ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Autonomous Systems Lab
Prof. Roland Siegwart

Master Thesis

Exploring Aerial Manipulation with an Omnidirectional Hexacopter

Spring Term 2025

Supervised by:
Mike Allenspach
Thomas Stastny
Eugenio Cuniato

Author:
Elias Steiner

Declaration of Originality

I hereby declare that the written work I have submitted entitled

Exploring Aerial Manipulation with an Omnidirectional Hexacopter

is original work which I alone have authored and which is written in my own words.[1]

Author(s)

Elias	Steiner

Student supervisor(s)

Mike	Allenspach
Thomas	Stastny
Eugenio	Cuniato

Supervising lecturer

Roland	Siegwart

With the signature I declare that I have been informed regarding normal academic citation rules and that I have read and understood the information on 'Citation etiquette' (`https://www.ethz.ch/content/dam/ethz/main/education/rechtliches-abschluesse/leistungskontrollen/plagiarism-citationetiquette.pdf`). The citation conventions usual to the discipline in question here have been respected.

The above written work may be tested electronically for plagiarism.

25.05.2025, Zürich
Place and date


Signature

[1] Co-authored work: The signatures of all authors are required. Each signature attests to the originality of the entire piece of written work in its final form.

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every written paper or thesis authored during the course of studies. In consultation with the supervisor, one of the following three options must be selected:

- (●) I confirm that I authored the work in question independently and in my own words, i.e. that no one helped me to author it. Suggestions from the supervisor regarding language and content are excepted. I used no generative artificial intelligence technologies[1].
- ( ) I confirm that I authored the work in question independently and in my own words, i.e. that no one helped me to author it. Suggestions from the supervisor regarding language and content are excepted. I used and cited generative artificial intelligence technologies[2].
- ( ) I confirm that I authored the work in question independently and in my own words, i.e. that no one helped me to author it. Suggestions from the supervisor regarding language and content are excepted. I used generative artificial intelligence technologies[3]. In consultation with the supervisor, I did not cite them.

Title of paper or thesis:

Exploring Aerial Manipulation with an Omnidirectional Hexacopter

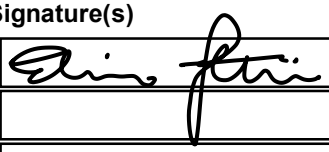
Authored by:
If the work was compiled in a group, the names of all authors are required.

Last name(s):	**First name(s):**
Steiner	Elias

With my signature I confirm the following:

- I have adhered to the rules set out in the Citation Guide.
- I have documented all methods, data and processes truthfully and fully.
- I have mentioned all persons who were significant facilitators of the work.

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

Place, date	**Signature(s)**
25.05.2025	

If the work was compiled in a group, the names of all authors are required. Through their signatures they vouch jointly for the entire content of the written work.

[1] E.g. ChatGPT, DALL E 2, Google Bard
[2] E.g. ChatGPT, DALL E 2, Google Bard
[3] E.g. ChatGPT, DALL E 2, Google Bard

Contents

Abstract

This thesis explores the feasibility and precision of aerial manipulation using a fully actuated omnidirectional hexacopter platform developed at the Autonomous Systems Lab at ETH Zürich. A reactive peg insertion task serves as the central benchmark to evaluate the system's performance since it includes multiple different aspects that were found to be particularly challenging by related research groups: target identification without a priori knowledge, high end effector precision, and transitioning from free flight to contact with a rigid object.

The final system integrates a vision-based object tracker with robust outlier rejection, a composite controller with policies for individual motion axes, and passively compliant end effectors to mitigate contact forces. During a test phase, the full system was able to successfully perform the peg insertion task. A hole with a diameter of 0.8 cm could reproducibly be hit with the end effector after a slow and supervised approach, and prolonged contact of more than 10 seconds is possible. However, the actual platform precision is estimated to be approximately 2-3 cm, as an area of this size could be struck with full certainty in all circumstances.

A key emphasis is placed on modularity: multiple variants of the visual detector and tracking modules were implemented and tested, and different compliant end effector attachments were evaluated. Both learned and classical detection algorithms achieved true positive rates of more than 70% in good visibility conditions and at least 20% in poor conditions. Subsequently, a target estimation accuracy as low as 1.1 cm for close-range scenarios is possible and exhibits proportional degradation as the distance from the target is increased.

These results highlight that centimeter-level manipulation tasks with prolonged contact to rigid surfaces are achievable using vision-only object tracking and fully actuated UAV control, without external motion capture.

Symbols

Acronyms and Abbreviations

DoF	Degrees of Freedom
UAV	Unmanned Aerial Vehicle
ToF	Time of Flight (Camera)
FMD	Fused Deposition Modelling (3D Printing)
RGB	Red Green Blue (Camera / Image)
RAFT	Recurrent All-Pairs Field Transforms for Optical Flow [1]
ROS	Robot Operating System [2]
NMS	Non-Maximum Suppression
RMP	Riemannian Motion Policies [3, 4]
NaN	Not a Number (value)
PDF	Probability Density Function

1 Introduction

In recent years, unmanned aerial vehicles (UAVs) have seen widespread and fast adoption in both private and industrial settings for a wide range of tasks. Recreational usage such as videography or drone racing is largely responsible for private usage, where primarily fixed-tilt quadcopters are used due to their simplicity. Even in the case of industrial applications - with tasks involving construction, safety, agriculture or logistics dominating - fixed-tilt drones are heavily used. For such environments, a larger spread in the size of the platform or the number of rotors can be observed due to technical and strength requirements [5].
While fixed-tilt rotor platforms are highly effective, they have fundamental limitations in terms of control authority: during flight, not all degrees of freedom (DoF) can be controlled independently due to a partial coupling between the linear and rotational motion. These constraints are only of minor importance for tasks that are mainly concerned with dynamically controlling the drone and involve a lot of large-scale movement. However, as the limit of possible applications is pushed further, and especially for tasks involving precise aerial manipulation, this control constraint becomes problematic. Specifically, cases where a drone must exert forces or follow precisely constrained paths in contact with an external surface become difficult to implement on classical fixed-rotor systems.

In order to overcome these limitations and enable increasingly precise aerial manipulation, the Autonomous System Lab (ASL) at ETH Zürich has developed a fully actuated hexacopter platform. Each of the six propeller arms on this drone is independently rotatable, thus allowing for full six-degree-of-freedom motion and position control in hover. Decoupling rotation and translation allows this platform to either freely rotate about a stationary point in space or, alternatively, to exert controlled forces at its end effector while maintaining a constant attitude. Since fixed-rotor drones and - even more so - fully actuated drones are still a young branch of robotics, the research conducted on aerial manipulation to date is limited. The overarching aim of this thesis is therefore to explore the capabilities and limitations of aerial manipulation with the aforementioned platform by means of a representative prototype task, discussed in later sections.

1.1 Related Work

Aerial manipulation is a broad topic and includes a wide variety of tasks, hardware, and a combination thereof. For a single thesis, it will by no means be possible to cover a whole range of tasks, let alone a generalized approach to aerial manipulation. Hence, one suitable “prototype” task first had to be identified to have a

tangible project to work with. Such a task should remain representative of general aerial manipulation, and thus ideally include some of the key challenges and technical gaps that are commonly faced by existing research projects. For this, previous works of other research groups in the field of aerial manipulation were consulted. Both publications involving fixed-rotor and fully actuated tilt-rotor platforms were considered since the tasks to which they are applied are generally similar, and both can be helpful in understanding the current state-of-the-art.

A significant part of aerial manipulation research is conducted with the help of classical drones featuring fixed non-tilt rotors. However, due to their underactuated nature, simpler tasks are usually performed, such as the application of force [6, 7], tracking of trajectories [8, 9], or contacting and maintaining the connection to an object [10, 11, 12].
By fixedly tilting the rotor axes, a partial decoupling of rotational and linear motion can be achieved. Such platforms offer an initial solution to the underactuation problem and allow for successful execution of tasks like peg insertion [13] or surface tracking with force application [14]. However, fixedly tilted rotor platforms are limited in their actuation range, and some orientations are extremely inefficient or outright impossible to assume.
For these reasons, a common approach in aerial manipulation is to enhance an underactuated platform by mounting an actuated end-effector on the drone to increase the controllable degrees of freedom of the full system. A wide variety of manipulation tasks are made possible by using such platforms: aerial screwing / drilling [15, 16], peg insertion [17, 18], aerial writing [19], sensor installation and inspection tasks [18, 20, 21], and a variety of grasping tasks [22, 23, 24].
While this approach enables the execution of much more intricate tasks, it brings a challenging increase in weight from the actuated end-effectors and adds more complexity to control schemes. For this reason, in recent years, actuated tilt-rotor platforms have seen an increase in application. They offer full six-degree-of-freedom motion without the need for complex end-effector actuation. Some possible tasks are aerial drilling / screwing [25], application of strong force while remaining at a fixed location in space [26] or push-and-slide inspection tasks [27].

Analyzing the existing research conducted across all types of platforms reveals some recurrent and commonly faced challenges. The exact moment of contact making with a fixed surface, object, or between multiple flying platforms ("mode switching") is challenging. Both controls and hardware must be robust enough to handle a sudden change in drone dynamics and forces that are being applied. Closely related to "mode switching" is the challenge of remaining stable while in contact with a fixed surface during manipulation. Aerial vehicles inherently have no way to dissipate contact forces other than by compensating them with rotor actions [6, 7, 10, 11, 12, 16, 21, 22, 25]. Another limiting factor is the maximum force or torque that the platform can produce. This directly limits any forces that are applicable for a manipulation task or the amount of payload that can be carried - which is especially relevant in the case of mounting additional actuated manipulators [8, 9, 10, 11, 16, 22, 23].

Interestingly, a large share of research groups use a known target location in combination with absolute navigation in a motion capture environment. This is understandable, as it allows more focus to be laid on aspects other than target localization. However, this approach limits any real-world applicability, where the state information that can be provided by a motion capture system is usually not available. Only a few groups have explored working without a priori target information. A time of flight (ToF) camera can be used to perform depth servoing [27] or lines extracted from an image can be used as trajectory guidance [14]. However, in terms of precise single-point target tasks, only partial solutions have been explored: visual localization is either used only for ultra-close range alignment [25], only in simulation and close range [17], or it uses complicated handcrafted feature extraction limited to one specific object [23].

Furthermore, concrete numbers regarding the achievable precision with complete aerial manipulation systems are rarely mentioned. One result that could be found is a reported $\leq 1cm$ peg insertion task that could be successfully completed repeatedly [13]. Another group has evaluated the deviation in the lines of text written by a drone with an actuated manipulator. They report that the error range of the end effector tip is within ±1cm and the positional error of the drone body to be in the range of ±5cm [19]. Finally, the AEROARMS project has measured mean localization errors of around 10cm during outdoor inspection tasks [18].

1.2 Goals

In search of a fitting aerial manipulation task for this thesis, reviewing the literature has revealed that controlled force application and surface-following tasks are more thoroughly explored than any other task and would therefore likely yield less relevant findings. On the other hand, an interesting aspect to explore seems to be the general precision of a manipulation platform, as it is rarely examined. Secondly, the "mode switching" behavior of a drone when contacting a rigid object or the challenge of holding prolonged contact should certainly also be a part of our task, since these points are frequently mentioned as being difficult to overcome. Thirdly, refraining from using a known target location and a motion capture system to have knowledge of the absolute position of the drone will make any development more applicable to real-world scenarios where such state measurements are generally not available. While many groups make use of actuated manipulators mounted on flying platforms, this should not be required for this project. A fully actuated tilt-rotor platform is available and offers six controllable degrees of freedom. Adding a manipulator would therefore only drive up complexity as any more actuated degrees of freedom are not necessarily required.
Considering those findings, a peg insertion or screwing task seem to be fitting candidates, since both require sufficient point-accuracy and include contact making – in the case of screwing even prolonged contact holding. Additionally, by using a reactive object tracker instead of a motion capture environment and a priori target knowledge, real-world applicability is enhanced.

The goal of this thesis will therefore be to develop and implement a system that is successfully able to perform a reactive peg-insertion task with the existing fully actuated omnidirectional hexacopter platform at ASL. Performing this task enables us to further explore some of the underlying aspects of aerial manipulation such as: the usefulness of certain types of sensors, applicable control schemes, any additional hardware needed for the end effector, and finally to gauge what the overall achievable precision of our system is.

2 Problem Statement

2.1 Baseline

The drone platform used for this project and, therefore, to complete the peg insertion task is an omnidirectional hexacopter developed by ASL at ETH Zürich (Figure 2.1). It features six identical propellers, each mounted on an arm with full rotational capacity, actuated by a separate servo motor. By being able to tilt each rotor independently, this drone has six controllable degrees of freedom. The platform can theoretically assume any rotation while remaining at a fixed position in space and, vice versa, exert a force or move in any linear direction while maintaining any fixed rotation.
A generic attachment point for end effectors is available at the tip of one of the drone arms. By disabling the propeller of said arm, the rotational capabilities of the servo motor can be used to safely actuate any attached end effector. However, this capability was not used for any experiments during this project. Furthermore, the drone features an eye-in-hand-style RGB camera that is mounted on the same arm as the end effector. A Time of Flight (ToF) camera is also available and is mounted rigidly on the drone body.

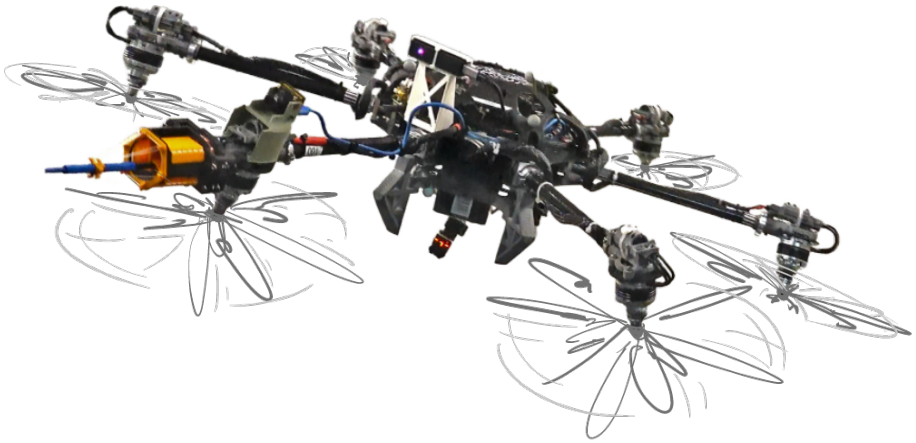


FIGURE 2.1: *The omnidirectional hexacopter platform, developed by ASL. An end effector, the eye-in-hand RGB camera, as well as the ToF camera are visible.*

For the peg insertion task, a 3D printed block of around 3 cm x 3 cm x 1 cm in size with a 0.8 cm hole serves as the target. This target is mounted on a whiteboard wall to simulate a hole in a vertical wall (Figure 2.2). Conveniently, this setup can be reconfigured to feature multiple targets or different layouts, depending on

the experimental requirements. In addition, a Vicon motion capture system was available during all experiments. However, usage was limited to the evaluation of system components (see Section 4.2) and sensor fusion for the drone's odometry estimation. A discussion regarding the impact of the motion capture system used for odometry estimation can be found in Section 6.

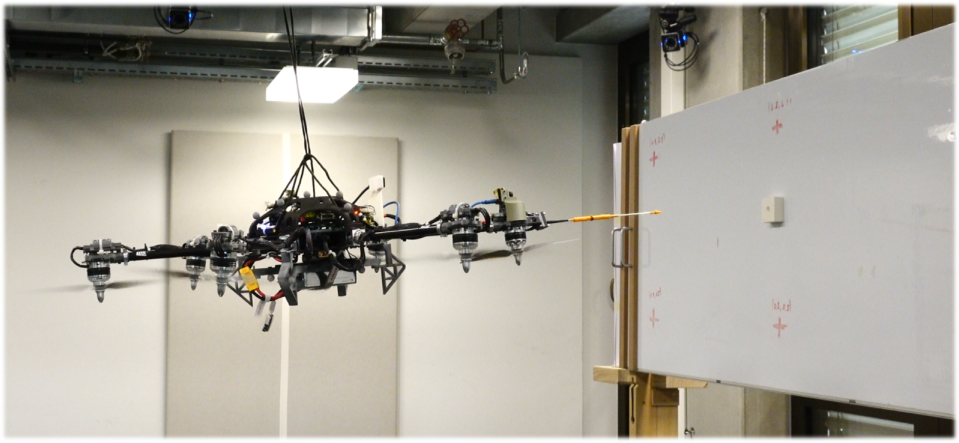


FIGURE 2.2: *A typical experimental setup with the drone in front of the white target block mounted on a whiteboard wall with sticky tape.*

2.2 Design Constraints

Besides aiming to complete the previously stated goals of achieving the peg insertion task and exploring aerial manipulation, certain design constraints were set at the beginning of this project.
The first constraint is best summarized as follows: pursuing modularity in all components that are going to be implemented. By making all subsystems as encapsulated as possible and ensuring that they work in a standalone fashion, multiple variants of components can be tested. Additionally, modularity allows the system to be more easily adapted to perform other tasks in the future, thus making developed components and gained knowledge more generally applicable.
The second constraint is to minimize the number and type of sensors that the system uses whenever possible. While providing more information, each added sensor also increases complexity and can introduce possible failure points simultaneously. Therefore, it can be beneficial to limit sensor usage if compromise is acceptable.

2.3 System Overview

Naturally, realizing the peg insertion task will involve multiple components working together – each taking care of an individual subtask. Understanding the function, boundaries, and inputs or outputs of those components is essential before starting any technical implementation and can help to adhere to the constraint of modularity. The following subsystems were determined to be necessary for the peg insertion task.

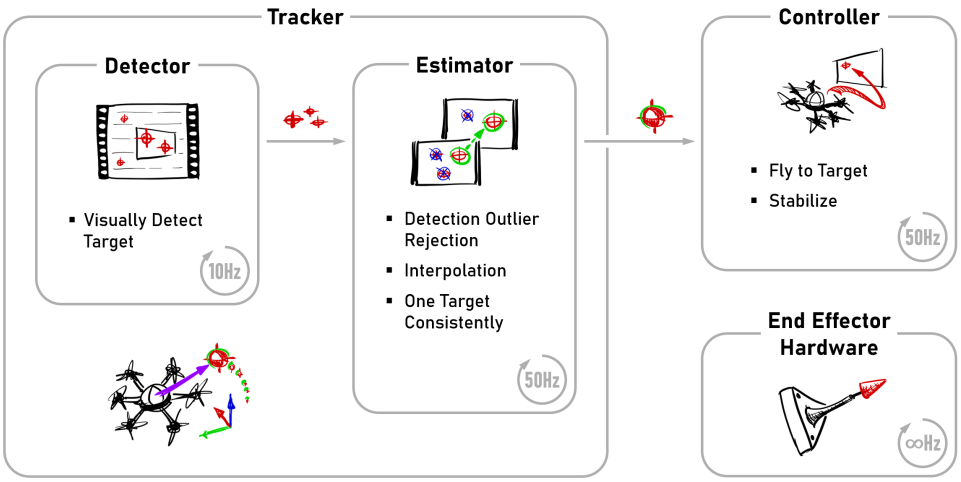


FIGURE 2.3: *A schematic system overview showing all modules, their connection and an expected execution frequency for each. The data passed from the Detector to the Estimator are raw detection points, whereas the Tracker outputs a single, consistent target estimate.*

Tracker Given the lack of a priori knowledge about the target's location, an RGB camera mounted in an eye-in-hand configuration will be used to visually locate the target and establish a baseline. The purpose of the Tracker module is to ultimately extract a single consistent estimate of the target location from only the camera input. The Tracker itself consists of a Detector and an Estimator module.

Detector As the first part of the Tracker, the role of the Detector module is to identify the target in an image and output detection points. Any detection algorithm that might be used for this task is subject to noise and likely contains outliers.

Estimator The Estimator Module constitutes the second and larger part of the Tracker. The main goal is to clean the raw detections by removing outliers, staying on one target consistently in the case of multiple targets being present in the image, and finally, interpolating between detections for a higher frequency estimate.
At the core of the Estimator is some knowledge of the system dynamics to predict where the next measurement of the target is expected to appear. A first logical choice would be to use optical flow for this purpose, as it can be directly determined from the same image input without the need for additional sensors. However, optical flow was examined and discarded in an early decision, since even learned estimations such as RAFT [1] proved to be computationally expensive (around 5-10 Hz are possible depending on the desired output quality). As replacement, the drone's odometry estimate is used (available at 200 Hz), allowing us to err on the side of caution and build a robust estimator.

Controller Once a consistent estimate is established, it can be given to a controller with the objective of steering the drone to the target location and holding a stable position at the target. The Controller module will have to fit within the existing control architecture currently used on the hexacopter platform.

End Effector Hardware Finally, a suitable attachment for the generic mounting point, available on one of the drone arms, must be designed. Such an end effector will serve as an extension of the arm, allowing for safe operation near a fixed target. Additionally, the attachment should feature some form of compliance to overcome the challenge of sudden mode change upon contact with an object.

3 Methodology

3.1 Tracker Module

The Tracker module is the first part of the complete system for the peg insertion task. Its role is to extract a single, consistent estimated location of the target from an RGB camera input. A visual detection algorithm first generates raw 2D detection points from the image (see Section 3.1.1). These are subject to slight positional noise, but more importantly, outliers are inevitably present. For this reason, an Estimator module subsequently cleans up the raw detections (see Section 3.1.3).

A common solution to the cleanup or "filtering" of such raw data is to combine measurements with knowledge of the system dynamics. This allows for predictions of future detections to be made from past measurements. The location where a possible next detection is likely to occur can be estimated, and thus outliers can be rejected, and discerning between multiple possible targets over time becomes possible. Additionally, knowledge of the system dynamics permits sensible interpolation between detections in a temporal sense for higher frequency estimates.
The drone's odometry estimate will serve as knowledge of the system dynamics for the Estimator module. However, this requires that any predictions be made in the form of 3D points. Consequently, the target estimation will also be three-dimensional. For this reason, depth information must be added to the 2D image detection points. A time-of-flight (ToF) camera provides accurate distance measurements, which are then fused with the two-dimensional detection points, augmenting them to full 3D detections (process discussed in Section 3.1.2).

All components of the Tracker module were implemented as a ROS package within the existing environment of the platform software stack developed by ASL. The nodes for this project were written exclusively in the Python programming language due to the vast amount of existing useful libraries and the ease of implementation. The entire source code of the Tracker module can be found in the GitHub repository for this thesis [28].

3.1.1 Visual Detector

For the Detector module, two distinct existing solutions were investigated and implemented side by side. A classic OpenCV blob tracker and a - more intricate - Ultralytics YOLOv8 learned image detection model [29]. Both work independently and can be used interchangeably for experimental comparison.

Blob Tracker

As a tried and proven image detection algorithm, the OpenCV blob tracker mainly served to establish a baseline for the Detector module. By detecting contiguous shapes of the same color in images, a blob tracker can extract hundreds of keypoints from a single image. Then, these are filtered by a set of parameters, such as size, circularity, or convexity. Tuning the parameters enables the blob tracker to respond to very specific objects only. However, due to its simplicity, this algorithm is limited to detecting flat and plain objects of uniform color only. Fortunately, the targets for the peg insertion task fulfill these requirements since they are dark circles of uniform size. The blob tracker was experimentally tuned until as many outliers as possible could be rejected without losing true positive detections. For exact details on parameter tuning, refer to Section A.1. All code related to the blob detector can be found in the file `scripts/node_detector_blob.py`.

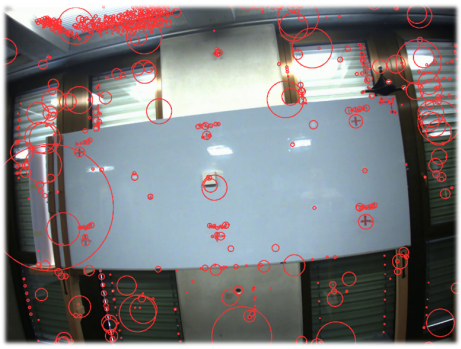


FIGURE 3.1: *Unfiltered blob detections.*

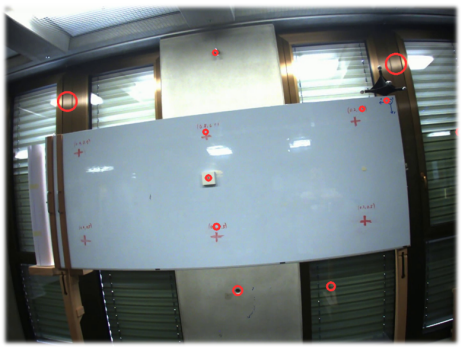


FIGURE 3.2: *Filtered blob detections.*

Ultralytics YOLOv8

The YOLOv8 model by Ultralytics [29] is a pretrained network, set up for image detection tasks. By fine-tuning the provided model with annotated images, YOLO can be customized to detect an immense range of individual objects and classify hundreds of classes with one network [30]. Furthermore, five differently sized variants of YOLOv8 are available: nano (n), small (s), medium (m), large (l), and extra large (x). The size primarily influences the number of channels in the convolution blocks of the network and therefore directly controls the total capacity and computational cost of each model [31]. Given that for this project only one type of object is detected without any intra-class variability, the smallest model (nano) was used. Ultralytics offers a convenient API interface (the Ultralytics Python module) for finetuning their pre-trained models. To do so, only the training images must be provided and set up in the correct folder structure of the project. In addition, all training parameters can be conveniently set in an auxiliary YAML file. The execution of the training and validation procedure itself, as well as all resource management, is automatically handled by the Ultralytics Python module.

Fine-Tuning Process The generation of custom annotated images for this project was performed by first capturing a short video sequence of the target setup. The

footage was captured directly using the onboard RGB camera of the drone. Importantly, different angles, lighting, and distances from the target are included to provide some variation. Static frames were then periodically extracted from the video sequence. With the help of an online tool [32], around 300 individual images were manually annotated with a bounding box, covering the target hole as precisely as possible.
During fine-tuning with these annotated images, the automatic online image augmentation provided by Ultralytics was heavily used. 300 images is a comparatively small sample size for vision models – even for fine-tuning. Not including augmentation led to overfitting during initial training runs [33]. Instead of manually covering more varied lighting conditions, viewing angles, or other variables, image augmentation can provide a useful "simulation" of these aspects without any additional labor. For details on the augmentations used for training, see Section A.2.1.

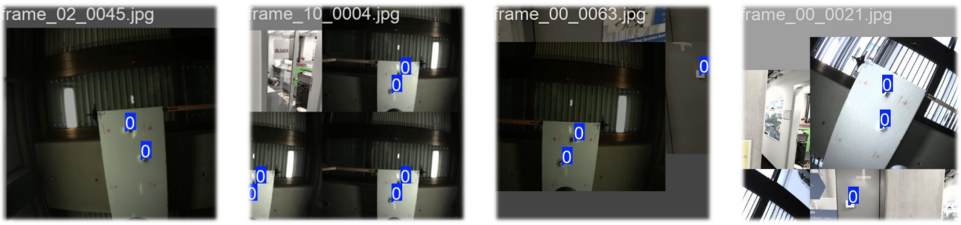


FIGURE 3.3: *Some example training images featuring augmentations like rotation, cropping, erasing, illumination change and mosaic augmentation*

Model Compiling for Efficiency Despite the remarkable efficiency of YOLO models during inference, it is still the most computationally expensive part of the entire system. For this reason and due to compatibility issues, the trained model had to be compiled to run natively on the drone's Jetson Orin NX. This was done using Nvidia's TensorRT framework [34]. First, the model was exported to the generic format `onnx` [35] using a built-in export function of the Ultralytics Python module. Then, TensorRT can be used to optimize and compile this generic model by leveraging low-level CUDA operations and eventually run it natively on a desired GPU model.
However, while this compilation step leaves the functionality of the network unaltered in principle, any pre- and post-processing steps provided by the Ultralytics module are lost. They were therefore reimplemented as close to the original version as possible by using the YOLOv8 source code. For details on this implementation, refer to Section A.2.2. All code related to the usage of the compiled YOLO model as part of the Tracker can be found in these files:

- `scripts/node_detector_yolo.py`
- `scripts/utils/multi_framework_yolo.py`
- `scripts/utils/tensorrt_inference.py`

3.1.2 Depth Fusion

Both detector types, the blob detector and the YOLO model, output only 2D points. However, since the Estimator will use the drone odometry as a tool for predicting future detections and interpolating temporally between them, any detections must be available as 3D points to the Estimator. A ToF camera was used as a source for the missing depth information of the detection points. It outputs a 640 x 480 pixel monochrome image of depth values and can be used to measure distances between 0.1 and 10 meters. This depth image is special in that it can also be interpreted as a three-dimensional point cloud. Pixel coordinates alone indicate a "ray" or direction on which a real-world point, which is measured by the ToF camera, must lie. The depth value of a corresponding pixel resolves the degree of freedom along this ray and allows us to reconstruct the coordinates of the original 3D world point.
Unfortunately, combining detections and depth information is complicated by the fact that the data originates from two sensors mounted at different locations. Furthermore, the raw output from the ToF camera is incomplete and noisy. Whenever reflections, transparent surfaces, or points that are too close or too far away are present, the ToF camera reports NaN values. Therefore, two different approaches for fusing the information were implemented: a normal-estimation-based approach and full sensor fusion.

Normal Estimation

A previous project at ASL that relied on depth estimation provided an algorithm to efficiently extract simplified measurements from the ToF camera. By only considering a small number of depth points in the center of the ToF camera frame, a plane is fitted through them. This yields one single estimation of the distance to a pivot point on the plane, as well as the plane normal direction. Knowing the characteristics of this plane in 3D space, it can then be transformed into the RGB camera coordinate frame and is extrapolated to determine the depth of any 2D detection point from the image coordinate frame. However, this approach is mainly suitable for experiments in which the target is surrounded by a single flat wall. For details on how the exact depth values are determined, refer to Section A.3.1.

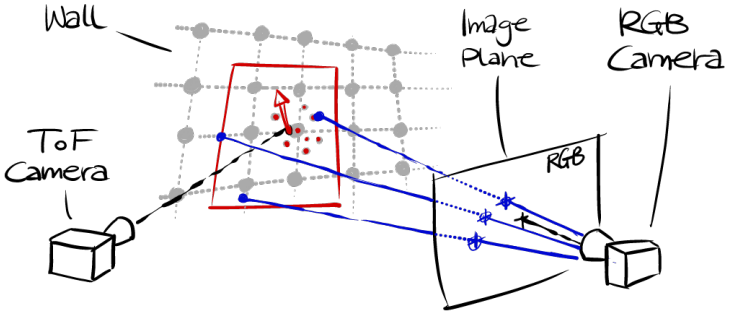


FIGURE 3.4: *Schematic visualization of the normal estimation process showing the fitted plane with an estimated normal direction.*

Full Sensor Fusion

For a more precise depth estimate, a complete sensor fusion was implemented simultaneously. The main challenges faced during the conception of this algorithm were computational efficiency and missing data in the ToF output.
Since the raw output point cloud from the ToF camera is quite granular, it was first aggressively downsampled to speed up any subsequent operations. By then transforming the downsampled points into the RGB camera coordinate frame, the depth information is now present in the same frame as the detections. However, simply finding the correct depth to a corresponding detection by taking values from the pointcloud can lead to ambiguous solutions and is more of an optimization problem than a straightforward equation (see Section A.3.2). To simplify this problem, all depth points are projected into the RGB camera image plane and binned into pixels – much like a standard image is formed. By heavily relying on numpy and vectorized operations, a new artificial depth image is created efficiently. Since it is now in the exact same image plane and coordinate frame as the RGB image, determining the depth value of a detection is as simple as a lookup of the value in the depth image at the corresponding pixel coordinates. For more details on the sensor fusion pipeline, refer to Section A.3.2. The implementation can be found in the `scripts/node_depthmap.py` file.

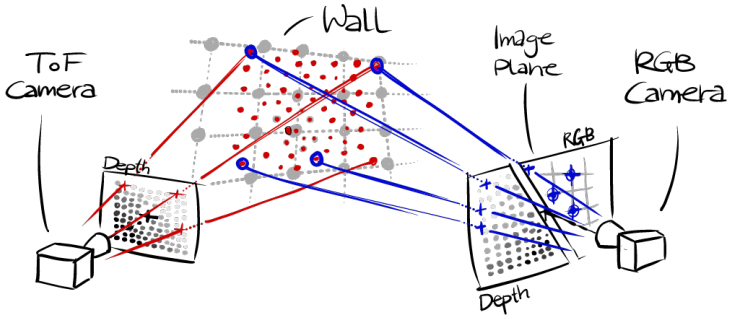


FIGURE 3.5: *Schematic visualization of the sensor fusion approach showing how the ToF measurement points are reprojected to form an artificial depth image for the RGB camera coordinate frame.*

3.1.3 Target Estimator

The Estimator module, tasked with cleaning up the detection points, ultimately outputs a single and consistent 3D estimation of the target location. Its functionality can be broken down into five distinct sub-algorithms (see Figure 3.6), each explained in detail in the following sections. The entire code related to the target Estimator is contained in the following files:

- `scripts/node_tracker.py`
- `scripts/utils/hole_tracker.py`

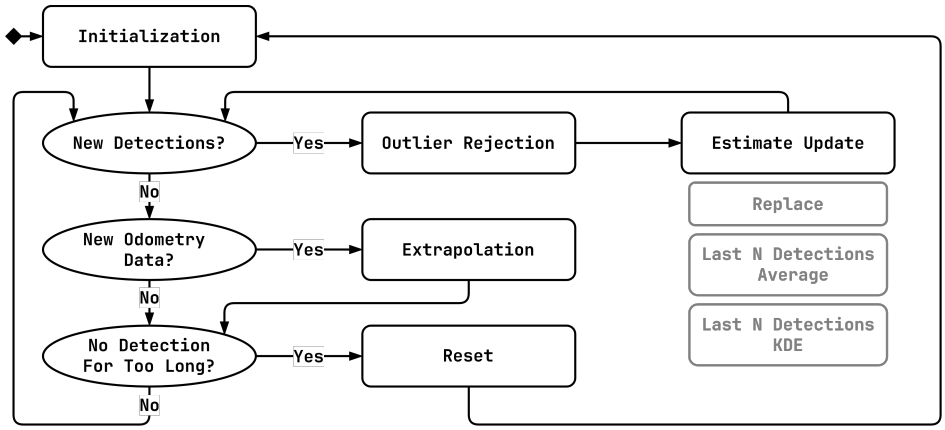


FIGURE 3.6: *A simplified flowchart explaining the Estimator runtime logic. After initialization, the estimator will continuously loop. Whenever new detections are available, outliers are removed and the estimate can be updated (different methods implemented). In case new odometry data is available, the existing estimate can at least be extrapolated. Finally, if no detection of the estimate has been made for too long, the Estimator will reset.*

Initialization

The initialization sub-algorithm is used when the Tracker is freshly started and has no current internal estimation of the target location. The only information available at that point is the raw 3D detection points. A pragmatic approach might be to just sample a random point from these detections and start tracking it under the assumption that it corresponds to the true target. In practice, however, this method is not robust and will predominantly lead to initialization on outliers. As a more robust alternative, multiple detections from different frames are collected over a short time span (around 1-2 seconds). These 3D points can then be aligned with the help of drone odometry data. This alignment can be subject to drift, but in practice, it is sufficiently robust to form a point cloud of recent detections.

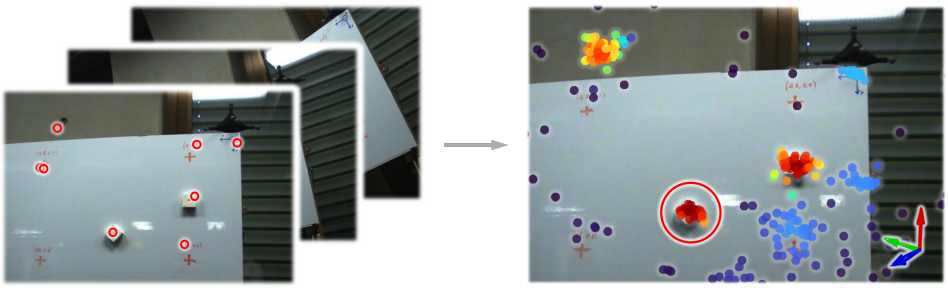


FIGURE 3.7: *During initialization, detections from multiple frames are aligned using the drone odometry data. On the right, a typical point cloud formed by this process can be seen. The red hues indiciate a high local density.*

In the next step, the densest point of this cloud is identified and taken as an initial estimate. Since a true target is likely to be detected repeatedly in the same location, whereas outlier points tend to be spread out more uniformly, this approach serves as a robust initialization method. To determine the local point density in a cluster,

Kernel Density Estimation [36] was used, as highly optimized implementations are available from the Scikit Python module. For details on the exact parameters used for Kernel Density Estimation, see Section A.4.

Outlier Rejection

After initialization, the Estimator now has an internal location, where it believes the target (a 3D point) is, relative to the drone's coordinate system. The challenge now lies in aligning and comparing this continually held internal estimate to any future detections – which are subject to outliers. Such a comparison would allow for coherent tracking of the target over time.
If both the drone and the target point were fixed with respect to each other, comparing the estimate to new detection points would be an easy task: outliers could be identified by just rejecting any points outside of a certain tolerance radius around the current estimate. On the other hand, a point aligning with the estimate could immediately be considered a true positive and could be used to update or refine the inferred target location.
However, in the case of our setup, the target point (fixed) is constantly moving relative to the drone since the drone itself is moving. For this reason, the estimate has to first be updated with knowledge of the system dynamics (drone odometry) before comparing it to new detections. The relative apparent movement of the estimate with respect to the drone coordinate frame can be determined through the following formula (rigid body kinematics):

$$\mathbf{v}_P = -\mathbf{v}_D - \boldsymbol{\omega} \times \mathbf{r}_{DP} \tag{3.1}$$

By propagating the estimate in this way, any movement of the drone is accounted for and matching with new detection points is possible again. If a detection falls within a small tolerance radius (typically 10 cm) of this updated estimate, it will be considered a confirming measurement. Any points outside are discarded. In case of multiple detections being close enough to the predicted estimate, the closest one is naïvely selected.

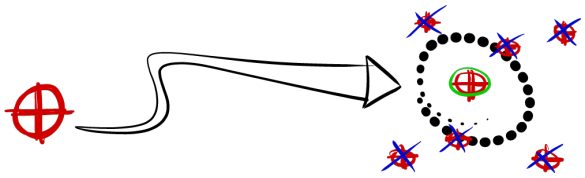


FIGURE 3.8: *A small tolerance radius is defined around the predicted location of the new detection. Any points lying outside can be rejected as outliers.*

This approach to outlier rejection is far from perfect, since it heavily relies on the prediction being sufficiently accurate, as well as on the rare occurrence of outlier points close to the actual target location. Fortunately, in practice, this method proved to be a very effective and simple solution to the outlier rejection problem. The time span between two detections in all experiments is well below one second (usually closer to around 0.1 seconds); therefore, drift in the odometry values is negligible. Additionally, for both Detector frameworks, outliers near the true target

are relatively uncommon. Especially in the case of the YOLO model, non-maximum suppression during post processing (see Section A.2.2), in fact, prohibits detections from being too close to each other altogether.

It should be noted, however, that using a fixed tolerance radius for the outlier rejection proved to be a poor choice in hindsight. The final evaluation has revealed that the overall precision of the Tracker is largely dependent on how close the target is to the drone (see Section 4.2). Therefore, for future experiments, a better approach would be to implement a variable radius that is automatically adjusted depending on the distance to the target - roughly following the proportional law determined by Tracker evaluation.

Estimate Update

As described so far, every time a new (non-rejected) detection is made, it would be used directly to fully replace the old estimate, and thus update the internal estimate. This approach of "fully trusting" the new measurement is simple and was generally found to work reliably during initial experiments. However, in the presence of a high number of outlier detections or outliers close to the true target, a particular failure mode could be repeatedly observed:
Since the tolerance radius for the outlier rejection must allow for some error in practice, on rare occasions, an outlier point can be accidentally accepted as a true target detection point. When this happens, the estimated target location rapidly deviates from the true location. If the deviation is large enough, another outlier point can subsequently be mistakenly seen as an acceptably close new detection point. As a result, the estimate starts to erratically deviate more and more from the true location. This behavior occurs because the estimate is allowed to be instantly updated to a fairly different position at every time step - ultimately restricted only by the tolerance radius.

To mitigate this, two new “update methods” were implemented and tested. Both rely on keeping a short history of the last confirmed detections instead of just considering the newest one. The location of these previous detections can be propagated in time by using the rigid body kinematics formula (3.1), much like for the initialization. Again, due to the odometry drift, the detection history must be sufficiently short and recent (around 10 detections are used in practice).
The two update methods only differ in how the estimated target location is then calculated from these historical detections. The first method just averages the 3D coordinates of all points. The second, once again, uses Kernel Density Estimation to determine a density for each point and then calculates the density weighted average of all 3D coordinates. This provides slightly more robustness against individual points deviating from the main cluster. Both new update rules force the estimate to be more persistent over time, and thus prohibit rapid deviation due to occasional erroneous detection points.

FIGURE 3.9: *A comparison of the average method (left) and KDE density weighted average method (right). Density weighting clearly allows the average to remain closer to the cluster "center" in the presence of outliers. Red hues indicate high local density.*

Extrapolation

During experiments on the platform, the Detector module typically runs at around 10 Hz. Consequently, the estimate would also be produced at the same frequency. Since it will subsequently be used by the Controller, increasing the estimation frequency could improve controller robustness during flights.
However, because the detection frequency is mainly limited by the platform's hardware, it cannot be easily increased. Therefore, "extrapolation" is used. Using drone odometry knowledge again (available at around 200 Hz) and applying the same prediction formula as before (3.1), a target estimate of much higher frequency can be obtained. From the last available detection point, it is extrapolated smoothly and linearly to achieve an arbitrarily granular temporal resolution – at least during a short time span, until deviation due to drift increases.

Resetting

Lastly, the Estimator has an automatic resetting mechanism. Since extrapolation with the rigid body kinematics formula (3.1) and drone odometry alone eventually suffers from drift, it should only be used over short periods of time (typically 1-2 seconds). For this reason, the Tracker automatically resets to the initialization step when no detection confirming the current target estimate is registered within a certain time limit.

3.2 Controller Module

With a consistent 3D target estimate now available, a controller can be implemented with the task of flying the drone to this target and holding a steady position. Since an existing and proven control stack is already implemented on the hexacopter platform, a lot of work could be reused - but the existing architecture had to be obeyed when designing the controller for this project. Conveniently, all the low-level controls that convert requested body accelerations into motor inputs could be used without any modifications.
For higher-level path planning controls, the existing stack uses Riemannian Motion Policies [3, 4]. This framework allows multiple "policies" to work together and accomplish intricate tasks. Each policy is in essence a small controller with the role of regulating a standalone task. The output of each policy is an acceleration vector

(usually a full drone body acceleration). By using a custom “metric” for each policy, they can then be combined in a sensible way, similar in spirit to an optimization formulation. This prevents mutual cancellation of the policies’ outputs - a common issue that would occur when combining them through naïve averaging.

The controller for the peg insertion task of this project was therefore also divided into subtasks, each handled by a separate policy. Any movement parallel to the target wall is controlled by a visual servoing policy. All attitude controls are taken care of by a "normal keeping" policy. Finally, distance to the wall – the remaining degree of freedom – is handled by a "distance keeping" policy.
The main benefit of splitting the control task into different subtasks is modularity. Each policy can be configured and possibly replaced independently from the others. This enhances the general applicability of the controller. In case a different task is approached in the future, which might require different control strategies, individual policies could be modified and reused instead of completely redesigning the control architecture. In addition, by being able to test and run policies in an isolated manner, development is greatly simplified.

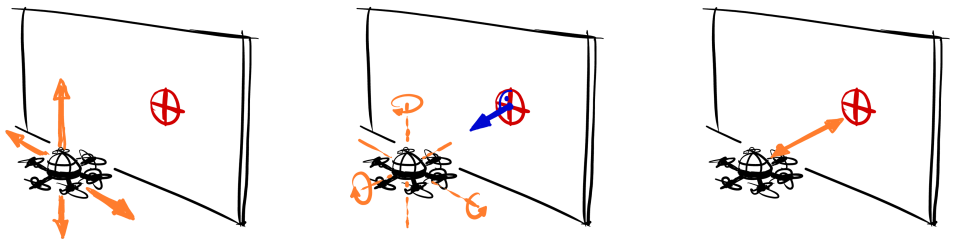


FIGURE 3.10: *The degrees of freedom that each control policy regulates (from left to right): the visual servoing policy, the normal keeping policy and the distance keeping policy.*

3.2.1 Visual Servoing

The choice for a visual servoing approach to control movement parallel to the target wall might seem curious, given that the previously discussed Estimator outputs a 3D point. However, the benefit of using a 2D visual servoing approach is compatibility with both 2D and 3D estimates. If, in a future project or task, a 2D-only Estimator were to be used, this Controller could still function without any modifications. At the same time, having a three-dimensional estimate does not hurt, as it can simply be converted to a two-dimensional one.

This policy has the goal of visually aligning the target estimate and the tip of an arbitrary end effector. Since the tip of the end effector appears at a fixed location in the image, it is not detected automatically, but rather set as a parameter once. The target location, on the other hand, is determined by projecting the 3D estimate point onto the normalized image plane.
A simple PD controller is then used to reduce the distance between these two points. The immediate control output is therefore a 2D acceleration vector in the image pixel space. However, since the low-level controller requires drone body acceler-

ations, the pixel accelerations must be transformed first. Thankfully, the image Jacobian allows us to do exactly that. It can be leveraged to determine a pixel velocity at any point in an image as a function of the velocities of the camera body - and vice versa [37]. With some simplifications, the same law can be used for accelerations as well. (for details, see Section A.5).

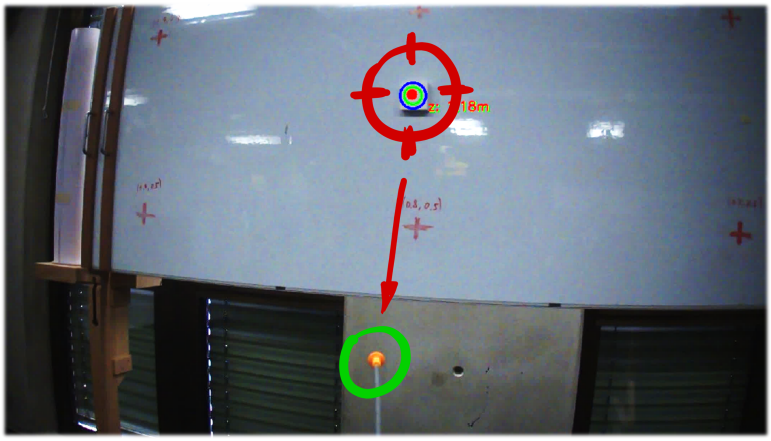


FIGURE 3.11: *The position of the end effector in the image (green) is fixed. The visual servoing policy accelerates the drone in such a way that the target (red) eventually aligns visually with the end effector.*

However, as the image Jacobian returns full 3D accelerations of the drone body (to achieve a given pixel velocity in the image) the output had to be constrained slightly - the visual servoing policy should only control movement parallel to the target wall. For this, a "metric" was used in the form of a projection matrix. A rank-2 projection matrix (3.2) naturally removes all components of a three-dimensional vector that do not lie inside the planar vector space spanned by the projector and hence is perfect for this task. The two vectors that constitute the projection matrix were chosen to be perpendicular to each other, as well as to the end effector axis. As a result, any acceleration along the end effector axis is eliminated. As this axis will ultimately be kept perpendicular to the wall, the projection results in only accelerations parallel to the wall remaining.

$$\mathbf{P} = \mathbf{v}_1\mathbf{v}_1^\top + \mathbf{v}_2\mathbf{v}_2^\top = \mathbf{I}_3 - \mathbf{n}\mathbf{n}^\top \tag{3.2}$$

3.2.2 Normal Keeping

During the peg insertion task, the drone end effector must always remain perpendicular to the target wall to be able to insert the peg successfully. Therefore, all attitude controls are taken care of by a normal keeping policy. A previous project at ASL provided this policy, and it could be used without significant modifications. However, it should be noted that normal keeping only constrains two out of three rotational degrees of freedom. Any rotation around the end effector axis is not controlled directly by this paradigm. Nevertheless, additional existing stabilization policies force the drone to stay horizontal if possible; therefore, effectively also locking this third rotational axis.

3.2.3 Distance Keeping

The distance keeping policy is arguably the simplest: for all experiments of this thesis, the remaining linear degree of freedom was controlled by an operator. This provided fine-grained control over the execution speed of any experiments and therefore greatly benefited safety.
However, as the ToF camera already provides depth information, implementing an automatic distance keeping policy would be trivial – especially since movements along this axis are generally required to be slow and predictable.

3.3 End-Effector Hardware

The drone platform used for this project features a generic mounting point at the tip of one of the rotor arms. For the initial experimental flights, a simple rigid rod was mounted on this arm. However, using such a setup to make contact with the target hole proved to be challenging. As soon as the drone contacts a rigid object, the tip of the end effector becomes constrained with a large leverage, thus overwhelming the controller with the sudden mode change. In practice, the platform would crash repeatedly, especially due to the normal keeping policy starting to oscillate and becoming unstable upon contact. These findings align with the challenges described by other research groups (see Section 1.1): the mode change from free flight to contact with a rigid object is often mentioned as hard to overcome.
A solution to this problem is typically some form of compliance in the system – either physically in the end effector or in the form of force-compliant controls. Due to time constraints, only the end effector compliance was explored for this thesis. Three different attachments were designed and manufactured, each featuring a different combination of compliant directions.

All subsequent parts were fabricated using a Prusa Mini 3D printer with a 0.4 mm nozzle and PLA filament. The part models were designed using OnShape [38].

3.3.1 Bending Compliance

The first attachment features bending compliance in two directions. A 2 mm diameter stainless steel rod is used as the main source of compliance. The rod is mounted with a 3D printed base, and a small plastic tip is added for better visibility in the camera image. All three parts are held together by a tight slip fit and, therefore, the attachment can be disassembled as needed, and parts can be replaced or adjusted.

3.3.2 Bending and Length-Wise Compliance

The second attachment combines the existing bending compliance with additional length-wise compliance. This is accomplished by adding a pogo-mechanism to the previous design. In this mechanism, a retained pin can slide freely inside a sleeve, while the elasticity is provided by a captive spring from a ballpoint pen.

3.3.3 Full Linear Compliance

Finally, an attachment with full linear compliance and, ideally, no rotational freedom was constructed. The main component of this attachment is a compliant hinge mechanism, allowing for linear movement in the two directions perpendicular to the end effector axis. This stage was then combined with the previous pogo-mechanism to achieve full linear compliance.
The hinge mechanism uses thin plastic strips embedded in the 3D printed parts to achieve high flexibility and durability. Since it provides almost no elasticity, rubber bands were used to center the main axis with an elastic pulling force. Additionally, to mitigate oscillations during flight, a damping mechanism had to be installed: a magnet attached to the moving axis induces eddy currents in a statically mounted copper plate, and thus dissipates energy. The major benefit of using such contactless damping in combination with compliant hinges is the total absence of static friction, which results in a very smooth and predictable motion.

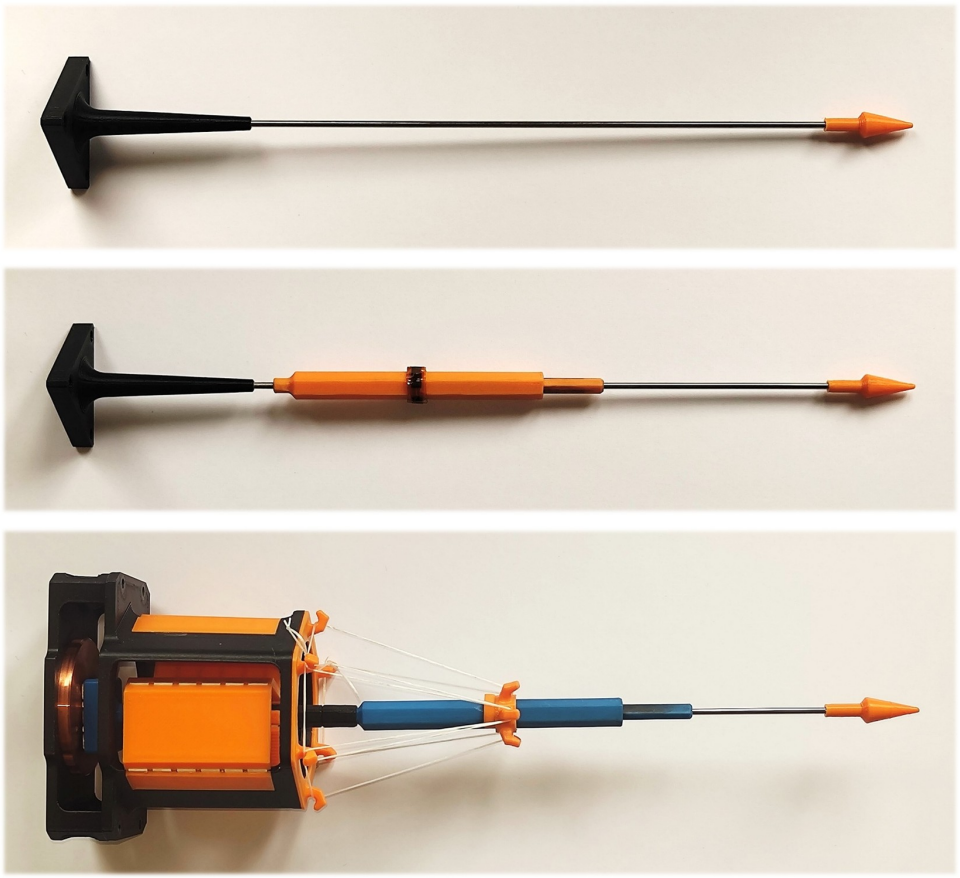


FIGURE 3.12: *All attachments from top to bottom: bending only compliance, bending and length-wise compliance and, finally, full linear compliance.*

4 Evaluation

The evaluation of the system that performs the peg insertion task is divided into multiple parts. Due to their modularity, the Detector and Tracker modules could be independently evaluated. Helpful insights could be gained regarding the advantages and drawbacks of the different Detector frameworks (see Section 4.1) and the overall Tracker precision (4.2). However, the Controller module, as well as the end effector attachments, had to be tested together with the other components since they cannot function alone. The findings of these full experiments are discussed in Section 4.3.

4.1 Detector Evaluation

For evaluating both the blob detector and the trained YOLO model, a video of the experimental setup with the target was recorded on the drone onboard camera. Five different scenarios were covered to ensure that the footage is representative of a real experiment run:

- Close to the wall (around 1m) while the target is being viewed from a perpendicular direction.
- Medium distance from the wall (around 2m), again, viewed perpendicularly.
- Far away from the wall (around 3m), viewed perpendicularly.
- Viewing the target from different horizontal angles, continuously varied between $\pm 120°$. A medium distance from the target (around 2m) was chosen for this scenario.
- Including rotation around the camera axis of $\pm 180°$ while being at a medium distance from the wall (around 2m) and viewing the target perpendicularly. For all other scenarios, rotation around the camera axis was excluded.

Then, around 60 static frames were extracted at regular intervals from each video, so that they cover all commonly encountered viewing scenarios. The location of the target in these frames was manually annotated by using the same tool that was used to label the YOLO training data [32]. Both variants of the detector were then set up to extract the target location in all of these images. Having access to the true location for each frame allowed for an easy evaluation of detector performance. Two metrics were recorded for each module (refer to Tables 4.1 and 4.2).
First, the percentage of frames in which a Detector correctly identified the target location was tracked (true positives). This metric directly measures how often a useful detection can be expected from either framework.

Additionally, whenever such a true positive detection was recorded, the total number of detection points for that frame was also noted – effectively measuring the number of outlier points. This metric is useful for gauging the precision or "signal-to-noise ratio" of the detector.

		Close	Medium	Far	Angled	Rotated
Blob	**detect rate**	**89.66%**	**81.82%**	**46.15%**	**20.90%**	**52.77%**
	# detections	3.66	14.18	10.13	11.57	7.88
Yolo	**detect rate**	**100.00%**	**92.73%**	**26.15%**	**26.87%**	**39.36%**
	# detections	1.12	1.14	1.00	1.17	1.19

Table 4.1: *Evaluation results for the blob tracker and YOLO model on a dataset featuring the standard target and five different scenarios.*

Looking at the results of this evaluation, some observations can be made (see Table 4.1). For the first three scenarios, performance is expected to drop with increasing distance from the target - which is clearly reflected in the data shown. This effect is largely due to the absolute size of the target decreasing and slowly nearing the camera resolution limit. Additionally, motion blur is amplified by decreased object size, thus distorting it and making it harder to detect (see Figure 4.1).
Viewing the target at an angle poses the challenge of significantly distorting the target visually, thus making it harder for both tracker frameworks to correctly detect it. Rotation, on the other hand, does not distort the target per se; however, a significant amount of motion, and consequentially blurring, is introduced, thus degrading performance as well.
At least when considering the detection rate metric, both detector frameworks seem to perform equally - especially given that some discrepancy is expected, since a lot of variables influence vision algorithms: motion, lighting, parameter tuning, and resolution, just to name a few.

Figure 4.1: *Examples of how the visual quality of the target decreases with further distance from the target.*

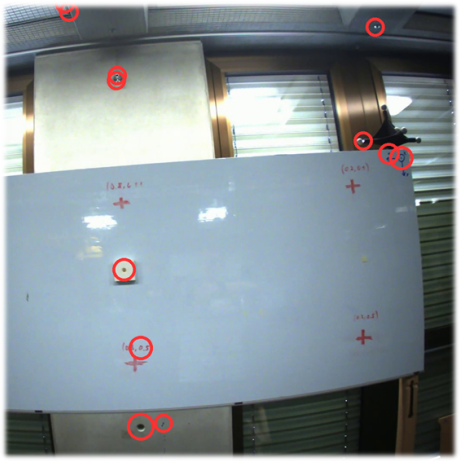


Figure 4.2: *Some examples of blob tracker outliers. A concrete plug remnant or Vicon markers appear visually similar to a hole.*

However, things look different when considering the number of outlier points per true positive detection. While the YOLO model rarely even detects any additional points (around 0.15 on average), the blob detector outputs an abundance of false positives (around 8.5 on average) (see Table 4.1). Still, this result is not unexpected, as both frameworks are fundamentally different and are not intended for the same purpose.
A YOLO model is trained to be very selective and identifies complicated objects that do not even necessarily appear visually uniform within the same class. The blob tracker, on the other hand, is a very general and simple tool that only uses a small set of parameters to discern uniform shapes. By inspecting some test images, the blob detector outlier points can often be justified, as they have a strong resemblance to the true target hole (see Figure 4.2).

To further gauge the selectivity and generalization of each detector Module, a second dataset was used. The square shape of the target was masked with the help of an arbitrary paper cutout, leaving only the hole itself visible. The same evaluation as before was then performed (refer to Table 4.2). Changing the prominent shape of the target can help us to find out whether the YOLO model is actually detecting the hole or rather the shape or environment around it.

		Close	Medium	Far	Angled	Rotated
Blob	**detect rate**	**72.92%**	**44.39%**	**26.19%**	**20.55%**	**38.12%**
	# detections	2.66	7.60	10.64	9.13	4.85
Yolo	**detect rate**	**4.17%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**
	# detections	1.00	–	–	–	–

Table 4.2: *Evaluation results for the blob tracker and YOLO model on a dataset featuring a masked target. The square shape was occluded with an arbitrary paper shape. However, the hole remained visible at all times.*

Clearly, the YOLO model is overly selective, and performance drastically deteriorates when the shape around the hole is changed – albeit the hole itself remaining unaltered. In comparison, blob tracker performance is largely unaffected by such target masking. However, this is unsurprising, as the YOLO model has been pretrained to learn to extract rich generalized features that can be used to identify complex objects. An isolated hole is not a descriptive feature at all, and naturally the learned model searches for other clues that can identify the target location more reliably - in this case the prominent shape around the target.
It should be noted that moderate performance differences between the two datasets are expected, as they are recorded by hand and are not exactly identical. The masked dataset specifically was subject to more motion blur, due to the camera being moved faster and the exposure being set slightly different.

4.2 Tracker Evaluation

Evaluating only the Detector modules omits the entire estimation process of the Tracker module. Therefore, the full Tracker was also evaluated. However, three components of it feature different implementation versions:

- Depth information: sourced through normal estimation (see Section 3.1.2) or full sensor fusion (see Section 3.1.2).
- The estimate update method can use just replacement of the old estimate with a new measurement, averaging of the last few detections, or even a density weighted average of them (refer to Section 3.1.3).
- Finally, both Detector types (blob tracker and YOLO model) can also influence Tracker performance (see Section 3.1.1).

Therefore, a method for evaluating all possible combinations in a reproducible and comparable manner had to be found. By recording a rosbag of all sensor data during a drone flight, it was possible to simulate a reproducible experimental setup simply by replaying the bag.
The Tracker module was then executed in all possible configurations successively. The estimated target location was recorded during each run and could be compared with the known true location (measured with a Vicon motion capture system). An absolute distance error between the estimated and the true location could then be determined. However, simply averaging this error over the whole experiment would not yield insightful results, since the tracker precision largely depends on how far away the drone is from the target. Therefore, all recorded errors were binned and plotted to show the average error for a certain range of distances from the target (all results are summarized in Figure 4.3).

First, the blob tracker was used in conjunction with all combinations of update methods and depth frameworks (see Figure 4.3, top half).
It should be noted immediately that in the case of the replace update method, tracker performance was extremely unstable and not reproducible, as the estimate tends to erratically deviate from the true location (described in Section 3.1.3). Error measurements where the target had clearly unrecoverably deviated from the true location were removed for these two plots to show some comparable datapoints. However, true performance can deviate arbitrarily and quite significantly, and the results of the replace update method in conjunction with a blob tracker should be treated with caution.
Interestingly, this shows that the more robust update methods work as intended. They are able to better tolerate a significant number of outliers, and the target estimation never deviates unrecoverably. There seems to be a slight performance difference between the density (KDE) weighted average and the standard average. This could indicate that the weighted average is indeed slightly less susceptible to outliers. However, the difference is only visible when the drone is close to the target and is not very pronounced.

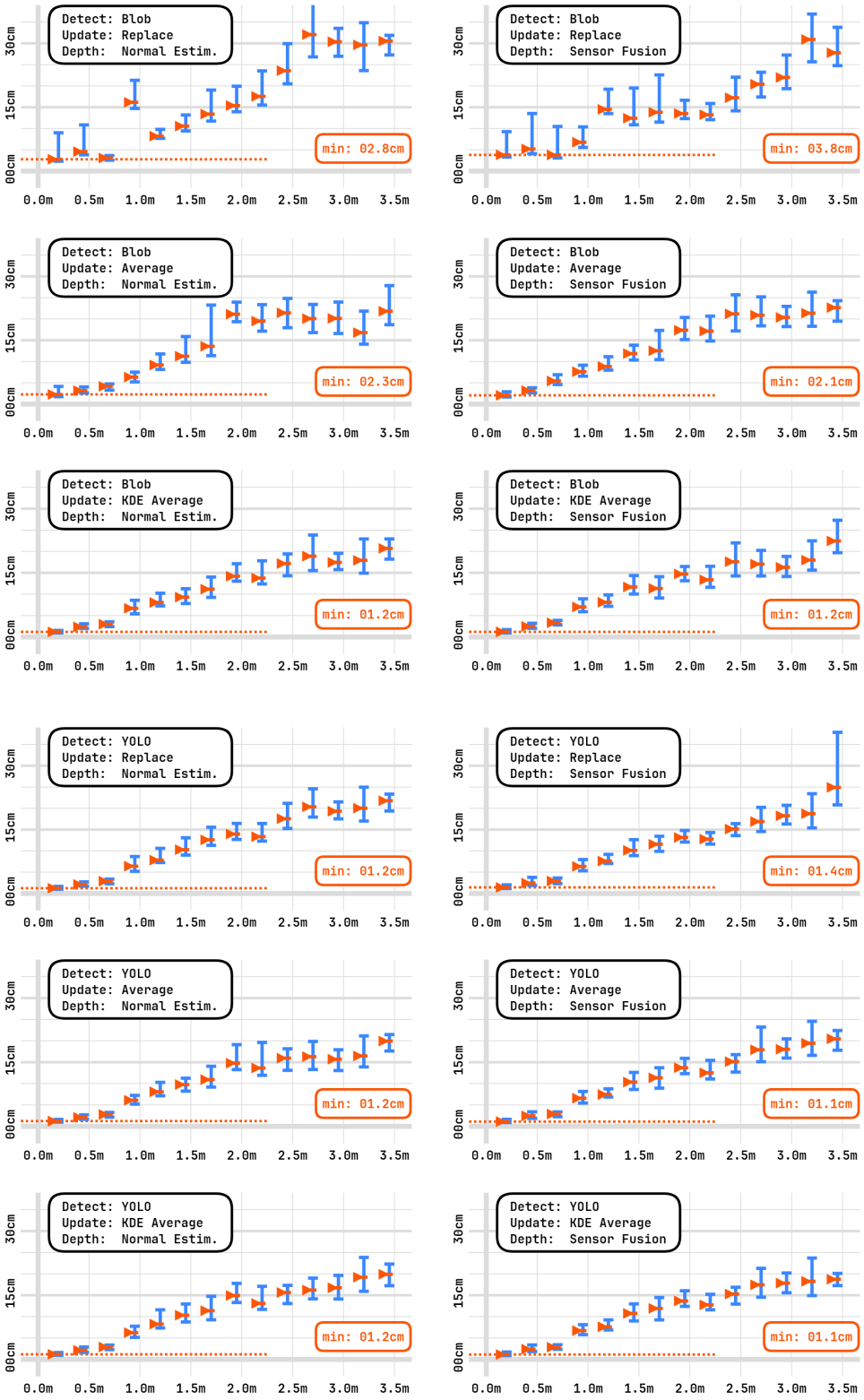


FIGURE 4.3: *A summary of all tracker evaulation results. Each y axis is the average precision per distance range and the x axis indicates distance from the target*

Analogously, the performance of the Estimator in conjunction with the YOLO Detector could also be evaluated (see Figure 4.3). Here, the results for all the different combinations appear to be similar. A well-trained YOLO model is extremely selective and only rarely produces outlier detections (see Section 4.1). Therefore, any benefits of the more robust update methods are mostly negated.

Interestingly, the last factor - the source of depth information - had absolutely no influence on the performance in any setting. This is likely due to the fact that the target is mounted on a relatively large planar wall. Depth values anywhere on the wall are well approximated by the normal estimation - the benefits of full sensor fusion are therefore neglected.
However, in earlier experiments, the ToF camera was mounted differently, leading to around a quarter of the field of view being obstructed by the rotor arm. In that case, the normal estimation was often not robust, as sometimes points from the drone arm were mistakenly used for the estimation. The full fusion approach was able to provide a much more stable depth measurement in that case, as it could fill in missing values and certain parts of the ToF image could be excluded.
Additionally, the sensor fusion approach could be beneficial for future tasks where a target is not located on a flat surface. In such a scenario, normal estimation would not be reliable.

Throughout all tracker evaluation runs, a very similar proportional relationship could be observed between the estimate precision and the distance to the target. This is explained by multiple factors.

- A target that appears visually smaller and is further away leads to any detector errors being amplified in absolute terms.
- The ToF camera has a precision specified as a percentage of the distance measured. Therefore it is more precise, the closer an object is – at least until a certain cutoff distance, which was never undercut.
- The motion of the target point relative to the drone due to rotation is amplified proportionally with distance. Faster relative movement of the target implies that any extrapolations made by the Estimator using the drone odometry are over a further distance and therefore less precise.
- At larger distances, the target size shrinks so much that it nears the resolution limit of the camera. This leads to rarer true detections (as seen in Section 4.1) and thus more reliance on extrapolation, which is prone to drift.

4.3 Full System Evaluation

The complete system, including the Controller module and the custom end effector attachments, was evaluated through real flight experiments, as these components cannot be independently tested. For all tests, the drone was stabilized roughly in front of the target wall. Then, the rotor belonging to the camera and end effector arm was disabled to allow the target to be safely approached. The Controller

and Tracker modules were then enabled, and the distance to the target was slowly manually decreased until contact with the target was established. The following configuration of Tracker components was used during all final experiments: a YOLO model in conjunction with the density weighted average update method. Full sensor fusion was used to source the depth information.
In most cases, some oscillations were present in the system, requiring the operator to wait a short duration until the target hole could be successfully entered with the end effector tip. After insertion, contact with the target was maintained for approximately 10 seconds to assess stability and wait for possible adverse effects or control failures to occur.
Interestingly, the Tracker can register the target even while the end effector is in contact with it. This is likely a benefit of using a very low-profile attachment and might not be guaranteed in the case of a design that more prominently occludes the target. This means that in conjuction with the robustness of the Controller and end effector compliance, contact can be maintained more or less indefinitely.

Controller and Sytem Precision The Controller proved sufficiently robust to perform the task repeatably. All control policies required manual tuning, but a set of gains could be found that consistently resulted in successful insertions. The overall system precision was estimated to be approximately 2-3 cm. This is based on the observation that the larger target block (of roughly that size) could be reliably hit on every attempt, whereas insertion into the smaller hole (around 0.8 cm) often required a short settling time due to oscillations and multiple attempts.
One limitation was observed when all propellers were used for flying. Since the RGB camera is mounted on the rotating arm, it can be subject to significant movement if the same arm is used to steer the drone simultaneously. This led to a sort of coupling between Tracker and Controller and resulted in ocillations. However, by lowering the gains for the visual servoing policy, this problem was largely mitigated - nevertheles, stability is improved by disabling the camera arm during flights.

Compliant Attachments The initial catastrophic destabilizing behavior that was present after contact with the rigid target could be resolved by all three compliant attachments equally well. Allowing for a small amount of imprecision in the movement significantly reduces the external forces on the drone - the forces that likely destabilized the controller before. In addition to that, length-wise compliance greatly helped in maintaining prolonged contact. As there is also some imprecision in the movement perpendicular to the wall, having a spring-loaded pin allowed the end effector tip to stay inside the hole at all times. Without such compliance, contact was often lost after even just slight movement.
The full linear compliance attachment, although working well, is likely unnecessarily complicated for the peg insertion task. Due to its size, it occludes a larger part of the camera and there are more parts that can break. On the other hand, it does allow for off-axis rotation and keeps the end effector axis perpendicular to the wall - even under movement, which is not the case for the two bending attachments. These properties could potentially be useful for a future manipulation task that requires them, for example screwing.

5 Conclusion

5.1 Goals and Constraints

Clearly, the initial goal of performing a reactive peg insertion task was successfully completed in this thesis and could be reproduced multiple times during the final experiment. Through evaluating the task execution and additional individual experiments, valuable insights regarding aerial manipulation could be gained.

Most importantly, the precision of the entire system is estimated to be around 2-3 cm for a single point target - an area of this size can repeatedly be hit with high confidence. Furthermore, compliance is crucial for aerial manipulation involving contact with a rigid object. Best suited for the peg insertion task was a combination of bending and length-wise compliance. Regarding the controller, a visual servoing policy in conjunction with a normal and distance keeping policy proved to be effective in steering the drone to the point target. However, a rigidly mounted camera with an unobstructed view is required for the visual servoing approach.

Initially, two design constraints, modularity and minimal sensor usage, were established for this project. With almost every component being easily replaceable and reconfigurable, this constraint was definitely adhered to. As a consequence, multiple variants of some components could be evaluated and insightful measurement data were collected. Minimal sensor usage, on the other hand, could unfortunately not always be satisfied. In an early decision, optical flow was considered as an estimate of the system dynamics but was discarded due to computational cost and quality concerns. To err on the side of caution, the drone odometry was directly used and a ToF camera had to be included as a consequence. In hindsight, especially after seeing how steadily the drone flies when steered by the Controller module, using an optical-flow-only-based estimation approach would likely be feasible.

5.2 Components

5.2.1 Detector Module

Both Detector frameworks explored during this thesis worked satisfyingly well for the peg insertion task. Detection rates of around 25% for poor visibility of the target and up to 70-100% for good visibility are possible with both implementations. Nevertheless, they exhibit different behavior in some scenarios and have unique strengths and shortcomings.

The blob tracker is easy to set up and requires only minimal manual tuning of its parameters. A simple shape like the hole for the peg insertion task can be easily detected with a tuned blob tracker, but it is not very selective in doing so. By design, generally many outlier points are also registered (around 8.5 per true detection). However, this generalizing behavior is advantageous in case different types of hole must be tracked in the future.
The YOLO model, which was finetuned for this project, was harder to implement, as it requires generating the training data and setting training parameters. However, thanks to the Ultralytics Python API, this process is straightforward and comfortable. As a result, a Detector with strong selectivity (around 0.15 outliers per true detection) could be implemented with limited effort. This property of the YOLO model proved to be valuable for the subsequent Estimation process as less intricate methods for cleaning the data are needed.
One downside of the YOLO detector is over-selectiveness. By masking the shape around the target hole, it could be shown that the model does not generalize to detect holes, but rather uses more prominent features around it to identify the target. This implies that a YOLO detector must retrained to work with different types of holes.

5.2.2 Tracker Module

The approach of combining detections and predictions (using drone odometry) to form a consistent estimate resulted in a robust and precise Tracker. Errors as low as 1.2 cm could be achieved for close distances to the target, deteriorating to around a 20 cm precision at three meters of distance. Furthermore, a strong proportional dependence could be shown between precision and distance from the target. Detections from the YOLO model are reliable enough so that no special attention has to be paid to estimate formation during the target tracking. However, the blob tracker greatly benefited from storing a short history of detections and averaging them to make the estimate more robust.

Regarding the source of depth information, both implemented systems worked equally well for the peg insertion task. Since a large flat wall surrounds the target, normal estimation is a good approximation. However, for other tasks, where the target is located on a less uniform surface, or in case of severe occlusion of the ToF camera, the sensor fusion approach can be advantageous.

5.2.3 Controller

The combination of a visual servoing policy with a normal and distance keeping policy was an effective and simple solution for the peg insertion task. Modularity is especially useful as policies can be tuned independently. However, having the camera mounted on a rotating arm is subobtimal for visual servoing since it introduces a lot of movement and can lead to an undesired coupling of the Controller and Tracker. Therefore, a fixed mounting location should be preferred.

5.2.4 End Effector Hardware

Including some form of compliance in the end effector was nonnegotiable as allowing for slight imprecision in the drone movement relieves the controller from strong external forces. A combination of bending and length-wise compliance proved to be optimal for the peg insertion task. This not only decouples the external forces but also allows for prolonged contact with the target by allowing for some error in the distance to the wall as well.

6 Outlook

Despite the satisfactory experimental results of the peg insertion system designed for this thesis, there still remain some aspects that might be improved or changed in the future - especially after considering some of the insights this thesis provides.

Future Tasks and Corresponding Adjustments Initially, peg insertion was chosen as a fitting task, as it includes many of the challenges commonly faced in aerial manipulation. However, a task like screwing might be more relevant to real world applications and includes additional difficulties, such as maintaining prolonged contact with a rigid surface and actuation of the end effector while in contact with the target. Screwing might therefore be an interesting next step and would certainly require an additional component: a state machine that monitors the whole execution with the responsibility of starting and stopping the screwing motion itself. Importantly, the position of the RGB camera would definitely have to be changed to a fixed location on the drone body in case the screwing task is pursued. Rotating movement would be a difficult challenge for the visual servoing policy, which is already susceptible to smaller movements from just free flight.
For future tasks, in general, it could be beneficial to have the option of manually choosing between a few different possible targets registered by the Tracker. This might be implemented through a simple user interface and would offer a bit more flexibility in the case of multiple targets being available.

Tracker and Estimator Improvements A large part of the work during this project was spent developing a robust object Tracker, exploring different subcomponents of it, and evaluating the performance. Considering the strong precision-distance dependence that was discovered during the Tracker evaluation, a small but important change could be made regarding the outlier rejection. Instead of a fixed tolerance radius, a variable size could be used. A fitting scaling factor can easily be determined by considering the precision evaluation data. This would likely extend the effectiveness of outlier rejection across a broader range of distances from the target.
Furthermore, evaluating Tracker performance outside of the motion capture environment could provide more realistic results. At the moment, the drone odometry that is used by the Estimator is fused with measurements of the velocities from a motion capture system. In a real world scenario, a standalone state estimator generally still provides odometry data - however, such data might be subject to more drift. Since the Estimator only uses odometry to bridge short detection gaps (around 0.1 seconds), this is unlikely to have a devastating impact. Nevertheless, such an evaluation could provide helpful insights.

Additionally, an optical-flow-only estimation approach could be explored, as it would likely be robust enough in hindsight. Optical flow would also help mitigate the problem of degrading odometry estimation quality outside of a motion capture environment, since it is not subject to any drift - at least regarding velocity estimation.

Control Improvements A very logical next step for the controller would be to automate the distance keeping policy. All the information needed is available, as depth information is already used by the Tracker.
In a broader sense, the controller accuracy could likely be improved by directly controlling the end effector frame - and not the drone body as is done currently. Since the total length of the drone arm and end effector attachment is close to one meter, any imprecision in body movement is amplified significantly.
Lastly, the addition of a control law that can eliminate static tracking errors should be considered. As currently only PD controllers are used for all policies, offsets from the control setpoints were commonly observed, even in an equilibrium state. Diminishing these errors could especially benefit the visual servoing policy, since this static error had to be compensated by offsetting the end effector tip location in the image internally.

Bibliography

[1] Z. Teed and J. Deng, "RAFT: Recurrent all-pairs field transforms for optical flow."

[2] ROS - robot operating system by open robotics (www.ros.org).

[3] N. D. Ratliff, J. Issac, D. Kappler, S. Birchfield, and D. Fox, "Riemannian motion policies."

[4] M. Pantic, L. Ott, C. Cadena, R. Siegwart, and J. Nieto, "Mesh manifold based riemannian motion planning for omnidirectional micro aerial vehicles," vol. 6, no. 3, pp. 4790–4797.

[5] "The swiss drone industry report 2024 v1.4 (drone industry association switzerland)."

[6] T. Ikeda, S. Yasui, M. Fujihara, K. Ohara, S. Ashizawa, A. Ichikawa, A. Okino, T. Oomichi, and T. Fukuda, "Wall contact by octo-rotor UAV with one DoF manipulator for bridge inspection," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.

[7] F. Forte, R. Naldi, A. Macchelli, and L. Marconi, "Impedance control of an aerial manipulator," in *2012 American Control Conference (ACC)*. IEEE, pp. 3839–3844.

[8] H.-N. Nguyen, S. Park, and D. Lee, "Aerial tool operation system using quadrotors as rotating thrust generators," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.

[9] G. Hunt, F. Mitzalis, T. Alhinai, P. A. Hooper, and M. Kovac, "3d printing with flying robots," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 4493–4499.

[10] H. W. Wopereis, J. J. Hoekstra, T. H. Post, G. A. Folkertsma, S. Stramigioli, and M. Fumagalli, "Application of substantial and sustained force to vertical surfaces using a quadrotor," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 2704–2709.

[11] V. Nayak, C. Papachristos, and K. Alexis, "Design and control of an aerial manipulator for contact-based inspection," version Number: 1.

[12] J. Sugihara, T. Nishio, K. Nagato, M. Nakao, and M. Zhao, "Design, control, and motion strategy of TRADY: Tilted-rotor-equipped aerial robot with autonomous in-flight assembly and disassembly ability," vol. 5, no. 10, p. 2300191.

[13] M. Ryll, G. Muscio, F. Pierri, E. Cataldi, G. Antonelli, F. Caccavale, D. Bicego, and A. Franchi, "6d interaction control with aerial robots: The flying end-effector paradigm," vol. 38, no. 9, pp. 1045–1062.

[14] G. He, Y. Jangir, J. Geng, M. Mousaei, D. Bai, and S. Scherer, "Image-based visual servo control for aerial manipulation using a fully-actuated UAV," version Number: 1.

[15] M. Schuster, D. Bernstein, P. Reck, S. Hamaza, and M. Beitelschmidt, "Automated aerial screwing with a fully actuated aerial manipulator," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 3340–3347.

[16] D. Kim and P. Y. Oh, "Human-drone interaction for aerially manipulated drilling using haptic feedback," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 9774–9780.

[17] V. Lippiello, G. A. Fontanelli, and F. Ruggiero, "Image-based visual-impedance control of a dual-arm aerial manipulator," vol. 3, no. 3, pp. 1856–1863.

[18] A. Ollero, J. Cortes, A. Santamaria-Navarro, M. A. Trujillo Soto, R. Balachandran, J. Andrade-Cetto, A. Rodriguez, G. Heredia, A. Franchi, G. Antonelli, K. Kondak, A. Sanfeliu, A. Viguria, J. R. Martinez-de Dios, and F. Pierri, "The AEROARMS project: Aerial robots with advanced manipulation capabilities for inspection and maintenance," vol. 25, no. 4, pp. 12–23, publisher: Institute of Electrical and Electronics Engineers (IEEE).

[19] D. Tzoumanikas, F. Graule, Q. Yan, D. Shah, M. Popovic, and S. Leutenegger, "Aerial manipulation using hybrid force and position NMPC applied to aerial writing," version Number: 1.

[20] S. Hamaza, I. Georgilas, M. Fernandez, P. Sanchez, T. Richardson, G. Heredia, and A. Ollero, "Sensor installation and retrieval operations using an unmanned aerial manipulator," vol. 4, no. 3, pp. 2793–2800.

[21] M. Tognon, H. A. T. Chavez, E. Gasparin, Q. Sable, D. Bicego, A. Mallet, M. Lany, G. Santi, B. Revaz, J. Cortes, and A. Franchi, "A truly-redundant aerial manipulator system with application to push-and-slide inspection in industrial plants," vol. 4, no. 2, pp. 1846–1851.

[22] C. Korpela, M. Orsag, and P. Oh, "Towards valve turning using a dual-arm aerial manipulator," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 3411–3416.

[23] H. Seo, S. Kim, and H. J. Kim, "Aerial grasping of cylindrical object using visual servoing based on stochastic model predictive control," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE.

[24] J. Fishman and L. Carlone, "Control and trajectory optimization for soft aerial manipulation," version Number: 3.

[25] C. Ding, L. Lu, C. Wang, and C. Ding, "Design, sensing, and control of a novel UAV platform for aerial drilling and screwing," vol. 6, no. 2, pp. 3176–3183.

[26] X. Tao and S. Y. Ko, "Tilt-x: Development of a pitch-axis tiltrotor quadcopter for maximizing horizontal pulling force and yaw moment," vol. 14, no. 14, p. 6181.

[27] K. Bodie, M. Brunner, M. Pantic, S. Walser, P. Pfändler, U. Angst, R. Siegwart, and J. Nieto, "An omnidirectional aerial manipulation platform for contact-based inspection," publisher: arXiv Version Number: 2.

[28] E. Steiner, "Object tracker GitHub repository: eliassteiner1/hole_tracker (www.github.com/eliassteiner1/hole_tracker)," original-date: 2025-02-02T17:26:43Z.

[29] Ultralytics YOLOv8 (https://docs.ultralytics.com/models/yolov8/).

[30] What is the maximum number of classes YOLOv8 can learn? (https://github.com/ultralytics/ultralytics/issues/1130).

[31] Brief summary of the YOLOv8 model structure (https://github.com/ultralytics/ultralytics/issues/189).

[32] CVAT - leading data annotation platform (www.cvat.ai).

[33] Best practice for training YOLOv8 with very small datasets (https://github.com/ultralytics/ultralytics/issues/6201).

[34] NVIDIA TensorRT (developer.nvidia.com/tensorrt).

[35] ONNX - open neural network exchange (www.onnx.ai).

[36] S. Węglarczyk, "Kernel density estimation and its application," vol. 23, p. 00037.

[37] P. Corke, W. Jachimczyk, and R. Pillat, *Robotics, Vision and Control: Fundamental Algorithms in MATLAB®*, ser. Springer Tracts in Advanced Robotics. Springer International Publishing, vol. 147.

[38] Onshape - product development platform (www.onshape.com).

[39] J. Kannala and S. Brandt, "A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses," vol. 28, no. 8, pp. 1335–1340.

A Supplementary Material

A.1 Blob Detector Parameter Tuning

Tuning the Opencv blob detector involves setting the parameters listed in Table A.1. The following strategy was used to determine a working set of parameters.

First, only `minRepeatability` and `minDistBetweenBlobs` were set to the standard values recommended by OpenCV. The `Threshold` values and step size were set to cover a broad range of illumination (50 to 200) at a fairly granular step size (10). Filtering with all the other parameters was then temporarily disabled.
A few example images with detectable holes were selected. Since these are ideally perfect full circles, parameters close to `Circularity` = 1.00, `Convexity` = 1.00 and `InertiaRatio` = 1.00 are expected to work well. However, in reality, the holes are not perfect circles; therefore, the acceptable range of these parameters had to be expanded. To do this, each parameter was enabled one by one, and the range was adjusted, so that the targets in all the test images after filtering are just barely being detected. Through this process, the true targets are still being detected when all filters are enabled at the end, while as many outliers as possible are rejected. Finally, two images where the target is both as small and as large as can be realistically expected were used to set the area filter parameters.

Parameter	Value	Description
`filterByArea`	`min: 12` `max: 500`	Used to filter blobs by their area in pixels.
`filterByCircularity`	`min: 0.70` `max: 1.00`	Filters blobs by how close they are to a perfect circle which is a value of 1.00 (square: 0.785, line: 0.00).
`filterByConvexity`	`min: 0.75` `max: 1.00`	Filters blobs by their ratio of the convex hull to the true area (0.00 - 1.00).
`filterByInertiaRatio`	`min: 0.40` `max: 1.00`	Filters blobs by their elongation (square or circle: 1.00, line: 0.00).

`Threshold`	`min: 50`
	`max: 200`
	`stp: 10`
		Thresholding values for all the binarized images that are generated. Blobs are extracted from all images.

`minRepeatability`	`val: 2`	Controls how many times the same blob has to be detected on multiple binarized images.
`minDistBetweenBlobs`	`val: 10`	Minimum distance between two blobs in pixels.

Table A.1: *A list of all blob tracker parameter values that were used for the evaluation experiments proved to be effective.*

A.2 YOLO Detector Details

A.2.1 Training Parameters

The YOLOv8 model was finetuned using the Ultralytics Python module, which offers a comfortable training API. All training parameters can be configured via an external YAML file. In addition to standard settings such as the path of the dataset, the batch size, and the number of training epochs, the API offers useful online augmentation. This technique applies randomized procedural image modifications during training to improve generalization. The following augmentation parameters were used to train the YOLO Detector module.

Parameter	Value	Description
`dropout`	`0.1`	Standard dropout probability during training.
`batch`	`20`	Training batch size (images per iteration).
`epochs`	`800`	The total number of epochs for finetuning.
`augment`	`True`	Enable online augmentation pipeline.
`close_mosaic`	`20`	Disable the mosaic augmentation in last 20 epochs. Can help to stabilize the training before completion.
`hsv_h`	`0.1`	Max random hue shift (fraction of 360°).
`hsv_s`	`0.7`	Max random saturation scaling (0 - 1).
`hsv_v`	`0.5`	Max random value (brightness) scaling (0 - 1).
`degrees`	`45`	Max random rotation angle (°).
`translate`	`0.2`	Maximum amount of random translations applied (fraction of image size).
`scale`	`0.7`	Max random scaling factor.
`shear`	`20`	Max random shear distortion angle ().
`perspective`	`0.0002`	Random perspective warp strength.
`flipud`	`0.5`	Vertical flip probability.
`fliplr`	`0.5`	Horizontal flip probability.
`bgr`	`0.0`	Random channel swap probability.
`mosaic`	`0.9`	Mosaic augmentation probability (randomly generates a new image by stitching together multiple others).
`erasing`	`0.3`	Maximum fraction of the image that is randomly erased.

Table A.2: *All the training paramters that were used to finetune the YOLO detector by using the Ultralytics API*

A.2.2 Pre- and Postprocessing for the Compiled Model

As discussed in Section 3.1.1, by compiling the YOLO model to run it natively on the Nvidia Jetson Orin NX, all pre- and postprocessing of the model in- and outputs was lost. Normally this is handled by the Ultralytics Python module. However, the compiled model is only able to perfom the pure network operations and only deals with raw tensor in- and outputs. Therefore, the following processing steps had to be emulated:

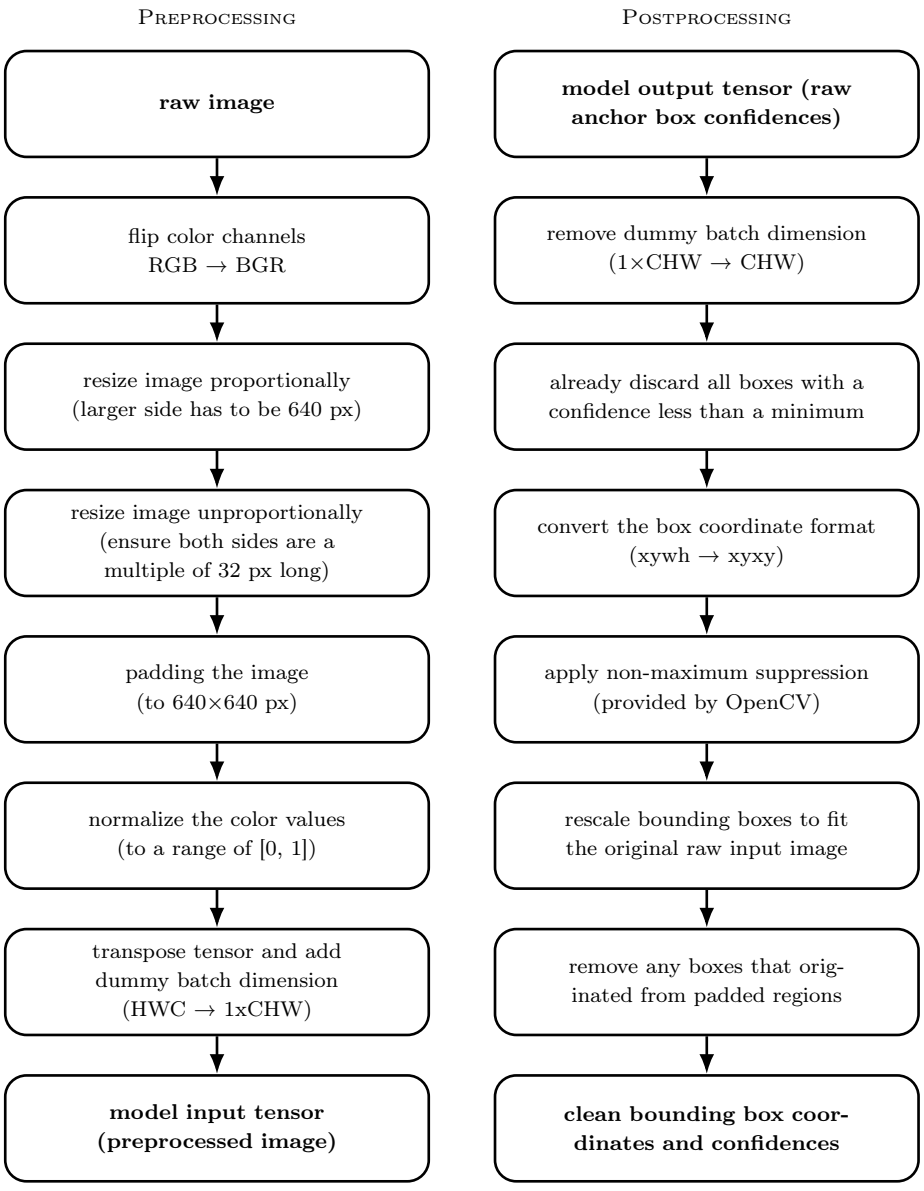


Figure A.1: *A summary over the emulated pre- and postprocessing steps that are normally handled by the Ultralytics Python module.*

Note that the source code for these reimplemented pre- and postprocessing routines can be found in `source/utils/multi_framwork_yolo.py`.

A.3 Depth Fusion

A.3.1 Normal Estimation Extrapolation Formula

For the normal estimation depth approach, first, the only information available is a normal vector and a pivot point (output of processed ToF point cloud). These essentially describe a plane in 3D space and can be expressed with respect to the camera coordinate frame. All points in the image are now assumed to originate from this plane - an acceptable approximation if the camera is mostly looking at a flat wall. Therefore, determining the depth of any image point only requires one to find the location where the corresponding point ray intersects the depth plane. Considering the full image formation formula is the first step in finding a closed-form solution to this problem [37].

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \overbrace{\begin{bmatrix} \hat{f}_x & 0 & c_x & 0 \\ 0 & \hat{f}_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}^{\text{Intrinsics}} \overbrace{\begin{bmatrix} \frac{\theta(x,y)}{r(x,y)} & 0 & 0 & 0 \\ 0 & \frac{\theta(x,y)}{r(x,y)} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}^{\text{Kannala-Brandt Distortion}} \overbrace{\begin{bmatrix} \frac{1}{Z} & 0 & 0 & 0 \\ 0 & \frac{1}{Z} & 0 & 0 \\ 0 & 0 & \frac{1}{Z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}^{\text{Projection}} \overbrace{\left[\begin{array}{c|c} \mathbf{R} & \mathbf{t} \\ \hline \mathbf{0} & 1 \end{array}\right]}^{\text{Extrinsics}} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

The image formation chain formula can then be inverted. For any given image point, the corresponding 3D world point can be determined with the inverted formula. Inverting the Kannala-Brandt distortion is not possible through a closed-form solution; therefore, an iterative approach must be used [39].

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \overbrace{\left[\begin{array}{c|c} \mathbf{R}^\top & -\mathbf{R}^\top\mathbf{t} \\ \hline \mathbf{0} & 1 \end{array}\right]}^{\text{Inv. Extrinsics}} \overbrace{\begin{bmatrix} Z & 0 & 0 & 0 \\ 0 & Z & 0 & 0 \\ 0 & 0 & Z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}^{\text{Unproject}} \overbrace{\begin{bmatrix} \mathcal{D}_{\circlearrowleft}^{-1} & 0 & 0 \\ & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}^{\text{Undistortion}} \overbrace{\begin{bmatrix} \frac{1}{\hat{f}_x} & 0 & -\frac{c_x}{\hat{f}_x} & 0 \\ 0 & \frac{1}{\hat{f}_y} & -\frac{c_y}{\hat{f}_y} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}^{\text{Inv. Intrinsics}} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Some assumptions and simplifications can now be made. The 3D world point is assumed to always originate from the wall plane and is expressed in wall coordinates (the x and y axes lie in the plane and the z axis along the normal direction). Therefore, its coordinates can be generically expressed as: $\begin{bmatrix} ? & ? & 0 & 1 \end{bmatrix}$. The extrinsics then have to be the transformation from the wall frame to the camera frame. Finally, the inverse intrinsics and the iterative undistortion can be applied to any image point right away. Therefore, on the right-hand side, the coordinates for a point in the undistorted normalized image plane can be written.

$$\begin{bmatrix} ? \\ ? \\ 0 \\ 1 \end{bmatrix}_{\text{wall}} = \overbrace{\begin{bmatrix} T_{11}^{-1} & T_{12}^{-1} & T_{13}^{-1} & T_{14}^{-1} \\ T_{21}^{-1} & T_{22}^{-1} & T_{23}^{-1} & T_{24}^{-1} \\ T_{31}^{-1} & T_{32}^{-1} & T_{33}^{-1} & T_{34}^{-1} \\ T_{41}^{-1} & T_{42}^{-1} & T_{43}^{-1} & T_{44}^{-1} \end{bmatrix}}^{\text{Inv. Extrinsics}} \overbrace{\begin{bmatrix} Z & 0 & 0 & 0 \\ 0 & Z & 0 & 0 \\ 0 & 0 & Z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}^{\text{Unproject}} \begin{bmatrix} P_x \\ P_y \\ 1 \\ 1 \end{bmatrix}_{\substack{\text{norm.} \\ \text{image} \\ \text{plane}}}$$

The problem now, of course, is that the depth value is the unknown coordinate Z. Since the inverse extrinsics can be deduced from the normal vector and the pivot

point, an equation can be formulated. Only the third row is considered and can be solved for the depth (Z).

$$0 = T_{31}^{-1} Z P_x + T_{32}^{-1} Z P_y + T_{33}^{-1} Z \cdot 1 + T_{34}^{-1} \cdot 1$$

$$Z = -\frac{T_{34}^{-1} \cdot 1}{T_{31}^{-1} P_x + T_{32}^{-1} P_y + T_{33}^{-1} \cdot 1}$$

In a last step, the terms from the inverse extrinsics can be simplified and vectorized. Considering the formula for inverting a generic homogeneous transformation matrix $T^{-1} = \begin{bmatrix} \mathbf{R}^\top & -\mathbf{R}^\top \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}$, we can see that all the T_{ij}^{-1} terms in the denominator are just the third row of $\mathbf{R}^\top$. The numerator term T_{34}^{-1}, on the other hand, is the last vector element of the expression $-\mathbf{R}^\top \mathbf{t}$, which itself is equal to the product of the third row of $\mathbf{R}^\top$ with $\mathbf{t}$.
The rotation matrix part is partially defined by our plane normal: the last row and column are both equal to the normal vector (the other axes are actually undefined, as normal estimation only returns one axis). $\mathbf{t}$ is exactly the pivot point, as it is responsible for translation. Finally, the individual coordinates from our undistorted image point can be gathered in one vector. This yields the final expression for the depth Z:

$$Z = -\frac{\mathbf{n}_{\text{norm. estim}} \cdot \mathbf{t}_{\text{norm. estim}}}{\mathbf{n}_{\text{norm. estim}} \cdot \mathbf{p}_{\text{img. plane}}}$$

A.3.2 Full Sensor Fusion Details

The main purpose of this sensor fusion is to obtain depth values for any pixels in the RGB camera image. Depth information is sourced from a ToF camera. If these two cameras were physically in the exact same location, fusion would be trivial: the depth and RGB image would be in the same image plane and corresponding values could just be looked up. However, in our case, the sensors are mounted in different locations, and therefore some processing is necessary to transfer the depth values to the RGB image plane. A lot of effort was spent on optimizing this pipeline and enabling it to run at up to 10 Hz on the platform, mainly through heavy use of vectorized numpy methods. The whole process is summarized in Figure A.2 and the implemented code can be found in the file `scripts/node_depthmap.py`.

It is important to note that the ToF camera output is a structured 3D point cloud. This means that it can easily be reshaped to also form a monochrome depth image when only the z values of all points are considered.

Determine ToF Camera Intrinsics Initially (only once) the intrinsic mapping (H) of the ToF camera has to be determined. It is used in the main point cloud processing pipeline to speed up the first step significantly. Since a depth camera outputs a full 3D point cloud, the "calibration" is straightforward and does not require recording special calibration images.

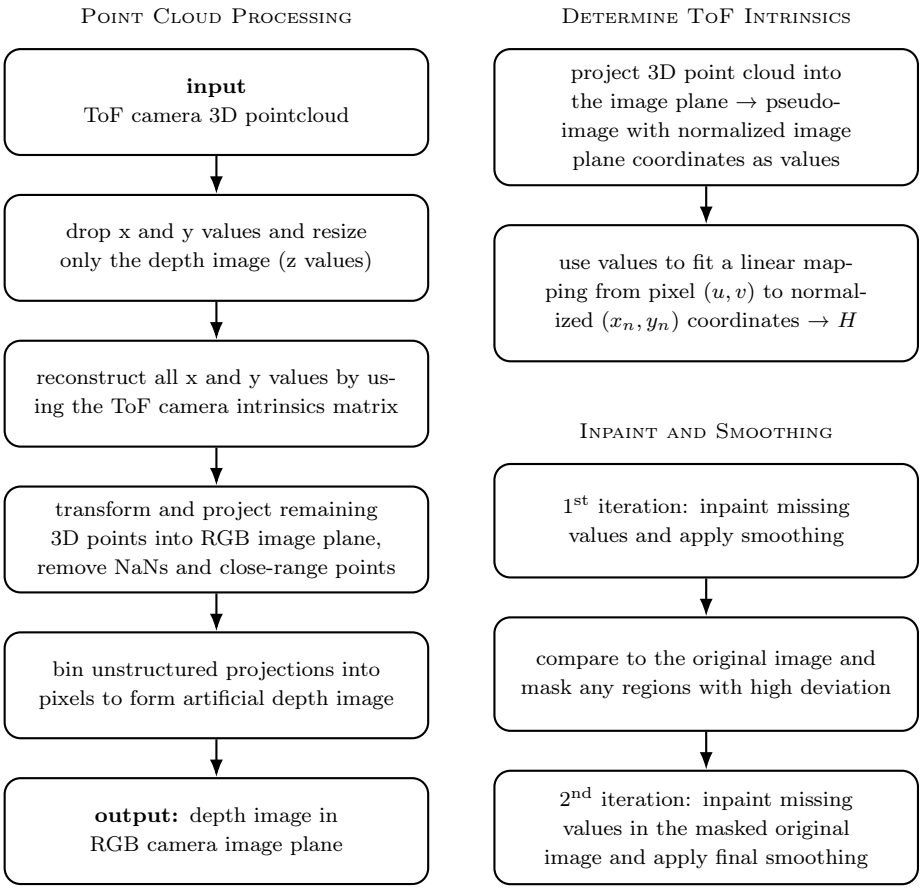


Figure A.2: *An overview of the three separate processes that are used to generate an artificial depth image in the RGB image plane.*

The x and y coordinates of all 3D points from one frame are simply normalized by their z coordinates (pinhole projection). This results in a pseudo-image matrix, where the indices are pixel coordinates, and the two values for each pixel are the corresponding x_n and y_n normalized image plane coordinates. As the camera intrinsics matrix is nothing more than a linear mapping between these two coordinate types, the pseudo-image provides all the necessary datapoints to determine H. A linear fit on both axes is used to determine the parameters $\hat{f}$ and c of the intrinsic matrix H.

Point Cloud Processing Pipeline As a first step in processing the raw ToF point cloud into a depth image in the RGB image plane, the number of points is drastically reduced. Treating 640 x 480 3D points during every subsequent operation would need unnecessary resources. Therefore, only the z values of all points are considered, and this monochrome depth image is resized to about a tenth of the resolution.
Then, using a lookup table of the ToF camera intrinsics, the x and y values for the remaining depth values can be reconstructed. This results in a downsampled version of the original point cloud. These points are then homogeneously transformed and projected into the RGB image plane. Any NaN values are removed and z values

that are below a certain threshold are cut off. The projection leaves the points in an unordered state and binning is necessary to recover a structured image.

Inpainting and Smoothing the New Depth Image The newly rasterized artificial depth image still contains many empty regions due to the measurement of NaN values. In order to fill any gaps, an iterative approach in two steps is used. First, missing values in the image are interpolated using the OpenCV inpainting method. Afterwards, smoothing is applied. The smoothed image can then be compared to the original one to determine regions where the values have significantly changed - this indicates noise or outlier points. These regions are then masked and removed from the original image. The same inpainting and smoothing operations are applied a second time to the masked image to produce a final depth image without considering noisy regions.

FIGURE A.3: *From left to right: the raw ToF point cloud with its dual depth image. The new artifical depth image and the inpainted and smoothed version.*

A.4 Estimator Initialization Details

Kernel Density Estimation is a technique for estimating the underlying probability distribution of a pool of discrete sample points. It is especially useful for multimodal data that cannot simply be approximated by a standard Gaussian or other common probability distribution functions. This is done by placing a kernel around each datapoint to give some "density" to the close neighborhood of the point. All kernel functions can then be summed up to determine a local density at an arbitrary location. By normalizing the summed density, a probability density function is obtained [36].

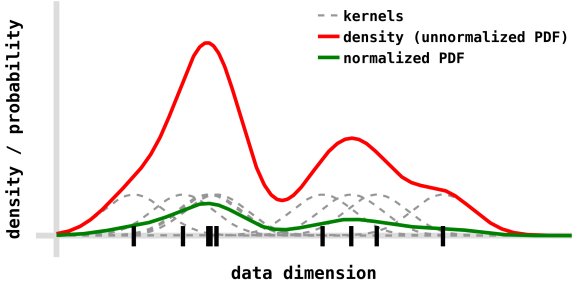


FIGURE A.4: *1D Kernel Density Estimation example*

The two main parameters that can be tuned are a "bandwidth" and the type of kernel function used around each datapoint. The kernel type was found to have no significant influence on the application of this thesis. Using other functions than a Gaussian mainly provided a computational benefit. However, bandwidth is important to set correctly, as it directly controls the width of the kernel. It is responsible for how much smoothing is applied to the data. A kernel size on the order of magnitude of the entire data range will result in oversmoothing and yields just a unimodal PDF. On the other hand, a kernel size that is about as large as the average distance between two datapoints will result in a high variance PDF.

For this thesis, it was found that setting the bandwidth to about the average expected cluster size (of the detection points) yields useful density values. From recorded data, it could be determined that detection points from the same target - aligned via drone odometry data - tend to spread out over a range of about 10 cm on average due to noise. Therefore, a bandwidth of 0.1 was used.
This parameter could be changed for a future task. It can be set to control the approximate size of a group of points, that is identified as a single connected cluster.

A.5 Visual Servoing Policy Details

For the visual servoing policy, it is necessary to relate accelerations from the image pixel space to accelerations of the camera (and drone) body. As a first step, the image Jacobian can be considered, which converts a camera twist (velocities) to image pixel velocities [37].

$$\dot{\mathbf{x}} = \mathbf{J}\dot{\boldsymbol{\nu}}$$

$$\begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix}_{\substack{\text{norm.}\\ \text{img.}\\ \text{plane}}} = \begin{bmatrix} xy & -(1+x^2) & y & -\frac{1}{Z} & 0 & \frac{x}{Z} \\ 1+y^2 & -xy & -x & 0 & -\frac{1}{Z} & \frac{y}{Z} \end{bmatrix} \begin{bmatrix}\boldsymbol{\omega}\\ \mathbf{v}\end{bmatrix}_{\text{camera}} \quad \text{where } x = \frac{X}{Z} \text{ and } y = \frac{Y}{Z}$$

The first equation can be differentiated with respect to time to obtain a relationship for accelerations. This yields a second mixed derivative term. However, since the Jacobian is usually only subject to slow change, this additional term was neglected.

$$\ddot{\mathbf{x}} = \mathbf{J}\ddot{\boldsymbol{\nu}} + \dot{\mathbf{J}}\dot{\boldsymbol{\nu}} \approx \mathbf{J}\ddot{\boldsymbol{\nu}}$$

Because the control law is defined in image space, and the corresponding camera twist must be computed, the inverse relationship is required to determine the necessary motion commands. However, the image Jacobian is not directly invertible; therefore, the Moore-Penrose pseudoinverse was used.

$$\ddot{\boldsymbol{\nu}} = \mathbf{J}^{+}\ddot{\mathbf{x}}, \quad \text{where} \quad \mathbf{J}^{+} = \mathbf{J}^{\top}(\mathbf{J}\mathbf{J}^{\top})^{-1}$$