform diameter. In this case, the white boxes are the grid map created with the lidar sensor. For visualization purposes, the thin wires are shown by thin red cylinders occupying the left of the goal-shaped obstacle. Then, the green line is the RRT* calculated path, the blue line is the drone's real path, and the orange line is the target waypoint.

The path planning parameters used for the experiment are shown in Table 5.4.

Planning parameters	Values
RRT search space range [x, y, z]	[[-1, 5], [-3, 0.5], [0.5, 1.0]]
RRT goal [x, y, z]	[5, 0, 1]

Table 5.4: Planning parameters for the Lidar and Contact-based navigation experiment

5.3.3 Evaluation metrics

The metrics to evaluate the collision platform integration into an autonomous mission will be the following:

- Mission completed: Whether the mission is completed.

- Number of collisions: Number of collisions needed to complete the mission.
- Contact event duration: Time the collision platform predicts to be in contact with the obstacle.
- Reaction time: Time to react since the collision has been detected.
- Angle error: Error between predicted and true collision angle used to include the contact box in the 3D grid map.

6 Results and Analysis

6.1 Durability Assessment of the Collision Platform

The collision platform has demonstrated robust structural integrity throughout all flights, recordings, and autonomous missions, enduring over 200 collisions without damaging its parts. The platform remained intact even during severe collisions or when the drone hit the floor due to position control loss. Occasionally, such impacts might slightly displace the platform but can be easily realigned manually.

In terms of replacements, no components were broken during the tests. The only maintenance required involved the elastic bands, which were replaced after one month of use. The elastic bands, responsible for keeping the outer ring centered relative to the inner ring, lost their elasticity due to constant stretching. Replacing the worn elastic bands with new ones effectively resolved this issue.

6.2 Contact detection

This section shows the results obtained from the different experiments done for the contact models and the angle and displacement models.

The results are presented with a color gradient to make the result analysis easier. For the contact model, the metrics used will be green when values are higher and red when values are lower. In contrast, the metrics are errors for the angle and displacement models, so the lower errors will be greener, while the highest errors will be red.

6.2.1 Contact model

First, the contact models trained with calibration data (models 1 and 2 from table 5.1) are evaluated against the calibration test set. The results are shown in Fig. 6.1.

Model	1	2
Dataset	C	C
Architecture	F	F
Input	R	D
H units	16	16
Accuracy	1	1
Precision	1	1
Recall	1	1
F1 Score	1	1
AUC	1	1

Figure 6.1: Contact detection models trained and evaluated with calibration test dataset

As can be seen in Fig. 6.1, all the metrics have the highest score for both models 1 and 2. This means that, within a static environment (no flight), the models trained can detect whether or not the platform is receiving a collision. These results could be expected as the training and test data were recorded with the calibrator tool so the model could learn patterns in a controlled environment.

The results from all the models evaluated with the flight datasets are shown in Fig. 6.2

Fig. 6.2 presents the results of contact prediction during various flights. It's crucial to note that, as detailed in Section 4.2.1, Model 1, trained with calibration data, was used to

Model	1	2	3	4	5	6
Dataset	C	C	FD	FD	FD	FD
Architecture	F	F	F	F	F	F
Input	R	D	R	R	D	D
H units	16	16	16	32	16	32
Accuracy	0.99	0.69	0.99	0.98	0.95	0.97
Precision	0.98	0.11	0.82	0.74	0.47	0.58
Recall	0.71	0.93	0.93	0.94	0.92	0.91
F1 Score	0.82	0.19	0.87	0.82	0.62	0.71
AUC	0.85	0.8	0.96	0.96	0.94	0.94

Figure 6.2: Contact detection models evaluated with the flight test dataset

label the contact events in the flight dataset. This led to Model 1 exhibiting high Accuracy and Precision values. However, it also showed a lower Recall, indicating that the model predicted too many False Negatives and missed many contact-labeled events.

Models trained using the differences in sensor values (Models 2, 5, and 6) as input features generally perform poorly.

Conversely, Model 3, trained with raw sensor values as inputs and contacts from the flight dataset, outperformed all metrics. This indicates its ability to predict contact events accurately and avoid mispredictions.

These results underscore the critical importance of training models with data resembling real-world scenarios. This approach enables the model to effectively generalize, a key factor in its performance.

6.2.2 Angle and Displacement model

First, the Angle and Displacement models trained with calibration data (models 1 and 2 from Table 5.2) are evaluated against the calibration test set. The results are shown in Fig. 6.3.

The calibration test dataset experiment shows that using sensor differences as input to train the model (Model 2) does not perform well compared with the other model. In contrast, the model trained using raw sensor values (Model 1) has lower errors, indicating that the model can learn patterns to detect contact angles and displacements in static conditions.

Then, all the models described in the table 5.2 are evaluated with the flight test dataset. The results are shown in Fig. 6.4

The same pattern analyzed previously for the calibration dataset is present for the flight dataset; models trained with sensor differences as inputs (Models 2, 5, and 6) tend to perform worse than the ones trained with discrete raw values or raw values within time series.

It's clear that models trained solely with calibration data (Models 1 and 2), without flight information, struggle to generalize when evaluated with flight datasets. This underscores the crucial importance of training with data closely resembling real conditions.

A clear trend regarding angle predictions has been discovered in how they perform better by increasing the model complexity. For the FFNN Models 3 and 4, increasing the general/specific hidden units (G Units/S Units) from 16/32 to 32/64 improves slightly in terms of MAE, MSE, and RMSE.

Model	1	2
Dataset	C	C
Architecture	F	F
Input	R	D
W size		
G Units	16	16
S Units	32	32
Angle MAE	0.23	1.44
Angle MSE	0.12	65.40
Angle RMSE	0.34	8.09
Angle StdDev	0.25	8.01
Angle Median	0.15	0.21
Angle IQR	0.26	0.23
Angle Max	2.95	75.46
Displacement MAE	0.01	0.03
Displacement MSE	0.00	0.00
Displacement RMSE	0.02	0.04
Displacement StdDev	0.02	0.04
Displacement Median	0.01	0.02
Displacement IQR	0.02	0.04
Displacement Max	0.08	0.26

Figure 6.3: Angle and displacement models trained and evaluated with the calibration test dataset.

Model	1	2	3	4	5	6	7	8	9	10	11	12
Dataset	C	C	FD	FD	FD	FD	FW	FW	FW	FW	FW	FW
Architecture	F	F	F	F	F	F	L+F	L+F	L+F	L+F	L+F	L+F
Input	R	D	R	R	D	D	W	W	W	W	W	W
W size	-	-	-	-	-	-	10	10	20	20	30	30
G Units	16	16	16	32	16	32	16	32	16	32	16	32
S Units	32	32	32	64	32	64	16	32	16	32	16	32
Angle MAE	28.91	43.81	16.61	16.57	19.37	19.91	11.06	12.69	9.76	9.64	9.90	9.31
Angle MSE	2042.13	3924.93	804.91	780.18	1049.96	1103.78	286.73	356.01	238.84	190.08	214.70	204.23
Angle RMSE	45.19	62.65	28.37	27.93	32.40	33.22	16.93	18.87	15.45	13.79	14.65	14.29
Angle StdDev	34.73	44.78	23.00	22.48	25.97	26.60	12.82	13.96	11.98	9.86	10.80	10.84
Angle Median	15.49	22.02	6.52	7.73	8.80	8.57	7.03	7.34	5.86	6.69	6.45	5.41
Angle IQR	29.29	53.37	17.16	18.27	21.37	22.69	8.55	13.39	9.91	8.97	10.39	9.00
Angle Max	177.42	178.90	132.43	171.40	177.34	159.46	81.15	64.22	134.35	55.21	64.87	59.32
Displacement MAE	1.04	1.16	0.58	0.62	0.75	0.76	0.63	0.64	0.62	0.64	0.73	0.66
Displacement MSE	1.74	2.78	0.63	0.75	0.92	1.00	0.78	0.71	0.68	0.72	0.88	0.80
Displacement RMSE	1.32	1.67	0.80	0.86	0.96	1.00	0.88	0.84	0.83	0.85	0.94	0.89
Displacement StdDev	0.85	1.67	0.79	0.86	0.96	0.99	0.86	0.82	0.80	0.83	0.91	0.88
Displacement Median	0.94	0.80	0.46	0.50	0.63	0.62	0.45	0.51	0.47	0.50	0.61	0.51
Displacement IQR	1.04	1.58	0.90	0.98	1.25	1.18	0.90	0.98	0.95	0.97	1.23	0.94
Displacement Max	4.14	8.08	4.05	5.29	4.39	4.58	4.09	3.68	3.46	3.13	3.45	3.97

Figure 6.4: Angle and displacement models evaluated with the flight test dataset.

Remarkably, models that incorporate time-series information, such as the LSTM module in the second approach (Models 7 to 12), show a substantial improvement regarding angle prediction. These models achieve a near 50% reduction in the predicted angle errors such as MAE, MSE, and RMSE. Between the time-series trained models, windows size 20 and 30 seem to perform better than windows size 10, with which not enough information could be captured.

However, angle and displacement predictions have demonstrated that they do not follow the same trend for model design. As seen in Fig. 6.4, the best displacement prediction model is Model 3, the simplest one trained with flight data. This contradicts the trend in angle prediction, where increasing model complexity led to better results. These un-

expected findings challenge our assumptions of creating a combined neural network to predict angles and displacements leveraging shared features.

Model 9 stands out in correctly predicting both angles and displacements. Figures 6.5 and 6.6 illustrate the comparison between the true and predicted angles and displacements derived from the flight test dataset by using Model 9.

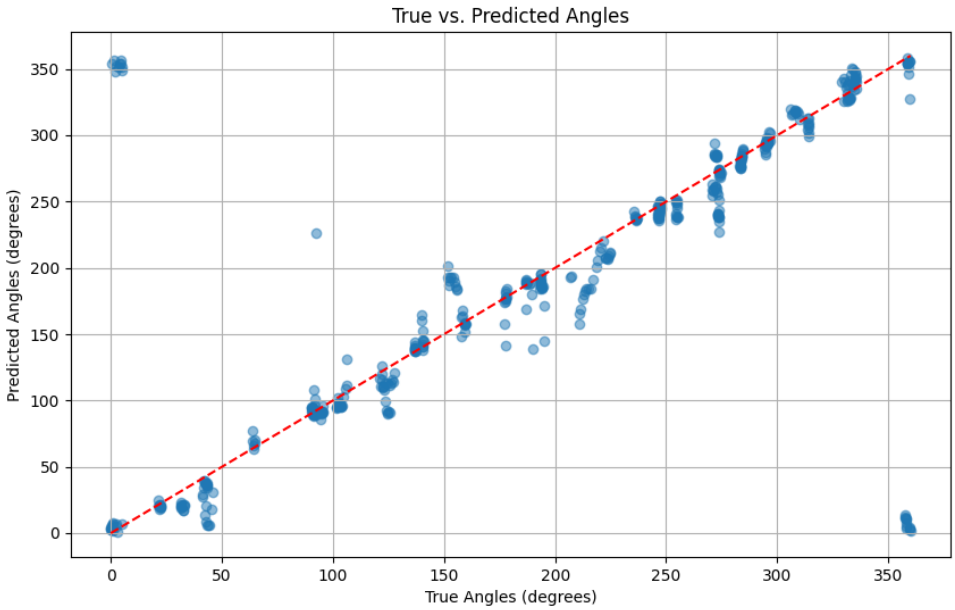


Figure 6.5: True versus predicted angles from the flight test dataset by Model 9.

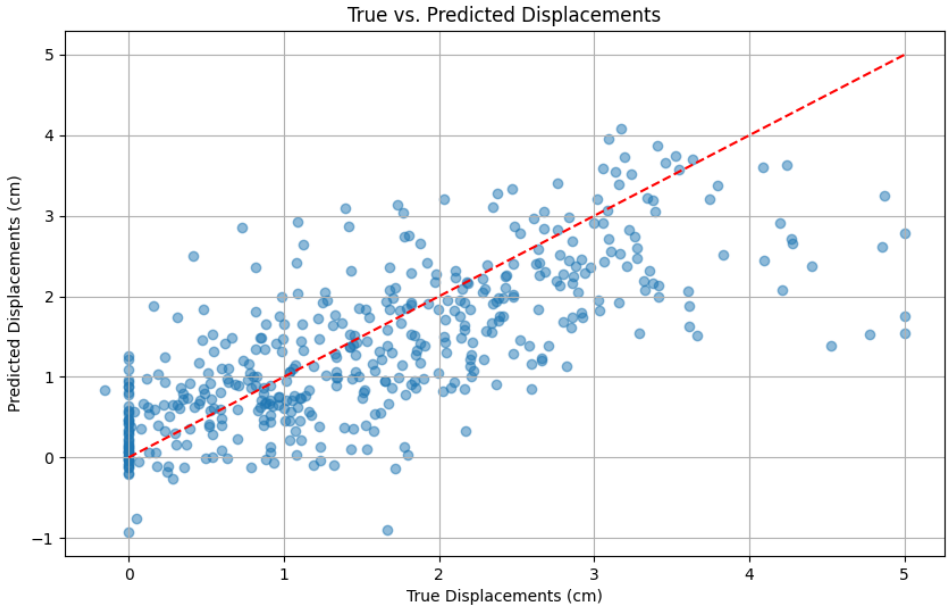


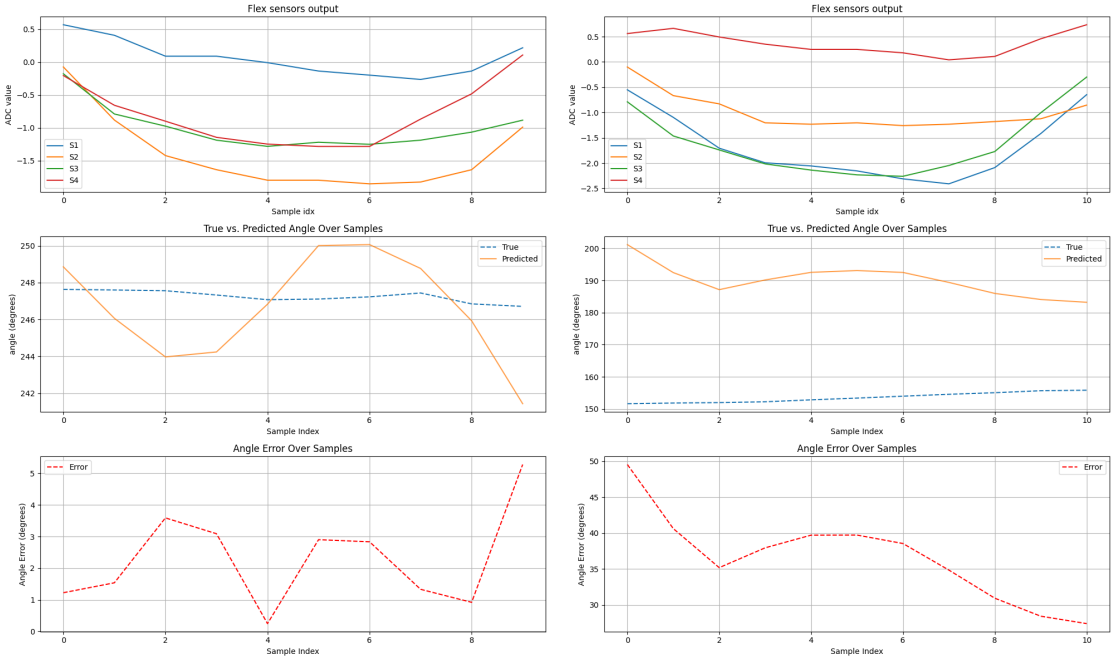
Figure 6.6: True versus predicted displacements from the flight test dataset by Model 9

Fig. 6.5 shows how the model can almost fit the predicted versus true collision angles into

a line along the entire UAV perimeter, with just a few visible outliers. This means that no significant errors are present. It can also be noticed that for angles 0 and 360, predictions of 360 and 0, respectively, are done, not being a problem because of the circular nature of the angles.

Fig. 6.6 demonstrates that displacement prediction is not working as well as the angle prediction. A trend can be seen in the predicted versus true displacements that fit the red dashed line (representing the best model performance). However, the values are more dispersed than the angle predictions.

Regarding the prediction during the contact events, from the initial contact to the contact release, Figures 6.7 and 6.8 leverage the time-series nature of a contact event. During the contact event, the first plot shows the flex sensor output, the second plot compares true versus predicted angle or displacements, respectively, and the third plot shows the prediction error.



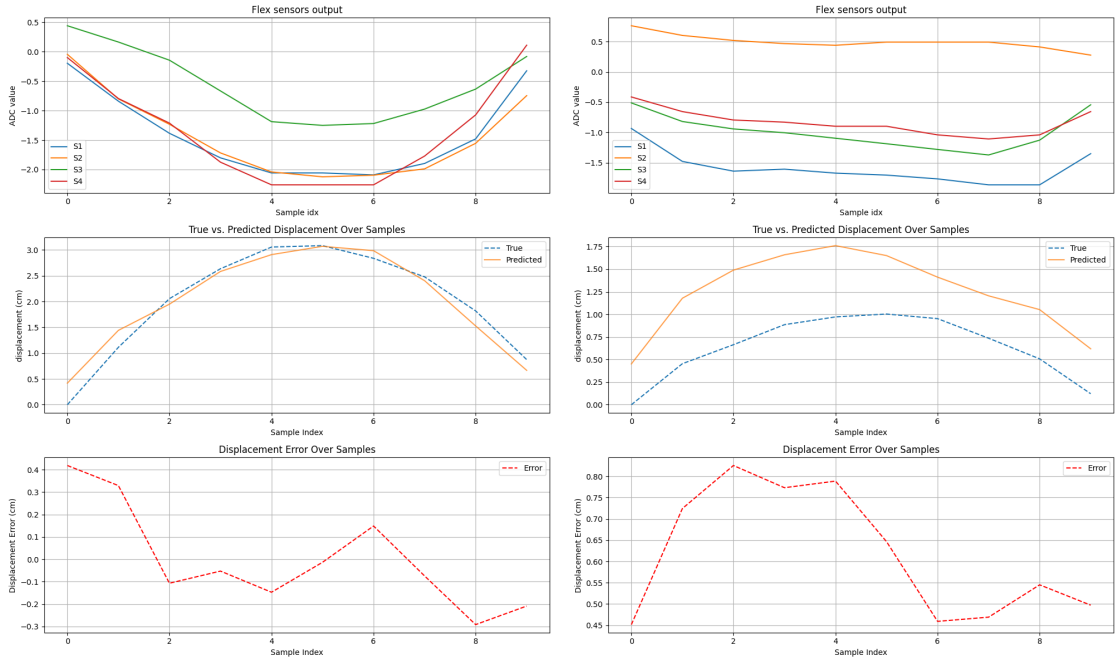
(a) 250 deg contact event

(b) 200 deg contact event

Figure 6.7: Different contact events showing flex sensor values, true vs predicted angle, and angle error

Fig. 6.7 shows how the angle prediction evolves during the contact event. Fig. 6.7a shows how the predicted angle oscillates around the true angle due to the sensor values' abrupt changes from one sample to another, leading to abrupt predictions but still around the true label with an error around 5 degrees. In contrast, 6.7b shows a more significant angle prediction error of 50 degrees at the beginning, decreasing until 30 degrees at the end of the contact event.

Fig. 6.8 shows how the displacement prediction evolves during the contact event. Fig. 6.8a shows how the predicted displacement almost imitates the displacement pattern from the true label, demonstrating its potential to predict continuous displacements. However, Fig. 6.8b show the correct predicted displacement pattern, increasing and then decreasing but with a clear offset to the true displacement.



(a) Contact event 1

(b) Contact event 2

Figure 6.8: Different contact events showing flex sensor values, true vs predicted displacement, and displacement error

This offset in the displacement prediction could be caused by the method defined to label displacements using OptiTrack in the training dataset explained in Section 4.2.1. This is due to using a prediction from another model to label the duration of the contact event, which is crucial to label the displacement over the contact. A delay in the initial contact prediction causes a minor total displacement along the contact event in the displacement labels.

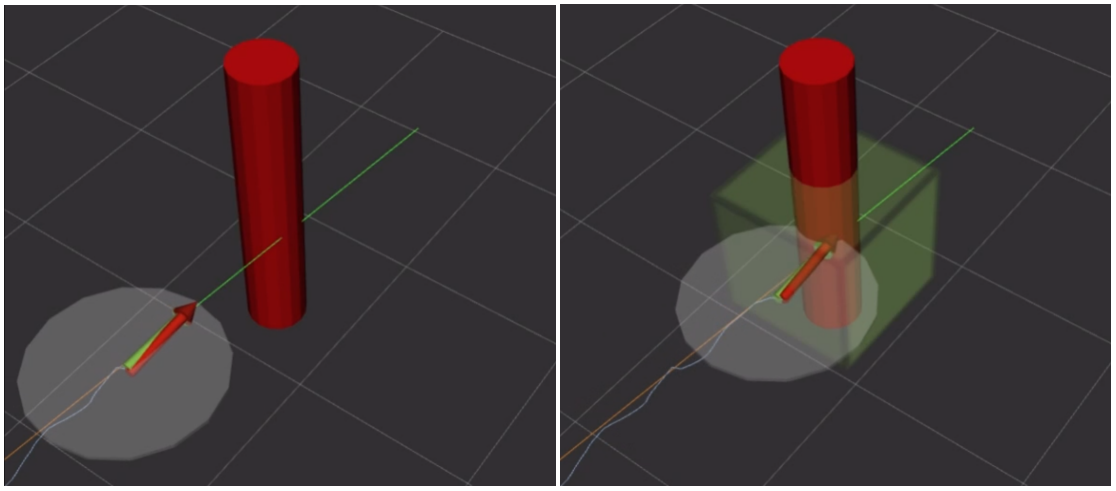
6.3 Autonomous Contact-based Navigation

In this section, the results related to the different autonomous missions performed are described.

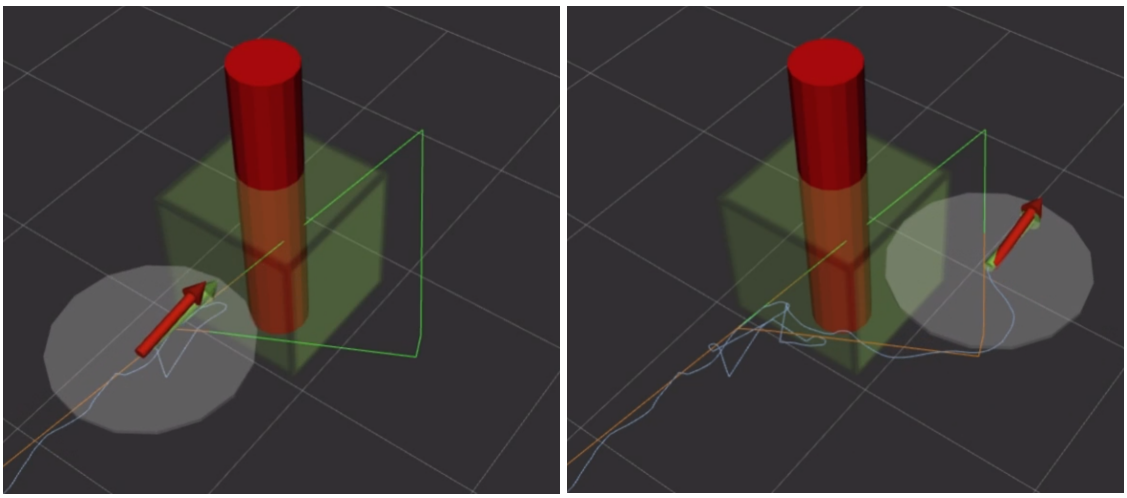
6.3.1 Blind Contact-based Navigation

Fig 6.9 shows the different mission stages. Fig. 6.9a shows the initial RRT* path (green line) calculated to collide the drone against the pipe. Fig. 6.9b captures the collision event between the drone and the pipe, showing how the contact is recorded in the 3D grid map (green box). Fig. 6.9c illustrates the collision recovery stage, where the drone moves backward by the waypoints defined by the *step back* parameter (three waypoints in this case) until it reaches a non-contact state. The figure also shows the recalculated RRT* path that avoids the newly marked contact box in the grid map. Finally, Fig. 6.9d shows the drone successfully avoiding the previously collided obstacle and proceeding towards the goal waypoint.

As seen in Fig. 6.9, the mission has been successfully completed by incorporating contact information into the navigation. Table **??** shows the mission metrics to evaluate the behavior. Due to the obstacle not being too wide, only one collision was enough to avoid after replanning a new trajectory.



(a) Initial path (b) Collision and contact map updated



(c) Collision recovering (d) Recalculated path

Figure 6.9: Blind Contact-based Navigation experiment showing planning and mapping stages

Metrics	Results
Mission Completed	Yes
Number of collisions	1
Contact event duration (s)	0.271
Reaction time (s)	0.037
Angle predicted (deg)	358.21
Angle ground truth (deg)	10.97
Angle error (deg)	12.76

Table 6.1: Blind contact-based navigation real experiment results

The contact event has been predicted within 0.271 seconds, which can be shown in Fig. 6.10. This time reflects how many time the navigation stack was aware of the collision. Since the collision has been received, the collision recovering logic has been triggered after 0.037 seconds, which means that the system can react fast to the collision.

Finally, the error in predicting the collision angle was 12.76 degrees, sufficient to accurately include the collision in the map and avoid the obstacle after a single collision.

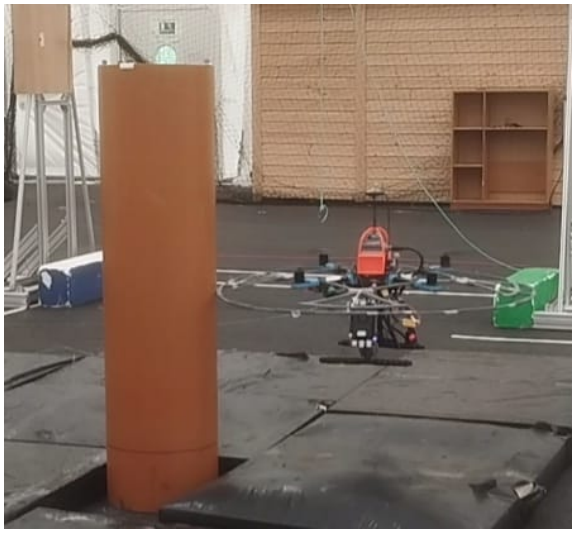


Figure 6.10: Collision event during the Blind Contact-based Navigation experiment.

6.3.2 Lidar and Contact-based Navigation

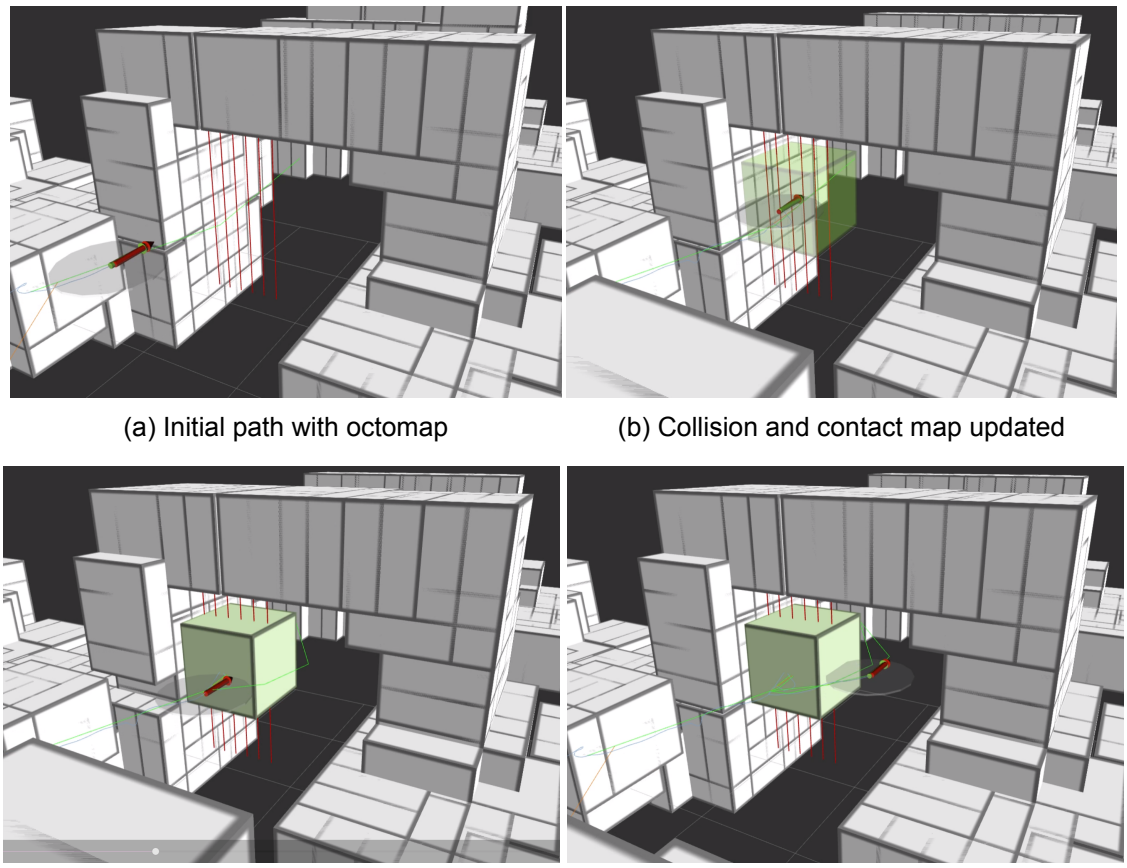
Fig. 6.11 shows the different mission stages. Fig. 6.11a shows the initial RRT* path (green line) calculated to collide the drone against the wires. The 3D grid map (white boxes) created by the LiDAR sensor is also shown, in which the wires are not included because the LiDAR did not detect them. Fig. 6.12b captures the collision event between the drone and the wires, showing how the contact is recorded in the 3D grid map (green box). Fig. 6.11c illustrates the collision recovery stage, where the drone moves backward by the waypoints defined by the *step back* parameter (three waypoints in this case) until it reaches a non-contact state. Finally, Fig. 6.11d shows the recalculated RRT* path that avoids the newly marked contact box in the grid map, as well as the drone successfully avoiding the previously collided obstacle and proceeding towards the goal waypoint.

This experiment demonstrates how the path-planning process integrates LiDAR and contact information to enhance obstacle detection and avoidance. The LiDAR sensor provides a 3D map of the environment it can detect, while the contact detection system refines this map by adding obstacles not detected by the LiDAR. This combination results in a more comprehensive environmental model.

Metrics	Results
Mission Completed	Yes
Number of collisions	1
Contact event duration (s)	1.053
Reaction time (s)	0.538
Angle predicted	3.74
Angle ground truth	0.23
Angle error	3.54

Table 6.2: Lidar and contact-based navigation real experiment results

Regarding the mission metrics, the lidar and contact-based navigation mission have been completed with only 1 collision. The duration of the contact event was 1.053 seconds,



(a) Initial path with octomap

(b) Collision and contact map updated

(c) Collision recovering

(d) Recalculated path

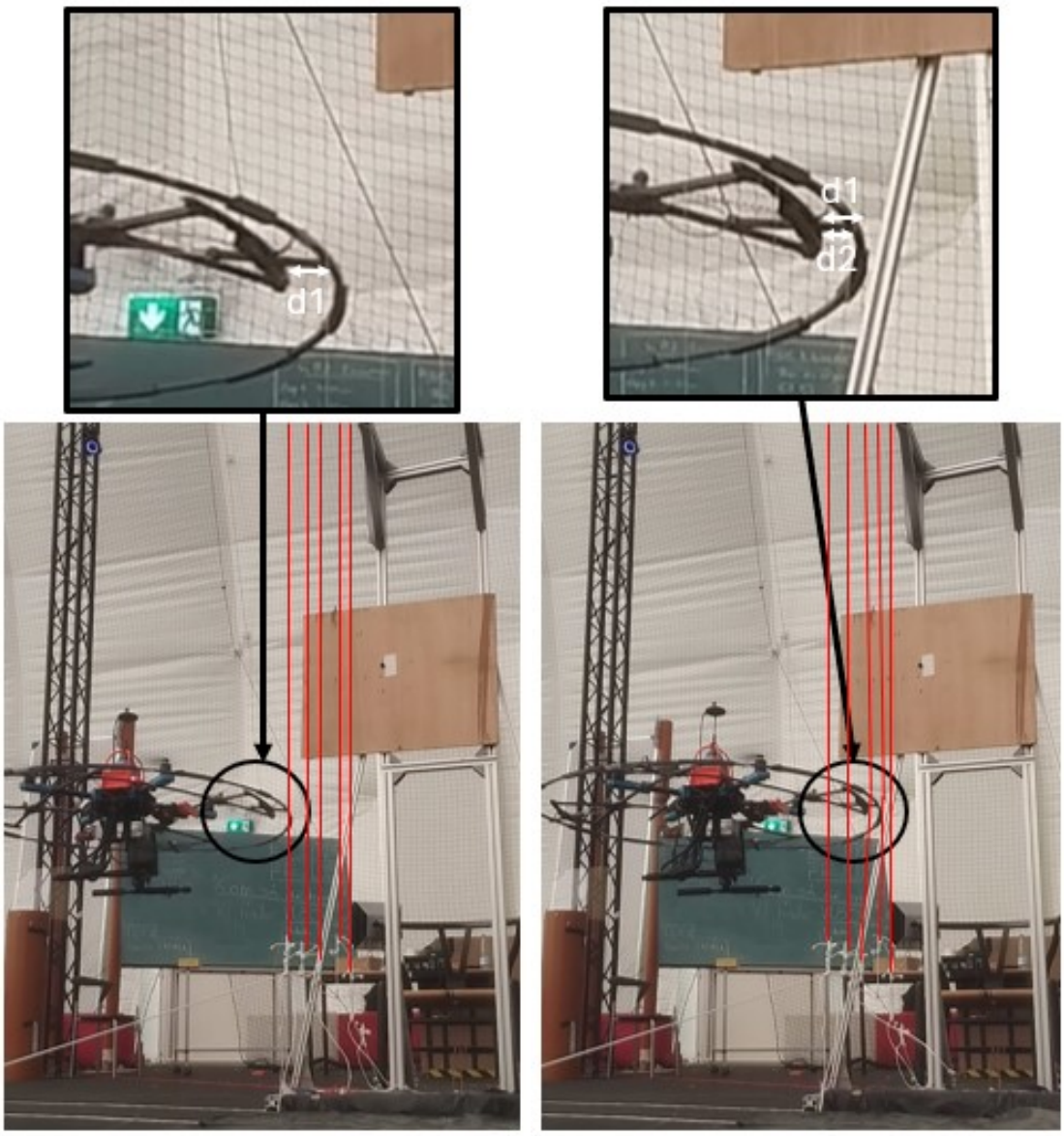
Figure 6.11: Lidar and Contact-based Navigation experiment showing planning and mapping stages

which is significantly longer than in the previous experiment. This extended duration is likely due to the flexible wires, resulting in a smoother and less abrupt collision as the drone pushes against them. This highlights the platform's ability to detect rigid and flexible collisions effectively.

The reaction time was 0.538 seconds, slower than the previous experiment. This slower reaction may be attributed to integrating LiDAR data into the planning algorithm, which can introduce additional computational delays.

The prediction angle error was 3.54 degrees, critical in accurately mapping the collision. This low error margin ensured that the collision was precisely included in the map, allowing the drone to replan its path effectively and avoid the obstacle with just one collision.

Fig. 6.12 shows the collision interaction between the platform and the thin wires (highlighted in red) not detected by the lidar sensor. As seen before the collision, the difference between the collision platform's outer and inner ring is d1. In contrast, during the collision, that distance is decreased to d2 because of the outer ring movement.



(a) UAV before collision (b) UAV colliding

Figure 6.12: Lidar and Contact-based Navigation experiment collision platform interaction with thin flexible wires

7 Discussion

In this thesis, the primary focus was on developing a collision-tolerant system to enable UAV autonomous navigation in obstacle-dense environments. The system consists of the mechanical design of the collision platform and the algorithm to predict the collision information.

The exploration began with assessing the most common autonomous collisions as shown in the studies [11] and [10], where thin obstacles are undetected, and localization drift appears, respectively. Based on this issue, existing collision platforms and contact detection approaches were explored in [28] and [29]. This research led to the development of a novel collision platform and a contact detection algorithm to detect collision events, orientation, and displacement to leverage contact information to recover from collisions.

Further, the research delved into integrating the previously designed collision system into a real autonomous mission, showing the collision system performance within a real scenario, also done in [28] and [31].

The testing phases, conducted in a controlled environment, facilitated a thorough validation of the proposed system. First, tests related to the collision platform's durability showed the platform did not break. However, as for maintenance, an elastic band replacement was needed after one month to keep the platform stable.

Then, some interesting insights from the contact detection models can be analyzed. Regarding contact detection, the results shown in Fig. 6.1 confirm that contacts are always correctly predicted in a static environment, as also achieved in [29]. However, the evaluation of contact detection shown in Fig. 6.2 demonstrates the ability of the collision system to work under flight conditions as well.

Regarding the collision angle predictions, successful results have been achieved. In contrast with the proposed method in this study [29], where contact orientation predictions are discretized into eight bins with a 45-degree resolution, the solution proposed can predict a continuous collision orientation, ranging from 0 to 360. The best results reported an MAE of 9.31º and a standard deviation of 10.84 º by leveraging collision time-series information (Model 12 in Fig. 6.4), meaning around a 35-degree resolution improvement. This increase in resolution is crucial when increasing the robot's dimensions, making angular errors more impactful as the size of the platform grows.

Additionally, collision displacement information has been added to the contact detection output. No other study has reported this inclusion in a 360º continuous platform. The most similar approaches are in [31], which outputs the force and torque from the collision, and in [28], which outputs the angle bent by a collision antenna as an impact magnitude. Regarding displacement predictions, results are not as accurate as the angles, but results show potential improvement. The best displacement prediction model (Model 3 from Fig. 6.4) reports an MAE of 0.58 cm and a standard deviation of 0.79 cm. However, when looking into the displacement during the contact events in Fig. 6.8, it can be seen how the displacement pattern is well predicted (increasing when colliding and decreasing when recovering). Still, some bias can be present, as shown in Fig. 6.8b, which causes the large error metrics. This bias could be caused by the labels used to train the displacement, which relies on the contact prediction, and it also does not consider the UAV pitch angle when recording displacement, assuming that a horizontal collision is produced.

The contact detection model results described above demonstrate that using neural networks fed by flex sensor information can provide accurate information about collision events.

Finally, the autonomous missions performed, integrating the collision information, demonstrate successful contact-based navigation with the simplest mapping and planning setup. The experiments showed the platform's ability to navigate without perception sensors and compensate for the lidar's lack of resolution.

Overall, the findings from this research provide compelling evidence that the innovative collision platform and contact detection algorithm allow for accurate prediction of collision events, orientations, and displacements, thereby improving the UAV's ability to recover from collisions and maintain its mission. The successful integration of this system into autonomous missions demonstrates its practical applicability and potential to operate effectively without relying on traditional perception sensors. These contributions address the key research goal and lay the groundwork for future studies to further optimize the system and explore application-specific enhancements.

8 Conclusion

This thesis has effectively demonstrated and validated the development of an autonomous navigation system tailored for obstacle-dense environments, confirming its practical applicability in real-world scenarios. By integrating a novel collision detection platform with a contact-based sensing system, the research has significantly enhanced autonomous aerial vehicles, enabling them to detect and recover from collisions in challenging conditions.

Extensive testing in static and flight scenarios corroborated the proposed solution and underscored the system's collision detection accuracy. The collision detection platform successfully performed contact-based autonomous missions, showcasing its potential to excel in real use cases. These accomplishments mark a substantial advancement in the field of contact-based navigation, showing promising potential for broader applications.

Future research could focus on optimizing the collision platform by enhancing its robustness and reducing its weight through improved designs and materials, which is crucial given the limited battery capacity of aerial vehicles. Additionally, the volume covered by the platform could be expanded from 2D to 3D protection, providing more comprehensive coverage and further improving the system's ability to detect and respond to collisions from all directions.

Furthermore, integrating additional sensors, such as IMUs, into the collision detection system by employing data fusion techniques could leverage the strengths of multiple information sources, thereby enhancing the overall system performance.

Another area of exploration could be force analysis based on platform displacements. By adjusting elastic joint strengths and predicting platform displacement, it is possible to determine the current force applied to the platform, which could be valuable for contact-based missions.

These future research directions will extend the current system's capabilities and contribute significantly to the broader field of autonomous vehicle technology, addressing key challenges and expanding the operational domain.

Bibliography

[1] Christos Papachristos et al. “Distributed infrastructure inspection path planning for aerial robotics subject to time constraints”. In: *2016 international conference on unmanned aircraft systems (ICUAS)*. IEEE. 2016, pp. 406–412.

[2] Sezal Jain et al. “Autonomous river exploration”. In: *Field and Service Robotics: Results of the 9th International Conference*. Springer. 2015, pp. 93–106.

[3] Jean-Christophe Zufferey, Antoine Beyeler, and Dario Floreano. “Optic flow to steer and avoid collisions in 3D”. In: *Flying Insects and Robots* (2010), pp. 73–86.

[4] James F Roberts et al. “Quadrotor using minimal sensing for autonomous indoor flight”. In: *European Micro Air Vehicle Conference and Flight Competition (EMAV2007)*. 2007.

[5] Christos Papachristos, Shehryar Khattak, and Kostas Alexis. “Autonomous exploration of visually-degraded environments using aerial robots”. In: *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2017, pp. 775–780.

[6] *PX4 Offboard Control*. https://docs.px4.io/v1.12/en/flight_stack/controller_diagrams.html.

[7] Keisuke Fujii. “Extended kalman filter”. In: *Refernce Manual* 14 (2013), p. 41.

[8] Frank Dellaert et al. “Monte carlo localization for mobile robots”. In: *Proceedings 1999 IEEE international conference on robotics and automation (Cat. No. 99CH36288C)*. Vol. 2. IEEE. 1999, pp. 1322–1328.

[9] Greg Welch, Gary Bishop, et al. “An introduction to the Kalman filter”. In: (1995).

[10] Fang Liao et al. “3D motion planning for UAVs in GPS-denied unknown forest environment”. In: *2016 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2016, pp. 246–251.

[11] V Karjalainen et al. “A DRONE SYSTEM FOR AUTONOMOUS MAPPING FLIGHTS INSIDE A FOREST–A FEASIBILITY STUDY AND FIRST RESULTS”. In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 48 (2023), pp. 597–603.

[12] Armin Hornung et al. “OctoMap: An efficient probabilistic 3D mapping framework based on octrees”. In: *Autonomous robots* 34 (2013), pp. 189–206.

[13] Xin Zhou et al. “Swarm of micro flying robots in the wild”. In: *Science Robotics* 7.66 (2022), eabm5954.

[14] Tong Qin, Peiliang Li, and Shaojie Shen. “VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator”. In: *IEEE Transactions on Robotics* 34.4 (2018), pp. 1004–1020.

[15] Steven M LaValle and James J Kuffner. “Rapidly-exploring random trees: Progress and prospects: Steven m. lavalle, iowa state university, a james j. kuffner, jr., university of tokyo, tokyo, japan”. In: *Algorithmic and computational robotics* (2001), pp. 303–307.

[16] Devin Connell and Hung Manh La. “Dynamic path planning and replanning for mobile robots using RRT”. In: *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE. 2017, pp. 1429–1434.

[17] N SV Rao et al. *Robot navigation in unknown terrains: Introductory survey of non-heuristic algorithms*. Tech. rep. Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States), 1993.

[18] Tomàs Palleja et al. “Modeling floor-cleaning coverage performances of some domestic mobile robots in a reduced scenario”. In: *Robotics and Autonomous Systems* 58.1 (2010), pp. 37–45.
[19] Markus Rude. “A flexible, shock-absorbing bumper system with touch-sensing capability for autonomous vehicles”. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS’96*. Vol. 2. IEEE. 1996, pp. 410–417.
[20] Holly A Yanco. “Wheelesley: A robotic wheelchair system: Indoor navigation and user interface”. In: *Assistive technology and artificial intelligence: applications in robotics, user interfaces and natural language processing* (2006), pp. 256–268.
[21] Adrien Briod et al. “The AirBurr: A flying robot that can exploit collisions”. In: *2012 ICME International Conference on Complex Medical Engineering (CME)*. IEEE. 2012, pp. 569–574.
[22] Adrien Briod. “Observation of insect collisions”. In: (2008).
[23] Switzerland Lausanne. “Design of Flying Robots for Collision Absorption and Self-Recovery.[Google Scholar]”. In: (2012).
[24] Pedro Silva Girão et al. “Tactile sensors for robotic applications”. In: *Measurement* 46.3 (2013), pp. 1257–1271.
[25] Miriam Fend, Hiroshi Yokoi, and Rolf Pfeifer. “Optimal morphology of a biologically-inspired whisker array on an obstacle-avoiding robot”. In: *Advances in Artificial Life: 7th European Conference, ECAL 2003, Dortmund, Germany, September 14-17, 2003. Proceedings 7*. Springer. 2003, pp. 771–780.
[26] Francesco Mondada et al. “The e-puck, a robot designed for education in engineering”. In: *Proceedings of the 9th conference on autonomous robot systems and competitions*. Vol. 1. 1. IPCB: Instituto Politécnico de Castelo Branco. 2009, pp. 59–65.
[27] *Spectra Symbol, ”Flex Sensor”*. http://www.spectrasymbol.com/.
[28] Christos Papachristos, Shehryar Khattak, and Kostas Alexis. “Haptic feedback-based reactive navigation for aerial robots subject to localization failure”. In: *2019 IEEE Aerospace Conference*. IEEE. 2019, pp. 1–7.
[29] Roberto Ponticelli and P Gonzalez de Santos. “Full perimeter obstacle contact sensor based on flex sensors”. In: *Sensors and Actuators A: Physical* 147.2 (2008), pp. 441–448.
[30] Adrien Briod et al. “Contact-based navigation for an autonomous flying robot”. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2013, pp. 3987–3992.
[31] Emanuele Aucone et al. “Synergistic morphology and feedback control for traversal of unknown compliant obstacles with aerial robots”. In: *Nature Communications* 15.1 (2024), p. 2646.
[32] Sorin Grigorescu et al. “A survey of deep learning techniques for autonomous driving”. In: *Journal of field robotics* 37.3 (2020), pp. 362–386.
[33] Mihalj Bakator and Dragica Radosav. “Deep learning and medical diagnosis: A review of literature”. In: *Multimodal Technologies and Interaction* 2.3 (2018), p. 47.
[34] Shashi Pal Singh et al. “Machine translation using deep learning: An overview”. In: *2017 international conference on computer, communications and electronics (comptelix)*. IEEE. 2017, pp. 162–167.
[35] Marius-Constantin Popescu et al. “Multilayer perceptron and neural networks”. In: *WSEAS Transactions on Circuits and Systems* 8.7 (2009), pp. 579–588.
[36] Aleksandra Vuckovic et al. *EEG Drowsiness 2002 MedEngPhy*. Mar. 2015.

[37] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536.
[38] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.
[39] *Medium, ”An Intuitive Explanation of LSTM”*. https://medium.com/@ottaviocalzone/an-intuitive-explanation-of-lstm-a035eb6ab42c.
[40] Jig Han Jeong et al. “Random forests for global and regional crop yield predictions”. In: *PloS one* 11.6 (2016), e0156571.
[41] Jing Cai et al. “Real-Time Arrhythmia Classification Algorithm Using Time-Domain ECG Feature Based on FFNN and CNN”. In: *Mathematical problems in engineering* 2021.1 (2021), p. 6648432.
[42] Luqiang Shi et al. “Tilt angle monitoring by using sparse residual LSTM network and grid search”. In: *IEEE Sensors Journal* 19.19 (2019), pp. 8803–8812.
[43] Heli Koskimäki et al. “Activity recognition using a wrist-worn inertial measurement unit: A case study for industrial assembly lines”. In: 2009 (June 2009). DOI: 10.1109/MED.2009.5164574.
[44] *Holybro, ”Holybro X500 v2 PX4 Development Kit”*. https://holybro.com/products/px4-development-kit-x500-v2.
[45] *ROS2, ”ROS2 Basic Concepts”*. https://docs.ros.org/en/iron/Concepts/Basic.html.

Technical
University of
Denmark

Elektrovej, Building 326
2800 Kgs. Lyngby
Tlf. 4525 1700

www.electro.dtu.dk