SESAME

SECURE AND SAFE MULTI-ROBOT SYSTEMS

Project Number 101017258

D2.3 Collaborative Sensor Fusion

Version 1.0
5 September 2022
Final

Public Distribution

University of Luxembourg

Project Partners: **Aero41, ATB, AVL, Bonn-Rhein-Sieg University, Cyprus Civil Defence, Domaine Kox, FORTH, Fraunhofer IESE, KIOS, KUKA Assembly & Test, Locomotec, Luxsense, The Open Group, Technology Transfer Systems, University of Hull, University of Luxembourg, University of York**

Project Partner Contact Information

Aero41 Frédéric Hemmeler Chemin de Mornex 3 1003 Lausanne Switzerland E-mail: frederic.hemmeler@aero41.ch	**ATB** Sebastian Scholze Wiener Strasse 1 28359 Bremen Germany E-mail: scholze@atb-bremen.de
AVL Martin Weinzerl Hans-List-Platz 1 8020 Graz Austria E-mail: martin.weinzerl@avl.com	**Bonn-Rhein-Sieg University** Nico Hochgeschwender Grantham-Allee 20 53757 Sankt Augustin Germany E-mail: nico.hochgeschwender@h-brs.de
Cyprus Civil Defence Eftychia Stokkou Cyprus Ministry of Interior 1453 Lefkosia Cyprus E-mail: estokkou@cd.moi.gov.cy	**Domaine Kox** Corinne Kox 6 Rue des Prés 5561 Remich Luxembourg E-mail: corinne@domainekox.lu
FORTH Sotiris Ioannidis N Plastira Str 100 70013 Heraklion Greece E-mail: sotiris@ics.forth.gr	**Fraunhofer IESE** Daniel Schneider Fraunhofer-Platz 1 67663 Kaiserslautern Germany E-mail: daniel.schneider@iese.fraunhofer.de
KIOS Maria Michael 1 Panepistimiou Avenue 2109 Aglatzia, Nicosia Cyprus E-mail: mmichael@ucy.ac.cy	**KUKA Assembly & Test** Michael Laackmann Uhthoffstrasse 1 28757 Bremen Germany E-mail: michael.laackmann@kuka.com
Locomotec Sebastian Blumenthal Bergiusstrasse 15 86199 Augsburg Germany E-mail: blumenthal@locomotec.com	**Luxsense** Gilles Rock 85-87 Parc d'Activités 8303 Luxembourg Luxembourg E-mail: gilles.rock@luxsense.lu
The Open Group Scott Hansen Rond Point Schuman 6, 5th Floor 1040 Brussels Belgium E-mail: s.hansen@opengroup.org	**Technology Transfer Systems** Paolo Pedrazzoli Via Francesco d'Ovidio, 3 20131 Milano Italy E-mail: pedrazzoli@ttsnetwork.com
University of Hull Yiannis Papadopoulos Cottingham Road Hull HU6 7TQ United Kingdom E-mail: y.i.papadopoulos@hull.ac.uk	**University of Luxembourg** Miguel Olivares Mendez 2 Avenue de l'Universite 4365 Esch-sur-Alzette Luxembourg E-mail: miguel.olivaresmendez@uni.lu
University of York Simos Gerasimou & Nicholas Matragkas Deramore Lane York YO10 5GH United Kingdom E-mail: simos.gerasimou@york.ac.uk nicholas.matragkas@york.ac.uk	

Document Control

Version	Status	Date
0.1	Document outline	3 March 2022
0.2	Initial draft	14 March 2022
0.3	First draft	28 March 2022
0.4	Internal reviews	11 April 2022
0.5	Internal reviews updates	25 April 2022
0.6	Internal reviews	16 May 2022
0.7	Internal reviews updates	24 June 2022
0.8	Final version for partner reviews	31 August 2022
1.0	Final updates and QA review	5 September 2022
1.1	Revised version	7 March 2023

Table of Contents

List of Figures

List of Tables

Acronyms

6DoF	Six Degrees of Freedom.
AeroLab	Aerial Robotics Lab.
APM	Autonomous Pest Management.
ATE	Absolute Trajectory Error.
CNN	Convolutional Neural Network.
COTS	Commercially available Off-The-Shelf components.
CPU	Central Processing Unit.
CUDA	Compute Unified Architecture.
EKF	Extended Kalman Filter.
FCN	Fully Convolutional Networks.
GNSS	Global Navigation Satellite System.
GPS	Global Positioning System.
GPU	Graphics Processing Unit.
HDMI	High-Definition Multimedia Interface.
IMU	Inertial Measurement Unit.
IR	Infrared Radiation.
KF	Kalman Filter.
LiDAR	Light Detection And Ranging.
LS	Least Squares.
LU	University of Luxembourg.
mAP	Mean Average Precision.
MAV	Micro Aerial Vehicle.
MIT	Massachusetts Institute of Technology.
ML	Machine Learning.
MRS	Multi-Robot System.
MSCKF	Multi-State Constraint Kalman Filter.
OS	Operating System.
RADAR	RAdio Detection And Ranging.
RAM	Random Access Memory.
RCNN	Region Based Convolutional Neural Networks.
RGB	Red, Green and Blue.
RGB-D	Red, Green, Blue and Depth.
ROS	Robot Operating System.

RSCNN	Residual Shuffling Convolutional Neural Network.
RSRCNN	Road Structure Refined Convolutional Neural Network.
SDK	Software Development Kit.
SESAME	Secure and Safe Multi-Robot Systems.
SSD	Single-Shot Detector.
SWLS	Sliding Window Least Square.
TOPS	Tera Operations Per Second.
UAV	Unmanned Aerial Vehicle.
UKF	Unscented Kalman Filter.
USB	Universal Serial Bus.
UWB	Ultra Wide Band.
VIO	Visual Inertial Odometry.
VRAM	Video Random Access Memory.
WP	Work Package.
YOLO	You Only Look Once.

Executive Summary

This deliverable reports on the perception and collaborative sensor fusion methods and algorithms developed in Task 2.3 as part of the Secure and Safe Multi-Robot Systems (SESAME) project. The primary objective is to develop perception and collaborative sensor fusion mechanisms that allow robots to comprehend their environment accurately. We consider vision-based sensors to develop object detection, object tracking, and semantic segmentation algorithms for scene understanding. State-of-the-art deep learning techniques are extended for the tasks required in the work package 2. Further sections of the report describe the methods and algorithms used in Task 2.3 of the SESAME project.

1 Introduction

This is the third deliverable of the Work Package (WP) 2, "D2.3: Collaborative Sensor Fusion" of the SESAME project. The primary objective of this report is to describe the current state of development of the perception and the collaborative sensor fusion modules.

This document details the choices behind the sensors and computational capabilities of the different aerial robotic platforms. It also describes the interfaces between the different modules. Additionally, it provides in-depth details of the algorithms behind the perception and collaborative sensor fusion modules. And finally, it provides some early results regarding the performance of the different algorithms.

This report lays the groundwork to base the work and research of task 2.4 (WP2). The system development process follows an engineering systems design approach (SESAME Methodology). It starts from the specifications and functionalities identified in the previous Deliverable D2.1 “Specification of MRS Capabilities”, submitted in Month 12 of the project (M12), and it derives the development concepts in order to fulfill the goals of task 2.3 (WP2).

1.1 Background

Perception is one of the key capabilities enabling robots to sense their environment, understand it, and make decisions. In robotics, perception can come from a wide variety of sensors. Usually, we differentiate active sensors such as Light Detection And Rangings (LiDARs), Global Positioning Systems (GPSs), or ultrasonics range finder, from passive sensors such as magnetometers, accelerometers, or cameras. Since a robot relies on the sensors for perception, incomplete or unreliable information such as faulty sensors or limitations in their field of view significantly reduces the robot’s ability to complete its mission successfully. MRSs can mitigate this risk by fusing fragmented, or partially correct information, from robotic team members to collaboratively compose a more accurate representation of the environment, thus, supporting reliable reasoning and better decision-making. The SESAME project aims to develop novel collaborative perception capabilities relying on robust object detection and position estimation methods for robot localization and navigation.

This report focuses on task 2.3 of the Sesame project, Collaborative Perception and Sensor Fusion, which aims at developing a collective perception mechanism enabling safe and robust robot navigation. The main goal being for the robot to carry on with their mission in the presence of adverse environmental conditions, cyber-attacks or faults. In the context of this task, the scope of cyber-attacks or faults considered are things such as a sensor becoming impaired: losing the GPS positioning, or a camera. Similarly, the scope of the adverse environmental conditions are bounded to events that may perturb the state estimation of a system, such as entering GPS denied areas. Building on recent advances in deep learning, the perception algorithms proposed here estimate the position of objects in a scene. This can for instance enable an observer robot to visually estimate the position of the target robot. Sharing this estimation with the tracked robot could help it recover from a faulty GPS sensor for instance. Likewise, the observer robot can detect obstacles in the vicinity of the other robot and share their position with it. This could prove useful if the target robot’s vision sensors were failing or scrambled, preventing it from sensing obstacles. Hence, our project is articulated around five key elements, the detection of objects, the estimation of the position of these detected objects, the tracking of these detections, the fusion of the onboard sensors with the information provided by the observer robot, and finally the segmentation of images and the navigable space. In this deliverable, we assume that all the robots have access to a recently updated global map and that they can localize themselves inside this map. This assumption is reasonable for all usecases, hence we will be focusing mostly on the detection of obstacles. To evaluate our algorithms on task 2.3, we chose to focus on a complex scenario related to the UAV usecase (T8.7, T8.13), in which drones are collaborating to deal with pests in viticulture environments.

1.2 Document Purpose

The primary objective of this report is to describe the current state of development of the perception and the collaborative sensor fusion modules.

It also describes the interfaces between the different modules. Additionally, it provides in-depth details of the algorithms behind the perception and collaborative sensor fusion modules. And finally, it provides some early results regarding the performance of the different algorithms.

1.3 Relationship to other Deliverables

This deliverable is part of the core elements of the Sesame project. In Sesame, the collaborative perception sensor fusion component is a key enabler of the robot safety(WP4) and security(WP5). In the event of a safety issue, for instance, if the GPS sensor of one of the robots were to fail, the safety EDDIs inside this robot would trigger an alarm, allowing this robot to request assistance. Then, another robot from the fleet would be dispatched near its last known position. Upon arrival to this position, the fully functional robot can estimate the position of the faulty robot and share this information with it. Similar scenarios could be devised in case of security breach, for instance if the GPS was jammed.

This deliverable is also tied to WP3, Executable Scenarios and System Modeling, which provides the physical configuration of the robots to the sensor fusion, and the sensor configuration to the perception components.

Within WP2, this deliverable is tightly integrated with T2.4, Multi-Robot Monitoring Online Trajectory Generation, and T2.5 Multi-Robot Collaborative Rescue Mission. Both of these tasks leverage the tools developed in T2.3 to identify other robots and precisely locate them in space.

Overall, this deliverable exposes a set of interfaces providing various information to other robots or algorithms, as can be seen in figure 1.

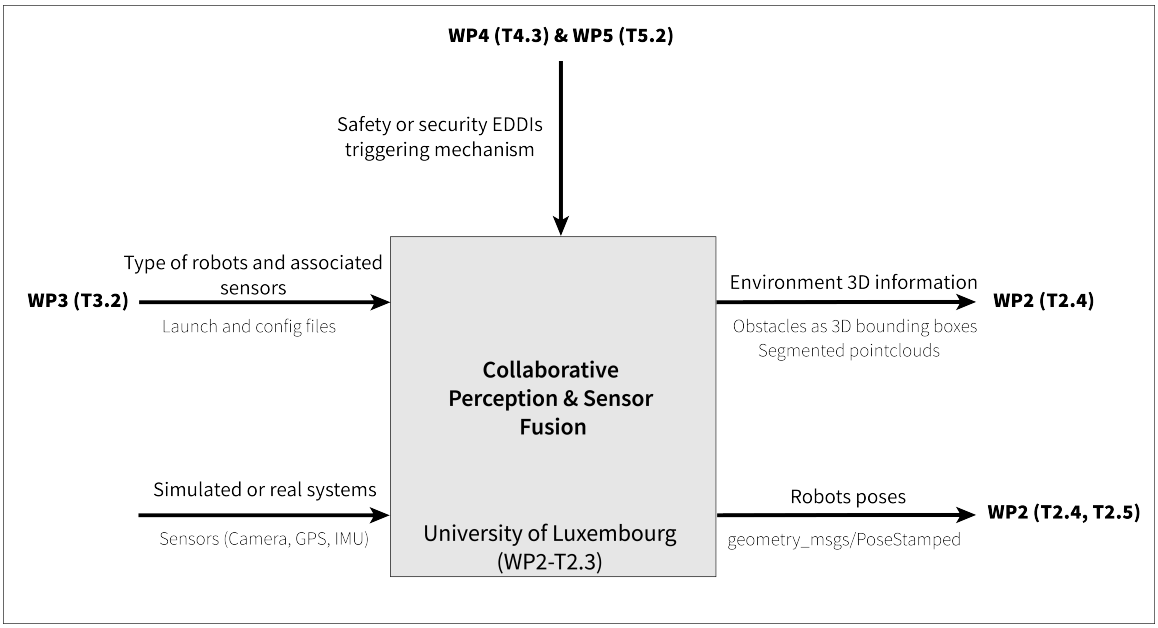

Figure 1: The external interfaces between the other modules are shown in terms of inputs and outputs. The rectangle box illustrates the focuses of this deliverable.

1.4 Document Structure

This Deliverable D2.3 consists of the following sections:

- Section 2 gives an overview of the related work in the field of MRS, with particular attention to perception and collaborative sensor fusion.
- Section 3 describes the perception and collaborative sensor fusion methods and algorithms for MRS as well as the segmentation.
- Section 4 explains the perception and collaborative sensor fusion development (software and hardware architecture) along with the experimental setup in detail.
- Section 5 describes the experimental overview and test results.
- Section 6 concludes this report.

2 Related Work

Collaborative MRS application scenarios require a rich, consistent, and accurate understanding of the environment and robot state. While single robots frequently suffer from sensor limitations (e.g., range, occlusion), robotic teams can combine multiple observations and share their results. Related work for perception and collaborative sensor fusion focuses on improving localization [1], planning [2], mapping [3], and object detection and tracking [4]. In addition, the efficient combination of both sensor limitations and movement commands of robots, known as active perception, has been studied for single robots [5] as well as for MRS [1], [2].

2.1 Perception

2.1.1 Drone Detection

In recent years, drone detection, and recognition have received much attention in various applications [6], [7], [8]. The concept of drone detection is the ability to detect the presence of a drone. Recognition is the ability to recognize the target drone. The problem of drone detection was investigated using the drone dataset and the proposed drone detection method. According to the studies [9], [10], the dataset for drone detection is obtained using active and passive sensors. In these studies related to the detection and recognition of drones using active sensors, the use of RAdio Detection And Ranging (RADAR) [9] and LiDAR [10] sensors is discussed. Problems with both of these sensors include high costs and limited integration into small drones. In addition, the use of thermal sensors results in lower accuracy due to low spatial resolution, and the use of acoustic sensors in drone detection and recognition has limitations such as high cost and limited onboard use. Therefore, due to the aforementioned limitations of using active sensors, visible imagery was used in the context of passive sensors that do not have the mentioned problems and do not have weight limitations when integrated into small drones. As previously mentioned, issues such as the unpredictable movements and speed of drones, the long-distance of the drone, its close resemblance to birds, its small size, the presence of hidden areas in the images, crowded backgrounds, the inability to separate the background, the problems with light in visible images, and different weather conditions challenge drone detection and recognition. For this reason, new methods of deep learning are used to solve the challenges based on studies.

In 2001, [11] detected moving objects using a set of visible images with fixed background and edge tracker methods. The object is then detected by finding the edge difference in successive images [11]. In 2011, [12] in a study called vision-based air collision detection system, detected drones using morphological filters to prevent airborne collisions. In 2016, [13] detected drones using background subtraction and image-based methods. Moreover, [14] proposed a new drone detection method by mounting cameras on a large variety of drones. In this work, the drone was detected by computing background motion with a perspective transformation model and detecting moving objects by foreground spatio-temporal features [14]. In 2017, [15] detected the drone using visible images and image sensors. In this study, the drone is detected using a saliency map, and it is localized using a Kalman filter [15]. In these studies, the traditional method of background subtraction has been used to detect drones, which do not have the appropriate accuracy and speed compared to modern methods. Later, researchers detected drones in a set of visible images using artificial intelligence-based methods and using VGG16 [16], and YOLO v2 (YOLOv2) neural networks [17]. The limitation of these studies was the low accuracy in detecting drones, which was improved in later studies by improving the methods used. In 2018, [18] detected drones in video datasets by subtracting background images and classification methods based on deep learning networks. In this article, the Kalman filter is applied to moving objects for better detection [18]. In this study, the deep learning method used can improve the accuracy of diagnosis using visual information. In 2019, drone detection was performed using YOLO [19], Faster-RCNN [20], and Single-Shot Detector (SSD) [20] methods. Region Based Convolutional Neural Networks (RCNN) and SSD methods were used to detect drones in video datasets, with the RCNN method showing better accuracy. The use of the YOLOv3 deep learning network in this study has resulted in improved accuracy and precision of drone detection compared to other methods due to its lightweight architecture and appropriate depth. In 2020, drones

were detected using YOLOv2, YOLOv3 [21], tiny-YOLOv3, YOLOv4 [22], Fast-RCNN, and SSD networks and the results were compared [22]. The three models YOLOv4, YOLOv3, and SSD were compared, and, respectively, YOLOv4, YOLOv3, and SSD had the best accuracy.

In 2021, using a deep learning network, the challenges in drone detection were examined below in more detail. In this year, segmentation-based methods were used to detect drones in crowded backgrounds [23], and another study detected drones in real-time using the YOLOv3 network on Nvidia Jetson TX2 hardware [24]. The use of this method has provided good accuracy and speed and is capable of detecting drones of various sizes. Other methods used to detect drones include Faster-RCNN, SSD, YOLOv3, and DETR, whose performance was examined in a series of visible images [25]. All the methods used in this study performed well in detecting drones, but YOLOv3 provided the best precision. Researchers have also recently used YOLOv4 [26], a pruned YOLOv4 [27], RetinaNet, FCOS, and YOLOv3 network in video and image datasets to achieve high accuracy in drone detection. The use of YOLOv4 in the first study provided acceptable drone detection results compared to similar studies and had better accuracy. Furthermore, in the next study, the networks used had good accuracy but good performance in detecting small and fast drones. Therefore, the pruned YOLOv4 method gave better performance compared to these methods. In 2021, [28] identified several types of multirotors and a fixed-wing with their commercial models in video sequences. The diagnostic system in this work is associated with a warning algorithm that sounds when the drone is observed. In this work, the standard Cascade R-CNN (Regions with Convolutional Neural Networks) architecture, Faster R-CNN, YOLOv3, YOLOv4, and YOLOv5 network were used to identify drones vs. birds. The discussion on detection in a variety of backgrounds with additional data also needs to be extended [28]. Based on the results of the studies, the YOLOv5 deep learning network presents higher accuracy and speed in detecting and recognizing drones in visible imagery than conventional methods. Therefore, this method was used to detect the drones.

2.1.2 Semantic Segmentation

Semantic segmentation is different from object detection as it does not aim at predicting bounding boxes around the objects. It does not distinguish between different instances of the same object. In order to perform semantic segmentation, a higher level understanding of the image is required. The algorithm should figure out the objects present and also the pixels which correspond to the object. Semantic segmentation is one of the essential tasks for complete scene understanding. Semantic segmentation divides the image into regions and labels each region with a predefined class label. Identifying the layout of the scene provides information about the spatial distribution of the object and its relationship with the environment. A brief overview of semantic segmentation is presented below. A detailed review of image semantic segmentation can be found in [29], [30].

The major focus on semantic segmentation in the aerial environment is currently dedicated to drones. The most exploited Convolutional Neural Networks (CNNs) in this field are the utilized Road Structure Refined Convolutional Neural Network (RSRCNN) model for road extraction in aerial scenes [31], tree-like structured CNN feature extractor [32], multi-modal data [33] as an input to a Residual Shuffling Convolutional Neural Network (RSCNN) [34], and Fully Convolutional Networks (FCN) based model [34]. We use a PyTorch implementation of semantic segmentation models on the Massachusetts Institute of Technology (MIT) ADE20K scene parsing dataset for this work [35]. ADE20K is the largest open source dataset for semantic segmentation and scene parsing, released by MIT Computer Vision team [36], [37].

2.2 Collaborative Sensor Fusion

Existing sensor fusion methods can generally be divided into filter-based or optimization-based methods [38], both of which are widely used in robotic applications. A multi-sensor fusion algorithm based on the EKF approach is proposed in [39], and [40]. This framework supports any number of sensors and can efficiently handle relative time state constraints. In [41], a UKF based framework is proposed, which also supports different types of absolute state update and relative time state update. Compared with EKF, UKF does not

need to derive a Jacobian matrix. However, due to the sampling numbers of the state vector, the computation load of this method increases, and it also brings new challenges to state augmentation. The optimization-based state estimation algorithm is another optional framework [42], [43]. Although theoretically, the method based on optimization can obtain higher accuracy due to multi-step iterative optimization, due to the influence of various parameter settings and implementation, the method based on filtering can also achieve accuracy comparable to the optimization-based method [44], [45], [46].

The purpose of multi-robot cooperative state estimation is that the localization of a single robot can not only use its own sensors, but also fuse the sensors or localization results of other robots, received through the communication system. The MRS is considered as a coupled system. The sensor information of different robots is connected to a network structure through the communication system. This structure requires high efficiency of calculation and high timeliness of communication. In [47], the benefits of collaborative state estimation are summarized as:

1, The state estimation of a single robot is improved by the information sent from other robots. For example, low precision sensors on a single robot can benefit from high precision sensors on other robots.

2, Provide redundancy in case of sensor failure. For example, when a single robot performs Visual Inertial Odometry (VIO) localization, the camera or IMU suddenly fails, and its state estimation can be recovered by other robots through relative localization.

The computing architecture of collaborative state estimation can be divided into centralized and decentralized. The centralized architecture is a single server connecting with multiple clients. And all coupling calculations are completed in the server. This has high requirements for the security of the communication system and the server between clients. Once the server fails, all calculations will end. The decentralized architecture uses peer-to-peer communication to complete information fusion on a single robot. In recent years, many papers have used decentralized or distributed architecture to implement collaborative state estimation. In [48], [47] and [49] consider the general decentralized collaborative state estimation for multiple sensors. [50] and [51] are two systems focusing on cooperative VIO for multi robot. The back-end of these systems is based on Kalman filter. In addition, [52] proposed a back-end based on SWLS optimization to fuse VI-Sensor and Ultra Wide Band (UWB) measurements.

3 Methods

In this section, we will introduce the different methods that we designed to solve task 2.3. One of the primary objective of this project is to robustly estimate the pose of a robot using its own sensors, but also using position estimates shared by other robots in its vicinity. More precisely, we will demonstrate the applicability of our algorithms using two robots: a detector robot, whose goal is to estimate the position of a target robot. Using this position estimation and its own sensors, the target robot estimates its position robustly. Using the extra information provided by the detector robot, the target drone can still estimate its state even if one of its positioning sensors were to fail. To estimate the position of the target robot, we rely on camera images and a depth sensor of the detector robot. Using a neural-network specifically trained to detect robot in images, we first identify the target robot in the image. Then, using a depth sensor, we measure the distance to the target robot. Leveraging this distance information, we can then estimate the relative position of the target robot in the detector robot's frame. This position is then projected into a global frame and share with the target robot.

As detailed in the introduction, we apply these algorithms to drones, hence, some of the method subsections will provide some application specific details, yet they should translate well to other type of robots. Figure 2 provides an overview of this objective when applied to drones.

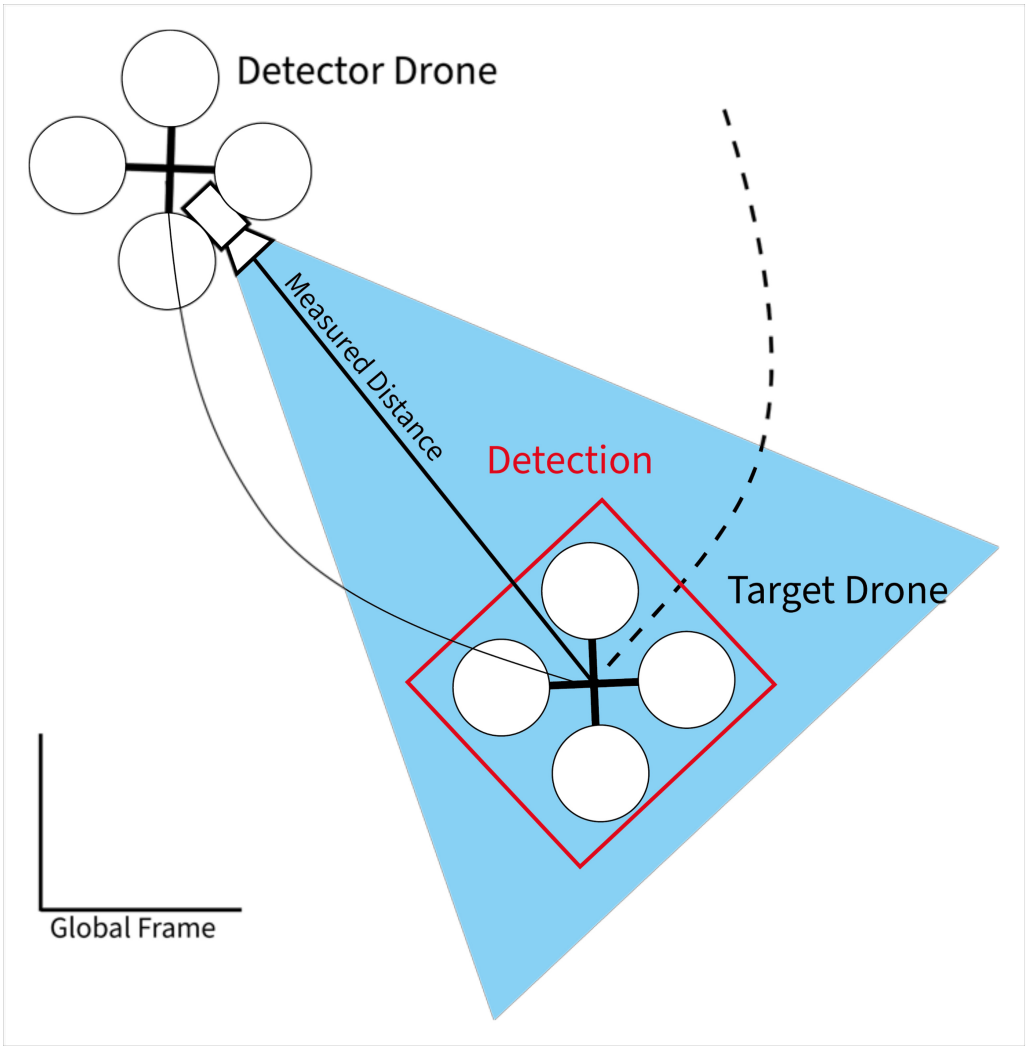

Figure 2: Overview of the sensor fusion task.

The second goal of this task is to provide scene understanding to the robots. To do so, we propose two different approaches. A classical approach based on scene segmentation. And an object detection based approach. In

the first one, the semantic segmentation, we train a neural-network to provide high level context to the robot. In this case, we build two different algorithms. One provides information regarding the occupancy of the space in front of the drone, i.e. it detects obstacles and free space in front of the drone. The other labels image region, in the case of the drone, we segment the pixels that belong to the ground or a wall for instance. On a rover, this information could be used to detect unsafe areas, such as avoiding stairs.

In the second one, we apply the same algorithms as the robot detection except, this time, we detect different obstacles that must be avoided. This has numerous advantages, first it comes at no extra computational cost, the inference time of the object detector does not change that much when the number of detected objects increases. Hence, adding to it the ability to detect humans, or other relevant obstacles is only increasing the cost of the tracking part which is computationally cheap compared to running a network, and can easily be threaded. Second, since it can be reduced to a 3D bounding box, this information is lightweight and can easily be shared across a fleet of robots. This is particularly true when compared to a map. Moreover, when done in a distributed fashion, the reconstruction of 3D maps is particularly heavy both in terms of compute and for the network. While the information provided by the object detection is not as rich as a 3D map, it is significantly lighter, making it better suited to our use-cases. Also, as mentioned in the introduction, we consider that an up-to-date global map already exists, and that the only changes that occurred are related to obstacles that can be identified using object detection.

Unlike the previous scenario example, we apply our obstacle detection algorithms to rovers, to show that our algorithms are not tied to drones and can easily be deployed on other systems. In fact, all the examples shown here are running on the same code when it comes to vision based state estimation.

3.1 Notations and Generalities

Before going through the different algorithms, let us take the time to define some of the notations that will be used in the rest of this document.

A vector from A to B is expressed as r_{AB}. The coordinates of this vector in the coordinate system C is expressed as ${}_Cr_{AB}$. In this report, we use quaternions to represent the rotation of rigid bodies, which is a non-singular expression. A detailed introduction to quaternions and their properties can be found in [53]. q_{BA} denote the attitude of a coordinate frame B with respect to frame A, with the corresponding rotation matrix $R\left(q_{BA}\right)$. The coordinate transformation is expressed as follows:

$$ {}_Br = R\left(q_{BA}\right) {}_Ar \tag{1} $$

According to quaternion algebra, we need to pay special attention to the addition of quaternion, $q + \delta$, and the subtraction between two quaternions $q_1 - q_2$, as they involve the operation between manifold and tangent space. The exponential mapping $\exp\left(\bullet\right)$ maps a vector in tangent space to a quaternion. The logarithmic mapping does the opposite.

$$ \begin{aligned} q + \delta &:= q \otimes \exp\left(\tfrac{\delta}{2}\right) \\ q_1 - q_2 &:= 2\log\left(q_2^{-1} \otimes q_1\right) \end{aligned} \tag{2} $$

Here, δ is a vector in tangent space, and $\otimes$ represents quaternion multiplication.

Formally, our problem scenario is defined as follows. Let us define the world coordinate system as W, and the IMU coordinate system as I. In our use case, we consider two Unmanned Aerial Vehicles (UAVs), as shown in the Figure 3. I_1 is the IMU frame of UAV1 and I_2 is the IMU frame of UAV2. We make the following assumption:

- The global localization of UAV1 is reliable, which means we can get UAV1's full pose.
- The IMU of UAV2 always works, but all other sensors of UAV2 used for global localization may fail. I.e. the global localization module of UAV2 may not be working all the time.
- UAV1 can provide the relative position between UAV1 and UAV2 through visual perception. This relative position can always be expressed as a global position.

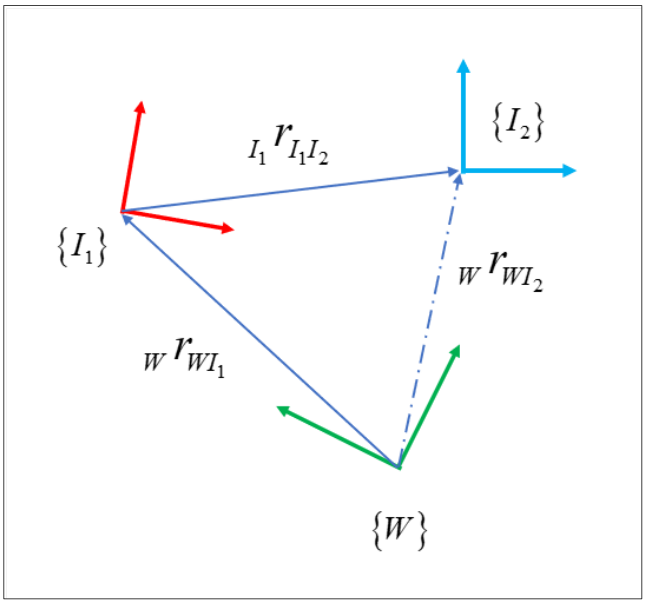

Figure 3: Coordinate Frames

3.2 Drone Detection

This module is used to detect drones within the field of view of the detector drone. The drone detection algorithm detects drones using images captured by the Red, Green, Blue and Depth (RGB-D) camera mounted on detector drones.

3.2.1 Drone Detection Training Dataset

To be able to detect drones, we train an object detection neural-network, i.e.YOLOv5 [54]. However, in order to train, we first need a dataset. Hence, we collected $1,339$ images from videos of drones flying inside the AeroLab of University of Luxembourg (LU). These images were annotated by manually drawing bounding boxes around the target drone. We only labeled the target drone to ensure it would detect the target drone and not others.

All images are labeled using RoboFlow[1], an image labeling tool. To optimize the labeling time, we only label one out of ten to thirty frames. If the drone has a large inter-frame movement, then we increase the labeling frequency, conversely, if the drone has low inter-frame movement we reduce the labeling frequency. To improve the robustness of our model against object motion, false positives, and motion blur, we include images with a wide variety of settings with intricate backgrounds (including chairs, fences, and other drones). Empirically, we found that without adding these random objects our detector would confuse the detector drone with bars, or truss. To further improve the robustness of our detector towards noise, and motion, we also add synthetic motion blur of various intensity, as well as typical data-augmentation, to the images. Adding these effects on the images robustifies our detector. We show some sample images of our dataset in Figure 4.

Among the $1,339$ images in the current dataset, we randomly select 90% for the training dataset and retain the remaining 10% as the testing dataset.

3.2.2 Drone Detection Algorithms

As alluded to earlier, our goal is to detect and track the target drone in the images captured by an RGB-D camera mounted on the detector drone. Using the bounding box of the target drone and the depth image, we

[1] https://roboflow.com/

Figure 4: Sample Trained Dataset Images.

then estimate the position of this target drone within the camera frame. We can then project the position into a global frame using the onboard Global Navigation Satellite System (GNSS) sensors, and then be supplied to the target drone. This drone being able to use this information within its own sensor fusion algorithms if one of its sensors were to fail. The first step to achieve our goal is the detection of the drone. To do so, we choose to rely on Deep-Neural-Network-based object detection models. More precisely, we used YOLOv5 as it offers a good trade-off between detection accuracy, and compute cost. In our case, the computational cost of the detection algorithm is critical. The object detection algorithm will be running onboard a medium size drone with limited compute capacities.

3.2.3 YOLOv5 Network Architecture

To perform the detection, we use the smallest version of YOLOv5, the S version. Unlike the other larger variants of YOLOv5, this smaller version has nine convolutional layers with 3×3 kernel layers and six pooling layers with 2×2 kernel layers. The final output of our network is a tensor of size $13 \times 13 \times 30$. These modifications make its memory footprint smaller, and significantly reduces its inference time. This makes it ideal for embedded applications. The drawback being a decrease in accuracy. However, the performance drop remains acceptable because we only seek to detect drones. Compared to the original algorithm, we only detect a single class, drones, instead of 80.

3.2.4 YOLOv5 Training and Inference

The YOLOv5 model is trained on a Google Cloud Server equipped with an Nvidia Tesla P100 Graphics Processing Unit (GPU). Because it's a small model, the training takes a short amount of time. Please note that the training can be carried on any recent GPUs with 2 or more 4GB. However, we recommend a GPU with 10 to 12GB of Video Random Access Memory (VRAM) as it allows training on larger batch-sizes, improving the overall performance of the drone detection algorithm.

To further improve the detector's performance, we resume the training using pre-trained weights. We use YOLO's default training weights as described in YOLOv5 [54]. The training and testing of neural networks are implemented in PyTorch [2], metrics, training, and validation are shown in Figure 5, 6, 7.

As for the inference of the network, we use TensorRT a common framework to execute quantize neural-network models. While we could use PyTorch directly, recent Machine Learning (ML) frameworks do not support Python 2.7. This is a problem as Ubuntu 18.04, the Xavier's Operating System (OS) comes with ROS Melodic, which only support Python 2.7. Hence, to make it work we would need to run a docker with Ubuntu 20.04, increasing the complexity of the system.

One of the main benefits of TensorRT is that it can be used inside C++ code, making it agnostic to the OS distribution, or ROS version. Furthermore, TensorRT optimizes the execution of a model on a specific GPU, improving the inference time.

Hence, we convert the PyTorch weights into an ONNX model, which we then convert into a TensorRT engine allowing us to optimize execution of the YOLOv5 S model for the Xavier. The model is then called inside a ROS Melodic node written in C++ and Compute Unified Architecture (CUDA).

[2] https://github.com/ultralytics/yolov5

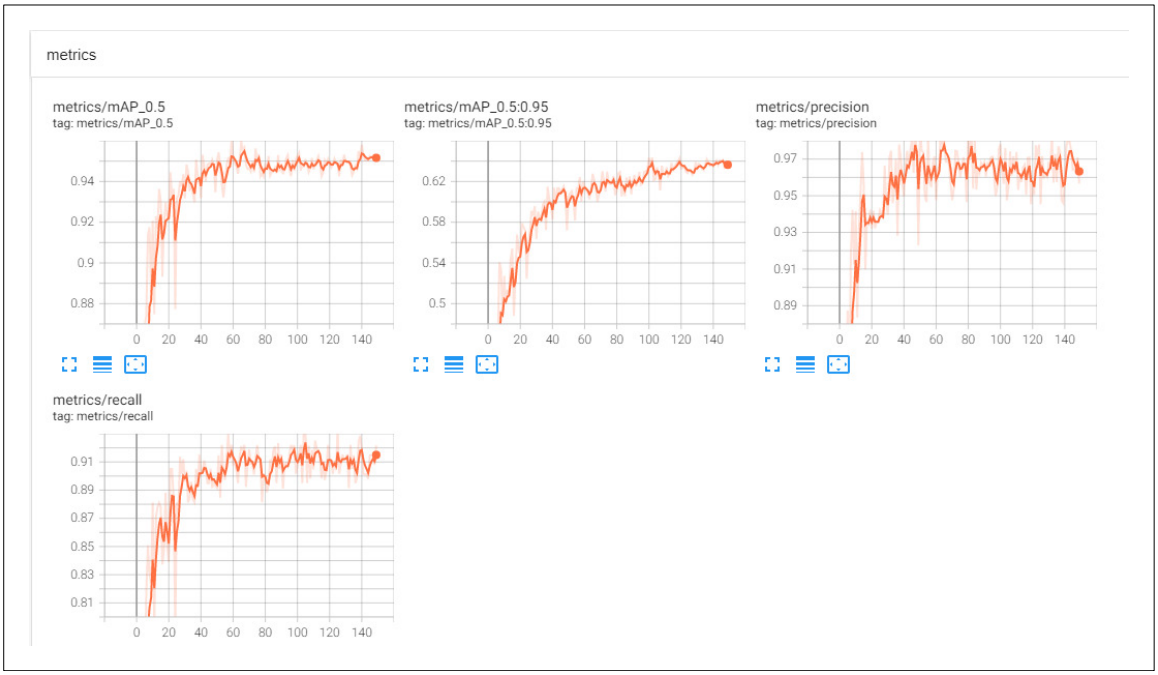

Figure 5: Metrics of Trained YOLOv5 model.

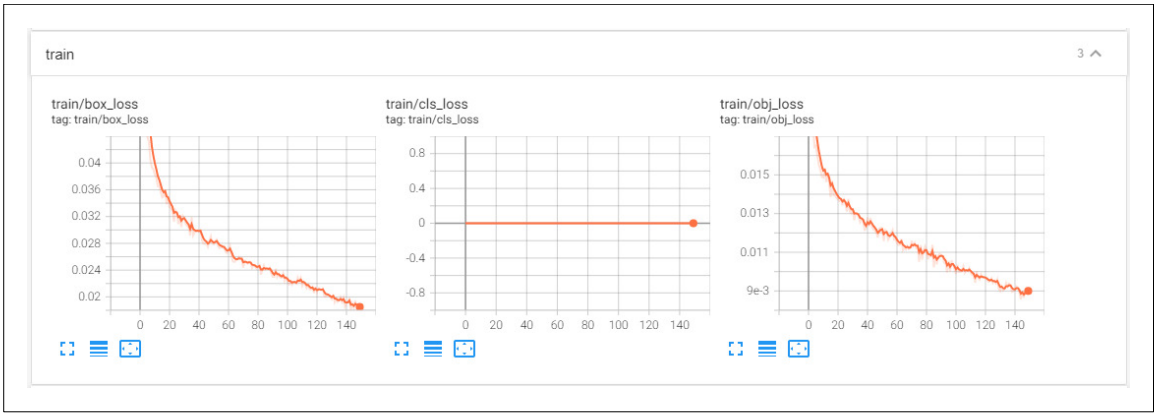

Figure 6: Loss and mAP for Training of YOLOv5 model.

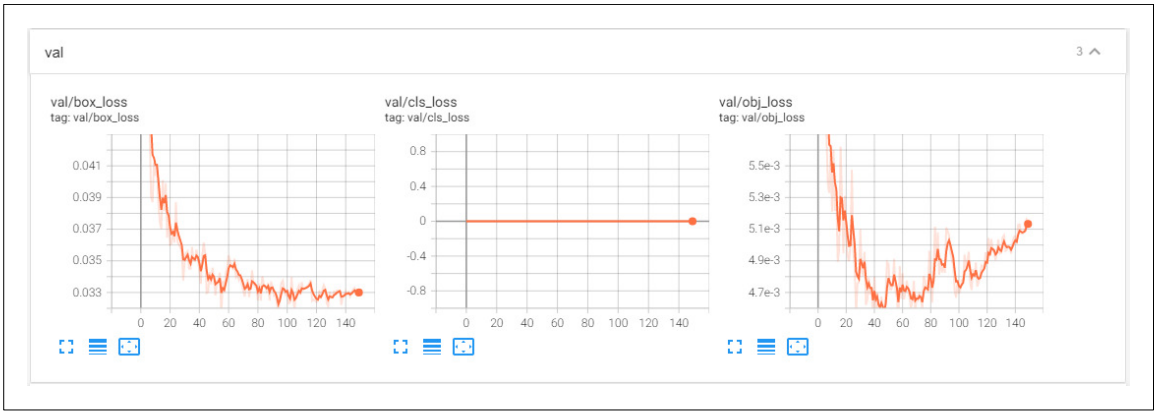

Figure 7: Loss and mAP for Validation of YOLOv5 model.

3.3 Position Estimation

Assuming the bounding box of the target robot is obtained, we denote (u, v) as the pixel coordinates of the box's center. The depth value Z is acquired from the depth camera. With these variables known, the relative position of the target robot can be calculated by following the pin-hole model, as in Equation (3).

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} Z \tag{3}$$

Even though, this simple model does not account for advanced lens distortion, like a plumb-blob model would, the pin-hole model position estimation is good enough. The plumb-blob model adds a lot of complexity, which increase the computational cost of the algorithm. Indeed, with the plumb-blob model, the transform is non-invertible. Hence, it requires gradient descent to find the inverse transform for each pixel. However, we do provide the mean to use a plumb-blob model if needed in our implementation of this algorithm.

One of the main challenge, when estimating the position of an object from images, is measuring the distance between this object and the camera. The bounding box is not always centered around the robot, which means that sometimes, taking the center coordinates can result in probing the space behind the robot, leading to erroneous measurements. This is particularly problematic with drones, as the space behind them is often far, resulting in extremely inaccurate measurements. Another issue comes from the relatively high level of noise present on the depth images from commercial RBG-D cameras, making the Z measurement fluctuate by a significant margin. To address both of these concerns, we measure the distance between every pixel within the bounding-box and the detector's robot camera using the pin-hole model. We store all these values inside a vector that we then sort from the smallest values to the largest. The first 5% are considered as outliers and rejected, the following 10% are averaged and used as the distance between the target robot and the camera. This distance is then reinjected inside the pin-hole model to compute the relative position of the target robot, as shown in Equation 4.

$$\begin{aligned} d &= \sqrt{X^2 + Y^2 + Z^2} \\ X &= Z * \frac{u - cx}{fx} \\ Y &= Z * \frac{v - cy}{fy} \\ Z &= \frac{d}{\sqrt{\frac{u-cx}{fx}^2 + \frac{v-cy}{fy}^2 + 1}} \end{aligned} \tag{4}$$

Leveraging this process, the measured distance is stable and the relative position improved. However, there is an offset due to the fact that we are averaging on 10% of the closest points. This offset needs to be calibrated for all the robots. On our drones, it is of around 15cm. Finally, since we know the global pose of the detector robot, we can then project the relative position of the detector robot into the global frame. In practice, in our implementation, we provide different mean of estimating this distance, so that the user can use the method that works best for its use-case.

3.4 Object Tracking

Now that we have a mean to detect the target robot, we need to track it. Indeed, when the detector misses the robot in an image, we still have to know where it is. Also, tracking the robot allows rejecting outliers,

or detecting multiple robots in parallel without confusing them. This problem of multi-target tracking is non-trivial, as this kind of tracker requires extensive tuning to work reliably.

In this work package, we choose to use a conservative tracker design. The propagation of the state is based on Kalman filters, while the association problem is solved using the Hungarian Matching algorithm. To track, we first need to define a state, in our case there are two possible ways to define the state. On the one hand, the drones can be tracked directly in the image frame, using pixel coordinates. On the other hand, the drones can be tracked in the global frame using the estimated xyz coordinates. Tracking in the global frame is simpler and provides significant advantages, however, it is not always possible. Hence, we provide the formulation of both problems.

3.4.1 Image Frame Tracking

When tracking objects directly from pixels, we use the center of the bounding boxes noted (u, v). We also want to keep track of the width and height of the bounding boxes (h, w), as it may help us differentiate between two objects. Furthermore, to be able to interpolate the position of the target robot in between two frames, we add to our state the velocity of the drone in pixel per second in the camera frame $(\dot{u}, \dot{v})$. Hence, we define our state X as in Equation 5:

$$X = \begin{bmatrix} u \\ u \\ \dot{u} \\ \dot{v} \\ h \\ w \end{bmatrix} \tag{5}$$

Equation 6 shows the dynamics of the tracked objects. We reduce the dynamics to a simple linear system, as we do not have any information on the commands or orientation of the system being tracked.

$$F = \begin{bmatrix} 1 & 0 & dt & 0 & 0 & 0 \\ 0 & 1 & 0 & dt & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

Equation 7 and 8 represent the observation vector Z and the associated H matrix, respectively.

With (u_{obs}, v_{obs}) the observed center of the bounding box, and (h_{obs}, w_{obs}) the height and width of the observed bounding box. The velocities are not measured, but derived within the Kalman filter.

$$Z = \begin{bmatrix} u_{obs} \\ v_{obs} \\ h_{obs} \\ w_{obs} \end{bmatrix} \tag{7}$$

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{8}$$

The filter is then updated and corrected following the standard formulas for linear Kalman filters. For the reader convenience, we recall the prediction and correction equations in Equation 9, and Equation 10 respectively. With Q, the process noise, a diagonal matrix where each coefficient on the diagonal correspond to the

uncertainty on the dynamics of each state of the system. R is the measurement noise, a diagonal matrix where each element on the diagonal correspond to the noise variance on each measured state.

$$\begin{aligned} X_{t|t-1} &= FXt-1|t-1 \\ P_{t|t-1} &= FPt-1|t-1F^T + Q \end{aligned} \tag{9}$$

$$\begin{aligned} Y_t &= Z_t - HX_{t|t-1} \\ S_t &= HP_{t|t-1}H^T + R \\ K_t &= P_{t|t-1}HS_t^{-1} \\ X_{t|t} &= X_{t|t-1} + K_tY_t \\ P_{t|t} &= (I - K_tH)P_{t|t-1} \end{aligned} \tag{10}$$

3.4.2 Global Frame Tracking

Instead of tracking in the image frame, one can track in the global frame. This can be particularly useful, as now, the only motion is the one of the tracked object, and not the addition of the tracking robot and the tracked robot. Our state X is defined as in Equation 11. Where x, y, z are the x, y, z coordinates in the global frame, $\dot{x}, \dot{y}, \dot{z}$ are the velocities in the global frame, and h, w are the width and height of the bounding box. We keep the bounding box's height and width, as it can be used to improve the matching.

$$X = \begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ h \\ w \end{bmatrix} \tag{11}$$

Equation 12 shows the dynamics of the tracked objects.

$$F = \begin{bmatrix} 1 & 0 & 0 & dt & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & dt & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & dt & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{12}$$

The observation vector Z and the associated H matrix are given in equation 13 and 14 respectively. With $(x_{obs}, y_{obs}, z_{obs})$ the observed global position of the object, and (h_{obs}, w_{obs}) the height and width of the observed bounding box. The velocities are not measured, but derived within the Kalman filter.

$$Z = \begin{bmatrix} x_{obs} \\ y_{obs} \\ z_{obs} \\ h_{obs} \\ w_{obs} \end{bmatrix} \tag{13}$$

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{14}$$

The filter is then updated and corrected following the formulas given in Equation 9, and Equation 10 respectively.

3.4.3 Association

The goal of the association part of the tracker is to match the currently tracked objects, with the newly observed objects. To do so, we use the propagation model of either filters (Equation 9). This propagates the previous states and gives us new states to match against the newly observed objects. In the rest of this section, we note the state of the i-th tracked object at time t as X_t^i, and the j-th newly observed object as O_t^j. To find the most optimal match, we compute the Euclidean distance for every possible combination of tracked/observed objects. Note that we could also use the IoU, as it is a combination of both the distance and the overlap between the bounding-boxes. To prevent the tracker from making undesirable matches, we add a threshold. If the distance is larger than a given threshold, then we replace this value with an arbitrarily large value, such as $1e6$. The distance function is given in 15.

$$d_{i,j} = \begin{cases} ||X_t^i - O_t^j||, & \text{if} d_{i,j} < \text{threshold} \\ 1e6, & \text{otherwise} \end{cases} \tag{15}$$

Once the distances are calculated, the matching is performed greedily using the Hungarian Algorithm. This method solves the assignment problem in polynomial time. Its goal is to minimize the total distance between all the pair of tracked/observed objects, to find the most optimal set of tracked/observed object pairs. In practice, this is solved using a bipartite graph [55].

Once the association is done, we evaluate if the match is correct. Sometimes, the assignment algorithm is producing incorrect matches. Hence, we verify that each match fulfills a set of conditions. In our case, we measure two things: the Euclidean distance between the matched pair, and the ratio between their bounding box area. The distance prevents from matching objects too far away, while the area ratio prevents from matching objects of different sizes. If a pair does not match these conditions, it is discarded. Further, the observation is considered as a new detection, and the tracked object is not matched with anything. If a pair matches these conditions, the observation is used to correct the state of the tracked object using Equation 10 of the Kalman filter.

3.5 Scene understanding Obstacle detection

3.5.1 Semantic Segmentation

Semantic segmentation refers to the process of associating each pixel in an image to a class label. For example, these labels could include a drone, person, car, etc., just to mention a few. To do so, we chose to rely on the PyTorch-based semantic segmentation model. More precisely, our model is made of a ResNet50dilated [56] for the encoder and a PPM_deepsup [56] for the decoder. This combination offers a good trade-off between pixel accuracy and inference speed. To train our model, we use two different datasets. For indoor environments we use ADE20K, one of the largest open source dataset for semantic segmentation and scene parsing, released by the MIT Computer Vision team. For outdoor environments, we use CMT seasons [57], a large scale datasets with images captures all around the year. However, there are other large databases available, for indoor environments, [58], [59], and outdoor scenes [60].

3.5.2 Navigable Space Segmentation

To segment the occupation of 3D space, multiple approaches can be followed. A common representation of the free and occupied space are cost-maps, most of the time they are 2D representation of the world. They are widely used for autonomous navigation in robotics, as they are light-weight and efficient. Their disadvantage is that they are 2D and cannot encode complex 3D information. Another method consists in using octrees, an efficient representation of 3D space. An octree is a tree data structure in which each internal node has exactly eight children. Unlike voxels, octrees split 3D space of blocks of different size. The blocks are small if more resolution is needed locally, or big if the information is consistent over a large region. This makes octrees highly memory efficient, as the discretization of the 3D space depends on information. A popular implementation of octrees for occupancy segmentation is Octomap [61]. Its main issue is that it is relatively heavy, and may not be able to run in real-time on the edge-compute-device when the map gets large.

To minimize the drone's computational load, we chose to build a local 2D cost-map. In this context, local means the map is expressed in the depth camera's frame, and only contains the information provided by the last depth measurement. This map provides the maximum height of the obstacle in the field of view of the drone. To generate such a map, we use the point cloud generated by the depth camera. This point cloud is then aligned with the gravity frame. In other words, the point cloud is rotated about the roll and pitch axis of the drone. This transformation ensures that the z of the point cloud is aligned with the z of the global frame. To find the transform, we can use two things, the state estimated by the sensor fusion algorithm, or if the state estimation is not available, one can use an IMU based method. Here, when the state estimation is not available, we use the IMU filter Madgwick to smooth the IMU measurement and estimate the orientation of the drone in the quaternion space. This quaternion orientation is then altered to only include an always null yaw. This new orientation is then converted into a rotation matrix, which we use to project the points into the gravity aligned frame. With this projection done, we can now use the z values of the points directly as the relative height between the obstacles and the drone. Finally, to create the cost map, we subdivide the (x,y) space in the drone frame into small bins. Using the projected point cloud, we take the maximum z value of the points in each bin and save store their value in the cost-map.

3.5.3 Obstacle detection

To detect obstacles, we consider two categories, fixed and dynamic obstacles. The detection and tracking of the dynamic obstacles is done just like the robots. The obstacles are detected inside the images, tracked, and the position of each tracked object is then estimated. The main difference with the robot detection, is what is being shared with the other robots. In the case of the detection and tracking of the robots, the detector-robot shares only the position of the target-robot. However, in the case of the obstacle detection, the detector-robot outputs the 3D bounding box of the detected obstacle. As the detection network is not 3D aware, the 3D bounding box is computed by supposing that the width and depth of the object are of similar size.

When the objects are known to be static, the strategy is different. In this scenario, we rely on 3D space tracking only, and consider that the objects cannot move. Hence, the state of the 3D kalman filter can be changed to equation 16. Likewise, the dynamic part of the kalman filter defined in equation 12 can be simplified as in equation 16.

$$X = \begin{bmatrix} x \\ y \\ z \\ h \\ w \end{bmatrix} \tag{16}$$

$$F = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{17}$$

The uncertainty on Q is also changed to be a diagonal matrix with very small values to ensure that the covariance remains positive-definite. Similarly to the dynamic obstacles, the static obstacles are shared with the other robots as 3D bounding boxes. Different approaches could have been taken, such a grid based approaches, however, we believe this approach to be more scalable and easier to use.

3.6 Generic Sensor Fusion Theory

Sensor fusion consists in using data from multiple sensors such that the resulting information is more accurate. Kalman filters are commonly used tools to perform sensor fusion for pose estimation. This family of filter has numerous variations. In this deliverable, we compare three classical observers on a state estimation task: an EKF, a MSCKF, and an UKF. Additionally, the SWLS method has also been adapted to evaluate its applicability to state estimation.

3.6.1 Notation and Generalities

Typical robotic systems can often be formulated with differential equations, as shown in Equation 18.

$$\dot{x} = f(x, u) + n_x \tag{18}$$

Similarly, the measured variables follow the equation given in 19. Where, x is the state vector representing our system, u is the control input, y is the measured variables, f represents the dynamic of the system, h is the observation model, and n_x and n_y are multivariate zero-mean Gaussian noises.

$$y = h(x) + n_y \tag{19}$$

3.6.2 Extended Kalman Filter

In estimation theory, Kalman filtering is an algorithm that estimates of some unknown variables given the measurements observed over time. This filter has a relatively simple form and requires small computational power, making it more popular in computational constrained platform. It can also indirectly estimate observable variables over time. The main advantage of this filter is the minimization of uncertainty effects on the estimated variables. The EKF is the nonlinear version of the Kalman filter, which linearizes about an estimate of the current mean and covariance.

This filter can be adapted to observe the dynamics of the system. To do so, we assume that we obtain the state vector x_k at the time k and the corresponding covariance P_k. In the state vector, we could include observable variables to estimate their values.

In the standard estimation process, a Kalman Filter (KF) takes sensor measurements as input and, given a motion model, calculates the state of the system. Thus, leveraging the current state x_k, control inputs u_k, and the dynamic equations (18), we can get the next state x_{k+1} at time $k + 1$. The model propagation is given in 20.

$$\begin{aligned} x &\leftarrow x + \dot{x}dt \\ x_{k+1} &= x_k + f\left(x_k, u_k\right) dt \end{aligned} \tag{20}$$

Next, we need to derive the error state dynamic model:

$$\begin{aligned} J &= \partial f / \partial x \\ \delta \dot{x} &= J \delta x \end{aligned} \tag{21}$$

J is the Jacobian matrix of f with respect to x. After obtaining the matrix J, we can predict the covariance P_{k+1} at the time $k+1$:

$$\begin{aligned} F &= \exp\left(Jdt\right) \\ P_{k+1} &= FP_kF^T + Qdt \end{aligned} \tag{22}$$

Where, F is the process transition matrix, Q is the preset covariance of state noise.

$$Q = \begin{bmatrix} \sigma_{x_0}^2 & & \\ & \ddots & \\ & & \sigma_{x_{\dim(x)-1}}^2 \end{bmatrix} \tag{23}$$

The i-th element of the main diagonal corresponds to the noise variance of the i-th state. All non-diagonal elements are 0.

After receiving the measurement, we need to update the state vector and covariance:

$$\begin{aligned} &\hat{y} = h\left(x\right) \\ &H = \partial h / \partial x \\ &S = HPH^T + R \\ &K = PH^TS^{-1} \\ &dx = K\left(y - \hat{y}\right) \\ &x \leftarrow x + dx \\ &P \leftarrow \left(I - KH\right) P{\left(I - KH\right)}^T + KRK^T \end{aligned} \tag{24}$$

H is the Jacobian matrix of h with respect to x. R is the preset covariance of measurement noise.

$$R = \begin{bmatrix} \sigma_{y_0}^2 & & \\ & \ddots & \\ & & \sigma_{y_{\dim(y)-1}}^2 \end{bmatrix} \tag{25}$$

The i-th element of the main diagonal corresponds to the noise variance of the i-th measurement. All non-diagonal elements are 0.

3.6.3 Multi-State Constraint Kalman Filter

The idea of this part draws lessons from [62]. First, we define the following state:

$$\begin{aligned} x &= \begin{pmatrix} x_I & x_C \end{pmatrix} \\ x_C &= \begin{pmatrix} x_{c_1} & \cdots & x_{c_N} \end{pmatrix} \end{aligned} \tag{26}$$

Where, x_I represents the head state, and N represents the window size. We get x_{c_i} through the clone of x_I at different times when we get measurements. The corresponding covariance is given in Equation 27. Where P

is the covariance matrix of x, P_{II} is the variance of x_I, P_{CC} is the variance of x_C, and P_{IC} is the covariance between x_I and x_C. The full derivation of P is described in Equation 30.

$$P = \begin{pmatrix} P_{II} & P_{IC} \\ P_{IC}^T & P_{CC} \end{pmatrix} \tag{27}$$

Similarly to the EKF, leveraging the commands u_k along with the dynamic model, we can get the state vector x_{k+1} at time $k+1$.

$$x_I \leftarrow x_I + \dot{x}_I dt \tag{28}$$

The complete derivation of the covariance P_{k+1} at time $k+1$ is given in Equation 29. Where, Q is the preset covariance of the state noise. J, F, and Q are the same as in EKF (see Section 3.6.2).

$$\begin{aligned} F &= \exp(Jdt) \\ P_{II} &\leftarrow F P_{II} F^T + Qdt \\ P &\leftarrow \begin{pmatrix} P_{II} & F P_{IC} \\ P_{IC}^T F^T & P_{CC} \end{pmatrix} \end{aligned} \tag{29}$$

When receiving the measurements, we perform state augmentation, and the covariance must be changed accordingly:

$$P = \begin{pmatrix} I \\ \frac{\partial x_c}{\partial x} \end{pmatrix} P \begin{pmatrix} I \\ \frac{\partial x_c}{\partial x} \end{pmatrix}^T \tag{30}$$

The measurement update is similar to the EKF, except that the measurement must be selected within the whole window.

$$\begin{aligned} \hat{y} &= h(x) \\ H &= \partial h / \partial x \\ S &= HPH^T + R \\ K &= PH^T S^{-1} \\ dx &= K(y - \hat{y}) \\ x &\leftarrow x + dx \\ P &\leftarrow (I - KH) P (I - KH)^T + KRK^T \end{aligned} \tag{31}$$

where, R is the measurement noise over the whole window. H and R have the same meaning as in Section 3.6.2.

3.6.4 Unscented Kalman Filter

In this part, we base our work on [63]. We assume that the initial state vector x at is μ and the corresponding covariance is $\sum$.

$$x = \begin{pmatrix} \mu & \mu + \gamma\sqrt{\sum} & \mu - \gamma\sqrt{\sum} \end{pmatrix} \tag{32}$$

where, γ is the tuning parameter. According to the control input and dynamic model, we predict the state vector and covariance:

$$\begin{aligned} x_i &\leftarrow x_i + \dot{x}_i dt \\ \mu &\leftarrow \sum_{i=0}^{2l} w_m x_i \\ \sum &\leftarrow \sum_{i=0}^{2l} (x_i - \mu) w_c (x_i - \mu)^T + Qdt \end{aligned} \tag{33}$$

where, subscript i indicates column number, w_m and w_c are the corresponding weights, l is the state dimension, and Q is preset covariance of state noise. It has the same meaning as EKF part.

After receiving the measurement, state and covariance update are as follows:

$$\begin{aligned} x &= \begin{pmatrix} \mu & \mu+\gamma\sqrt{\sum} & \mu-\gamma\sqrt{\sum} \end{pmatrix} \\ \hat{y} &= h\left(x\right) \\ \mu_y &= \hat{y}w_m \\ S &= \sum_{i=0}^{2l}\left(\hat{y}_i-\mu_y\right)w_c(\hat{y}_i-\mu_y)^T+R \\ K &= \left(\sum_{i=0}^{2l}\left(x_i-\mu\right)w_c(\hat{y}_i-\mu_y)^T\right)S^{-1} \\ d\mu &= K\left(y-\mu_y\right) \\ \mu &\leftarrow \mu+d\mu \\ \sum &\leftarrow \sum -KSK^T \end{aligned} \tag{34}$$

where, R is the preset covariance of measurement noise. It has the same meaning as EKF part.

3.6.5 Sliding Window Least Square

The method of Least Squares (LS) is a standard approach in regression analysis to approximate the solution of over-determined systems. It minimizes the sum of the squares of the residuals made in the results of each individual equation. Its most prevalent application is in data fitting. Applied to the online identification of dynamic systems, the algorithm should consider the input and the output over multiple time-steps to avoid generating outliers due to noise or random faulty measurements from the sensors. Ideally, the LS method should use all the data collected during the system operation. However, in real-time applications, this might not be applicable for two reasons. First, the amount of accumulated data could saturate the computational resources available in the robot. Second, if the system's dynamics changes over time, the information collected in the past might not accurately represent the robot's state in a specific time-step. For this reason, we choose to apply the SWLS method, in which the LS algorithm is applied on the data of the last n time-steps. This way, we create a sliding window of data that is used within the algorithm. We adopt the similar approach presented in [64].

The optimization variables in the sliding window are defined as follows:

$$\chi = \begin{pmatrix} x_0 & x_1 & \cdots & x_n \end{pmatrix} \tag{35}$$

Where, n is the size of the sliding window to be tuned offline for each system. Between the two states of the sliding window, we will integrate the control input to establish their constraints. In order to avoid repeated integration during the optimization iteration, we define the following pre-integration items: $\alpha_{b_{k+1}}^{b_k}$, $\beta_{b_{k+1}}^{b_k}$, and $\gamma_{b_{k+1}}^{b_k}$.

Where, k and $k+1$ represent the timestamp of adjacent states in the sliding window, and b represents the body frame.

Next, we need to derive the error state dynamic model:

$$\begin{aligned} \delta z &= \begin{pmatrix} \delta\alpha & \delta\beta & \delta\gamma \end{pmatrix} \\ \delta z_{i+1}^{b_k} &= F\delta z_i^{b_k}+Gn \\ F &= \partial z_{i+1}^{b_k} \Big/ \partial z_i^{b_k} \\ G &= \partial z_{i+1}^{b_k} \Big/ \partial n \end{aligned} \tag{36}$$

We can then propagate the covariance of pre-integration measurement:

$$P_{i+1}^{b_k} = FP_i^{b_k}F^T+GQG^T \tag{37}$$

Finally, our optimization problem can be written as:

$$\min_{\chi} \left\{ \|r_{prior}\|^2 + \sum \left\| r_P^k\left(x_k\right) \right\|_{W_P^k}^2 + \sum \left\| r_D^k \left(\begin{array}{cc} x_k & x_{k+1} \end{array} \right) \right\|_{W_D^k}^2 \right\} \tag{38}$$

Where, r_{prior} represents the prior constraint, $r_P^k\left(x_k\right)$ represents the measurement residual, $r_D^k \left(\begin{array}{cc} x_k & x_{k+1} \end{array} \right)$ represents the dynamics residual. W_P^k and W_D^k are weight matrix, which can be regarded as the inverse of covariance matrices.

The expression of $r_P^k\left(x_k\right)$ is:

$$r_P^k\left(x_k\right) = y - \hat{y} = y - h\left(x_k\right) \tag{39}$$

The expression of $r_D^k \left(\begin{array}{cc} x_k & x_{k+1} \end{array} \right)$ is as follows:

$$r_D^k \left(\begin{array}{cc} x_k & x_{k+1} \end{array} \right) = \begin{bmatrix} \alpha_{b_{k+1}}^{b_k} - \hat{\alpha}_{b_{k+1}}^{b_k} \\ \beta_{b_{k+1}}^{b_k} - \hat{\beta}_{b_{k+1}}^{b_k} \\ \gamma_{b_{k+1}}^{b_k} - \hat{\gamma}_{b_{k+1}}^{b_k} \end{bmatrix} \tag{40}$$

The least square optimization problem is solved by Ceres[3].

3.7 Collaborative Sensor Fusion: UAV Use Case

In this subsection, we apply the different tools presented in Section 3.6 to our sensor-fusion task on a UAV. We start by introducing the system's dynamic model and measurement model. Then, we detail how our algorithms track the changes in the state vector and discuss the results of each sensor fusion technique. To elaborate on the use case, we consider the detector and target drone as UAV 1 and UAV 2, respectively.

3.7.1 Dynamic and Measurement Model

In the following, we focus on the state estimation of UAV 2. The state estimation of UAV 1 is achieved using VIO and it is not described here.

To be able to estimate the pose of UAV 2, we first need to define the dynamic model of its IMU. We recall that the pose of the UAV 2 is estimated using both the relative position provided by UAV 1 and the IMU inside target drone. In Equation 41, we introduce the dynamic model of an IMU. Where, the raw accelerometer and gyroscope measurements from the IMU are noted a and, ω respectively.

$$\begin{aligned} {}_W\dot{r}_{WI} &= {}_Wv_I \\ {}_W\dot{v}_I &= R\left(q_{WI}\right)\left(a - b_a - n_a\right) + g \\ \dot{q}_{WI} &= \tfrac{1}{2} q_{WI} \otimes \left(\omega - b_g - n_g\right) \\ \dot{b}_a &= n_{b_a} \\ \dot{b}_g &= n_{b_g} \end{aligned} \tag{41}$$

With this model in mind we can thus define the state as in Equation 42. Where q, r and v represents the attitude, the position and the linear velocities of the system respectively, b_a is the acceleration bias and b_g is the gyroscope bias. Next, we define the following state vector x :

$$x = \begin{bmatrix} q \\ r \\ v \\ b_a \\ b_g \end{bmatrix} \tag{42}$$

[3]http://ceres-solver.org/

In order to simplify the notations, we do not include the coordinate frame's subscript in the dynamic equations, as it will not affect their understanding.

We can define the drone commands u as in Equation 43:

$$u = \begin{bmatrix} a \\ \omega \end{bmatrix} \tag{43}$$

Furthermore, we assume that we can obtain the following measurements from the state estimator and relative position calculator inside UAV 1.

$$\begin{aligned} y &= [r] \\ r &= {}_W r_{WI_2} = {}_W r_{WI_1} + R(q_{WI_1})_{I_1} r_{I_1 I_2} \end{aligned} \tag{44}$$

The VIO system of target drone can provide its own pose estimation ${}_W r_{WI_1}$ and $R(q_{WI_1})$, while the object detection can provide ${}_{I_1} r_{I_1 I_2}$ the relative position between the UAV 1 and UAV 2. In the end, we get the following system equation and measurement equation.

$$\begin{aligned} \dot{x} = f(x,u) + n_x &= \begin{bmatrix} \frac{1}{2} q \otimes (\omega - b_g) \\ v \\ R(q)(a - b_a) + g \\ 0 \\ 0 \end{bmatrix} + n_x \\ y = h(x) + n_y &= \begin{bmatrix} r \end{bmatrix} + n_y \end{aligned} \tag{45}$$

Where, n_x and n_y represent multivariate zero-mean Gaussian noise.

3.7.2 Particularization

To apply these filters to our problem, some modifications must be made. We first define the state used in the MSCKF, and then derive the equation of the SWLS for our specific problem.

MSCKF Particularization : As detailed in Section 3.6.3, to be able to apply the MSCKF filter, one needs to define its state. Equation 46 details the state that we will use in our state estimation problem.

$$\begin{aligned} x_I &= \begin{pmatrix} q & r & v & b_a & b_g \end{pmatrix} \\ x_{c_i} &= \begin{pmatrix} r_i \end{pmatrix} \end{aligned} \tag{46}$$

SWLS Particularization : Regarding the SWLS algorithm, the optimization variables in the sliding window are defined as follows:

$$x_k = \begin{pmatrix} q_k & r_k & v_k & b_{a_k} & b_{g_k} \end{pmatrix}, k \in [0, n] \tag{47}$$

Where, n is the size of the sliding window. We integrate the control input between the two states of the sliding window to establish their constraints. In order to avoid repeated integration during optimization iterations, we define the following pre-integration terms:

$$\begin{aligned} \alpha^{I_k}_{I_{k+1}} &= \int \int_{t_k}^{t_{k+1}} R^{I_k}_{I_\tau} (a - b_a - n_a) d\tau^2 \\ \beta^{I_k}_{I_{k+1}} &= \int_{t_k}^{t_{k+1}} R^{I_k}_{I_\tau} (a - b_a - n_a) d\tau \\ \gamma^{I_k}_{I_{k+1}} &= \int_{t_k}^{t_{k+1}} \frac{1}{2} q^{I_k}_{I_\tau} \otimes (\omega - b_g - n_g) d\tau \end{aligned} \tag{48}$$

Where, t_k and t_{k+1} are the timestamp of adjacent states in the sliding window, and I is the IMU frame.

The propagation process of the pre-integration terms are as follows:

$$\begin{aligned}
\alpha_{i+1}^{I_k} &= \alpha_i^{I_k} + \beta_i^{I_k} dt + \tfrac{1}{2} R\left(\gamma_i^{I_k}\right)(a - b_a)dt^2 \\
\beta_{i+1}^{I_k} &= \beta_i^{I_k} + R\left(\gamma_i^{I_k}\right)(a - b_a)dt \\
\gamma_{i+1}^{I_k} &= \gamma_i^{I_k} \otimes \begin{pmatrix} 1 \\ \frac{1}{2}(\omega - b_g)dt \end{pmatrix}
\end{aligned} \tag{49}$$

dt denotes the time interval between two adjacent control inputs. Then, we derive the error state dynamic model by linearizing the pre-integration terms as:

$$\begin{aligned}
\delta z &= \begin{pmatrix} \delta\alpha & \delta\beta & \delta\gamma & \delta b_a & \delta b_g \end{pmatrix} \\
\delta z_{i+1}^{I_k} &= F \delta z_i^{I_k} + Gn \\
F &= \partial z_{i+1}^{I_k} \Big/ \partial z_i^{I_k} \\
G &= \partial z_{i+1}^{I_k} \Big/ \partial n
\end{aligned} \tag{50}$$

The expression of $r_D^k \begin{pmatrix} x_k & x_{k+1} \end{pmatrix}$ is as follows:

$$\begin{aligned}
r_D^k \begin{pmatrix} x_k & x_{k+1} \end{pmatrix} &= \begin{bmatrix} \alpha_{I_{k+1}}^{I_k} - \hat{\alpha}_{I_{k+1}}^{I_k} \\ \beta_{I_{k+1}}^{I_k} - \hat{\beta}_{I_{k+1}}^{I_k} \\ \gamma_{I_{k+1}}^{I_k} - \hat{\gamma}_{I_{k+1}}^{I_k} \\ b_{a_{k+1}} - b_{a_k} \\ b_{g_{k+1}} - b_{g_k} \end{bmatrix} \\
\hat{\alpha}_{I_{k+1}}^{I_k} &= R_w^{I_k}\left(p_{I_{k+1}}^w - p_{I_k}^w - v_{I_k}^w dt - \tfrac{1}{2} g dt^2\right) \\
\hat{\beta}_{I_{k+1}}^{I_k} &= R_w^{I_k}\left(v_{I_{k+1}}^w - v_{I_k}^w - g dt\right) \\
\hat{\gamma}_{I_{k+1}}^{I_k} &= q_w^{I_k} \otimes q_{I_{k+1}}^w
\end{aligned} \tag{51}$$

4 Software and Hardware Architecture

To deploy our algorithms, we use a Jetson Xavier NX from Nvidia, it is one of the few onboard computers on the market that comes with a GPU in such a small form-factor. Also, this board is optimized for robotic uses, and its power consumption can be tweaked to allocate more (or less) power to the Central Processing Unit (CPU) or GPU. This is particularly useful to us, as on a drone, the power budget for the onboard computer is limited. The main drawback of using this board comes from its OS, as of today, the Jetson comes with Ubuntu 18.04. This is an issue as Ubuntu 18.04 uses Python 2.7 as by default, and so does the ROS Melodic, the version of the ROS that was developed for Ubuntu 18.04. In this project, we use ROS to run our algorithms as it comes with pre-package sensor interfaces, and allows for simple and efficient implementation of communication between threaded processes. As explained earlier, the main issues arise from the fact that ROS Melodic uses Python 2.7, which means that we cannot use state-of-the-art ML framework in ROS natively. One solution could be to use an Ubuntu 20.04 docker container inside the Xavier to execute the machine learning algorithms, while the navigation stack: MAVROS (MAVLink + ROS) runs on the native ROS installation. However, this increases the complexity of the system as we need to set up a ROS Master/Slave configuration on the Xavier itself, moreover, requiring two different OS versions is cumbersome and harder to maintain. An alternative solution, the one we choose, is to get rid of python for ML applications, and use C++ instead. Nowadays, most ML platforms provide simple means to convert or compile their models to binaries that can be called directly from C++ code, alleviating the need for python entirely, making their execution faster, at the cost of increased code complexity. Hence, for now, our whole development stack is built upon Ubuntu 18.04 and ROS Melodic. In addition to the framework to execute our algorithms, we developed simulation environments that allow us to test our algorithms before deploying them on a real system. This simulation is also tied to Ubuntu 18.04, and ROS Melodic on the Gazebo robotics simulator. Nonetheless, in an order to future-proof our work, we also provide ROS2 (foxy) compatible code.

The proposed SESAME MRS perception and collaborative sensor fusion ROS framework with nodes, topics, is shown in Figure 8. ROS nodes such as position estimation and VIO-stereo run on the onboard computer of a detector drone (monitor UAV) and provides the global position to the state estimator ROS node of the target drone (spraying UAV). The state estimator ROS node of the target drone takes pose data from its onboard IMU and position data from the detector node, performs the sensor fusion, and provides the accurate pose of the target drone to the navigation ROS node.

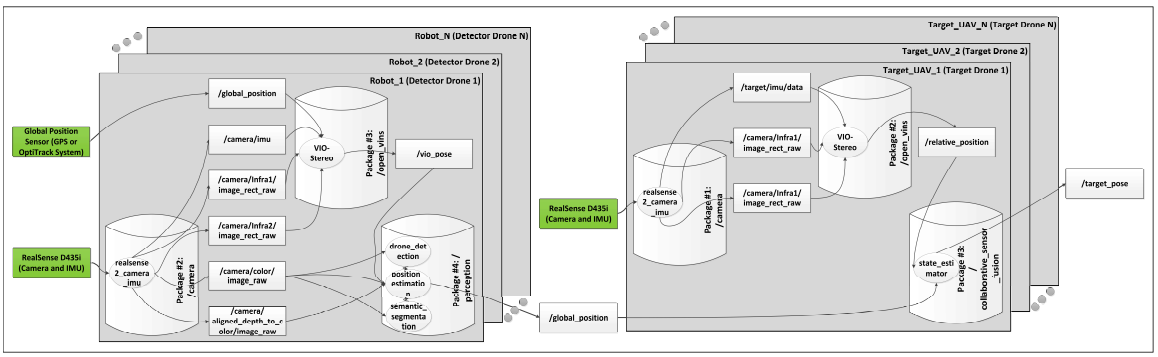

Figure 8: ROS Framework for Perception and Collaborative Sensor fusion.

In the first part, we provide a brief description of the experimental setup. The next section sketches the experimental overview with an operational scenario of autonomous pest management use case. Finally, we describe the test results in simulation and in real experiments using the AeroLab.

4.1 Experimental Setup

We illustrate perception and collaborative sensor fusion with an autonomous pest management use case operational scenario that involves two robots (detector drone and target drone) operating in a dynamic environment. The proposed SESAME MRS framework for autonomous pest management use case operational scenario is shown in Figure 9.

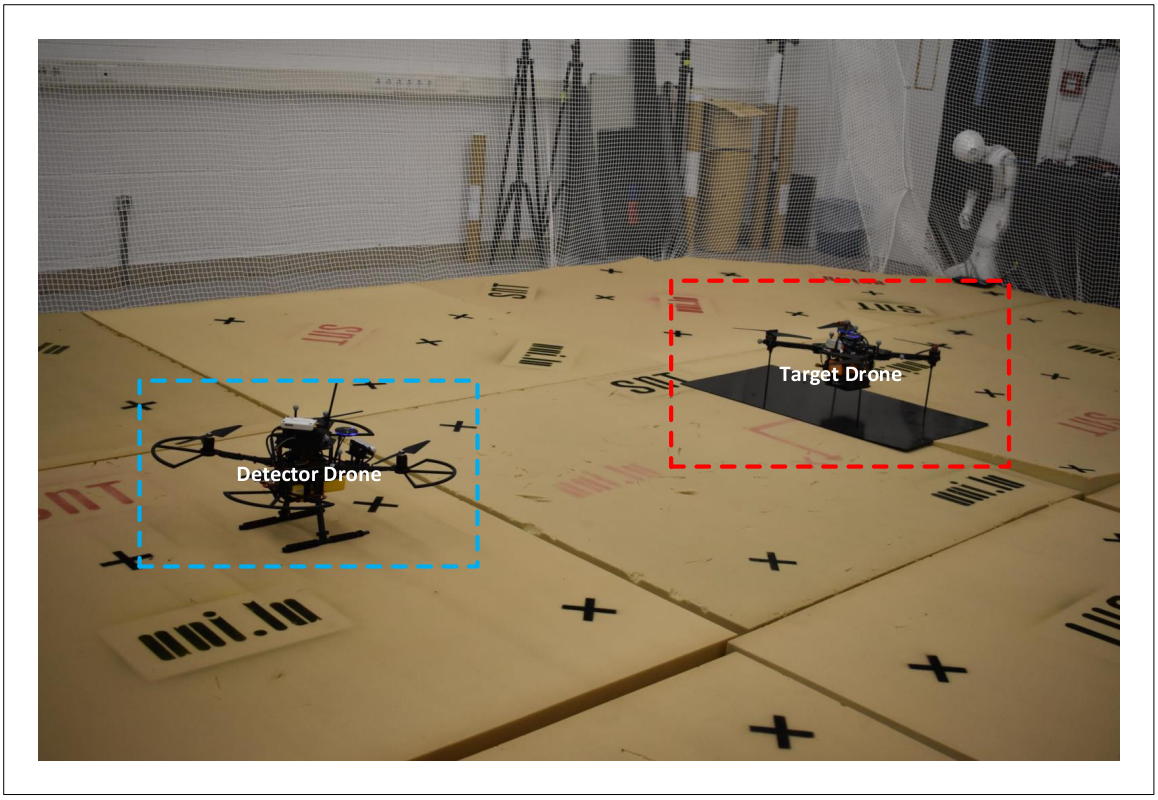

Figure 9: MRS Framework Experimental Setup.

The complete sensory equipment of the two robots (detector drone and target drone) is explained below.

4.1.1 Detector Drone

The detector drone has Pixhawk 4 [65] as flight controller (see Figure 10). The perception algorithm runs on the Nvidia Jetson Xavier NX onboard computer with RealSense D435i Camera.

- **Intel RealSense D435i Camera (Camera with IMU)**: The perception and sensor fusion algorithms described in Section 3 use the stereo, depth, and Red, Green and Blue (RGB) streams as well as the IMU data from this camera (see Table 1 for the full technical specifications). It is one of the most widely used cameras in robotics due to its performance, low cost, and small size factor. It is designed for simple integration, easy-to-use setup, and portability. The D435i belongs to the D400 Intel series, and it is based on the D4 Vision Processor. On of the key feature of this camera is its ability to generate depth images without external computational power. It is equipped with a Depth Module able to work in both indoor and outdoor environments with high resolution and medium-range. As for the IMU, it is

(a) (b) (c)

Figure 10: Pixhawk 4 used in the experiments. a) Detector Drone. It is equipped with a b) Nvidia Jetson Xavier NX, and c) Intel RealSense D435i Camera along with IMU.

Table 1: Intel RealSense D435i Camera Technical Specifications

Intel RealSense D435i Camera	**Specifications**
Depth Technology	Active Stereoscopic
Operating Range (Min-Max)	$\sim$.11m - 10m
Depth Resolution and FPS	1280 X 720
Depth Field of View	85.2 x 58
RGB Sensor	Yes
Tracking Module	No
Dimensions	90 mm x 25 mm x 25 mm
System Interface Type	Universal Serial Bus (USB) 3.0 Type C

used to estimate the translation and rotation of the robot along Six Degrees of Freedom (6DoF). This IMU combines accelerometers, and magnometers with gyroscopes to detect translation on three axes, as well as pitch, yaw and roll. Finally, the camera comes with an RGB Module, and two Infrared Modules. The RGB module is designed for a resolution up to 1920×1080. To create the depth images, the RealSense uses the two infrared cameras and leverages an Infrared Radiation (IR) projector to improve the robustness of the stereo depth measurement.
Depending on the lighting condition, the usable range varies between 0.10 m to 10 m. The camera features a free software developer's kit: the Intel RealSense Software Development Kit (SDK) 2.0, and ROS node which we use in this project.

- **Nvidia Jetson Xavier NX**: The drone detection, position estimation, drone tracking, and segmentation methods described in Section 3 all run on an Nvidia Jetson Xavier NX (see Table 2 for technical specifications).The Xavier NX belongs to Nvidia's Jetson product family, a line-up of high performance and low power consumption boards, design to run real-time artificial intelligence algorithms. They are well-suited for robotic applications thanks to their small factor and easy integration with other systems. In particular, the Jetson Xavier NX board is one of the smallest device in the Jetson family and uses only 10W of power in its high-performance profile. The board is built around a 6 cores ARM Cortex CPU and a 384-core Volta GPU with 8GB of shared memory.
  Regarding I/O it is equipped with four 3.1 USB ports that allows to connect up to 4 cameras, an High-Definition Multimedia Interface (HDMI) as well as an ethernet port. Moreover, it offers the possibility to use two power interfaces: through a barrel-jack or trough type-c connector, which eases the integration.

4.1.2 Target Drone

The selected base platform for target drone is created mainly from Commercially available Off-The-Shelf components (COTS) components and 3D printed parts as shown in Figure 11. The platform is built from the X-layout quadrotor frame on which we have mounted four T-Motor MN3508 KV380 with 1552 folding

Table 2: Nvidia Jetson Xavier NX Module Technical Specifications

Nvidia Jetson Xavier NX	**Specifications**
AI Performance	21 Tera Operations Per Second (TOPS)
GPU	384-core Nvidia Volta GPU with 48 Tensor Cores
glscpu	6-core Nvidia Carmel ARM v8.2 64-bit glscpu 6MB L2 + 4MB L3
Memory	8 GB 128-bit LPDDR4x 59.7GB/s
Storage	16 GB eMMC 5.1
Power	10 W | 15 W | 20 W
PCIe	1 x1 (PCIe Gen3) + 1 x4 (PCIe Gen4), total 144 GT/s
CSI Camera	Up to 6 cameras (24 via virtual channels) 14 lanes (3x4 or 6x2) MIPI CSI-2 D-PHY 1.2 (up to 30 Gbps)
Video Encode	2x 4K60 | 4x 4K30 | 10x 1080p60 | 22x 1080p30 (H.265) 2x 4K60 | 4x 4K30 | 10x 1080p60 | 20x 1080p30 (H.264)
Video Decode	2x 8K30 | 6x 4K60 | 12x 4K30 | 22x 1080p60 | 44x 1080p30 (H.265) 2x 4K60 | 6x 4K30 | 10x 1080p60 | 22x 1080p30 (H.264)
Display	2 multi-mode DP 1.4/eDP 1.4/HDMI 2.0
DL Accelerator	2x NVDLA Engines
Vision Accelerator	7-Way VLIW Vision Processor
Networking	10/100/1000 BASE-T Ethernet
Mechanical	69.6 mm x 45 mm, 260-pin SO-DIMM connector

propellers, the CUAV V5+ flight controller unit running the PX4 flight stack, IMU, the RealSense D435i (Camera with IMU), and an Intel NUC onboard computer. This NUC from intel contains an Intel i7 glscpu paired with 16 GB of Random Access Memory (RAM). It runs Ubuntu 18.04 LTS with ROS Melodic [66].

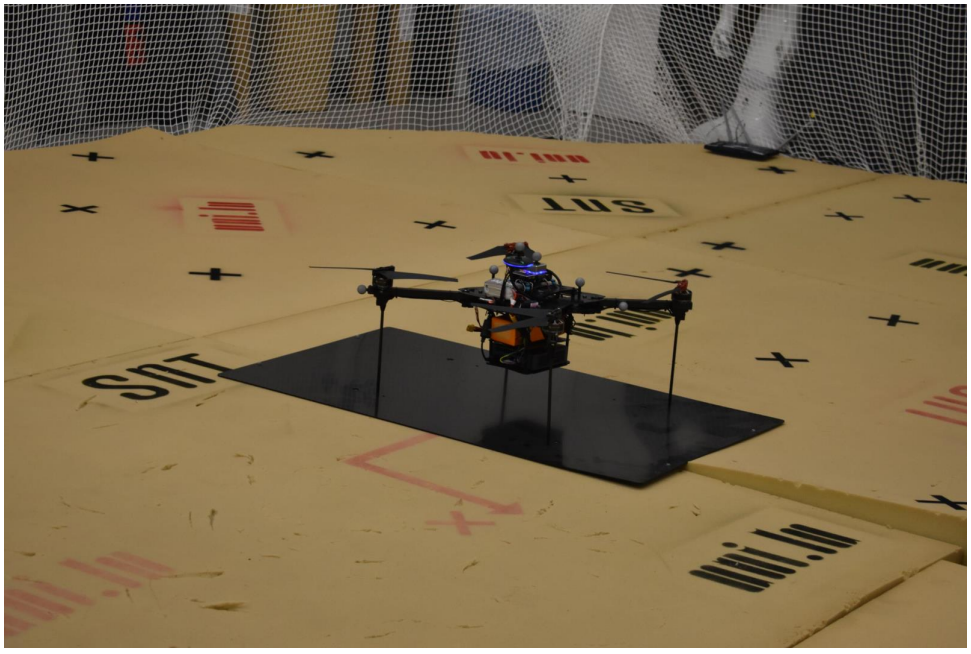

Figure 11: Target Drone

5 Experimental Overview and Results

5.1 Experimental Overview

In this section, we describe the experimental plan for perception and collaborative sensor fusion. In order to conduct the in-lab experiments, we divided the experiments into two parts — a) perception, and b) sensor fusion.

- **Perception**: We use YOLOv5 [54] in order to detect the object. Specifically, we consider the Autonomous Pest Management (APM) in viticulture use case to develop the proof of concept. Therefore, we consider the drone as an object in this context. As discussed in Section 3.2, the traditional YOLOv5 network recognizes 80 classes. However, the accuracy level in detecting the drone is low. Hence, the first step of object detection is to create a data set for the drone in order to train the YOLOv5 network. We use two heterogeneous drones — a) detector drone and b) target drone to generate the datasets. We describe the in-lab experimental scenario (Figure 12) for the real operational scenario of the APM in viticulture use case (Figure 13). Robot 1 acts as a detector drone with a camera and IMU. On the other hand, Robot 2 acts as a target drone with an IMU. Once the detector drone acquires the images of the target drone, the next step is to generate data set and train the YOLOv5 network as described in Section 3.2. Finally, the detector drone (Robot 1 in Figure 12) detects the target drone (Robot 2 in Figure 12) as described in Section 3.2.2. Additionally, the drone detection algorithm generates the relative position of target drone (Robot 2) from the detector drone (Robot 1), which is an essential part of the sensor fusion.
- **Sensor Fusion**: One of the primary objectives of the Task 2.3 is sensor fusion. In this context, the target drone (Robot 2 in Figure 12) loses its localization information due to sensor failure. Therefore, the target drone requires some external input for localization. Sensor fusion plays an essential role here. As shown in Figure 12, the detector drone (Robot 1 in Figure 12) detects the target drone using its vision and produces a relative position P_{12} of the target drone. Also, the detector drone knows its current position P_{w1} in the global frame. Therefore, the the position of the target drone with respected to global frame is $P_{w2} = P_{12} + P_{w1}$. The target drone receives its relative position in real-time from the detector drone in order to pass its current position to EKF for pose estimation.

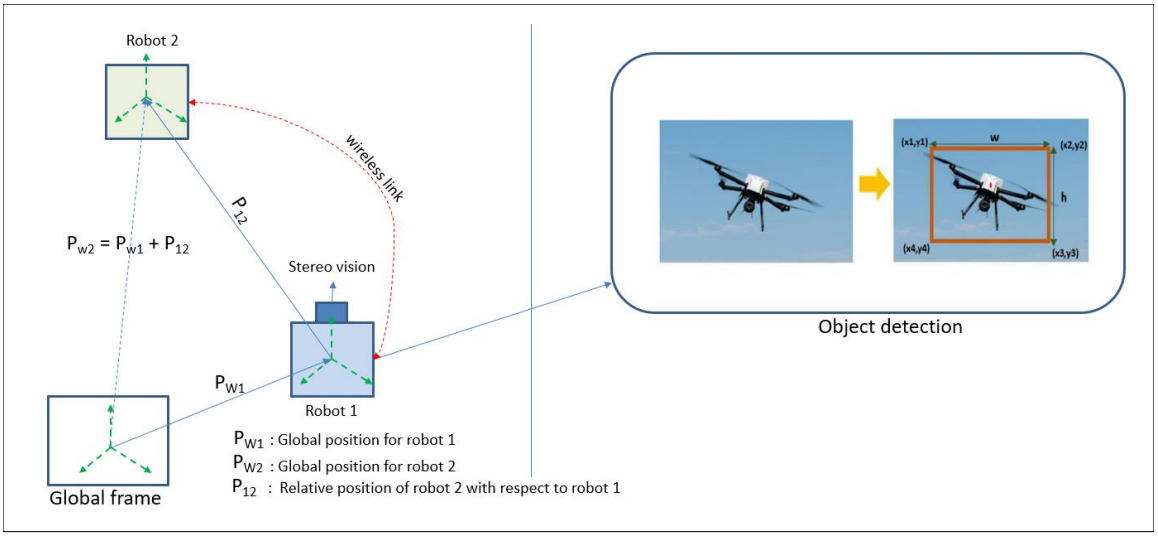

Figure 12: Experimental Overview.

Figure 13: Operational Scenario: MRS Mission in Autonomous Pest Management in Viticulture Use Case.

5.2 Experimental Results

This section covers the proposed perception and collaborative sensor fusion experimental evaluation results on simulation and prototype research MRS quad-rotor.

5.2.1 Detection Experiment with Gazebo Simulation

Initially, using the XTDrone [67] robotic simulator on the Gazebo simulation platform, we set up the virtual environment with a detector drone and target drone. On the detector drone, we set up the virtual RGB-D camera to match the geometry of the RealSense D435i (same field of view and image resolution). The detected results of the proposed drone detection with the RGB-D camera on an XTDrone simulator are shown in Figure 14.

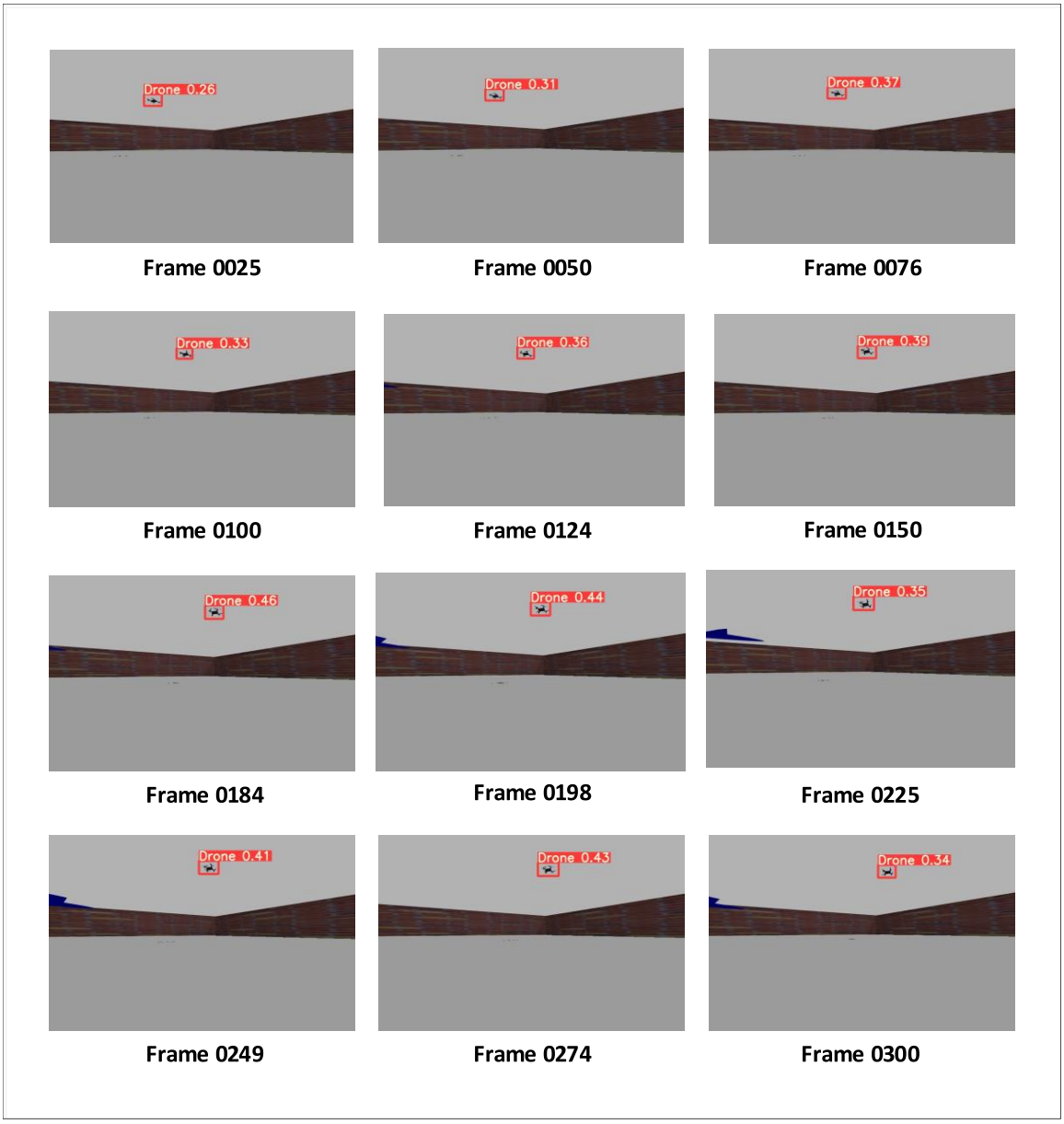

Figure 14: Simulation Testing Results.

5.2.2 Position Estimation Experiment

In addition to testing the detection algorithm, we also evaluate the complete vision-based position estimation stack. That is, the combination of the object detection, tracking of the objects, and the position estimation of the tracked objects. To do so, we simulate both drones with simulated GPS and IMU noises, and run state estimation algorithms (EKFs) on them. This gives us complete transform trees that can be leveraged to know the position of both drones within a global frame. In addition to these state estimation algorithm, we also simulate a RealSense d435 on the observer drone whose outputs are sent to our visual position estimation algorithms. This simulated RealSense also has noise in the depth image to ensure the readings are not perfect. Figure 15 shows the experimental setup view from RViz. On the images to the left one can see both the tracking and detection algorithms outputs. On the world view to the right, one can see the transforms of both drones in the global frame, the large green arrow shows the estimated position of the target drone in the global frame.

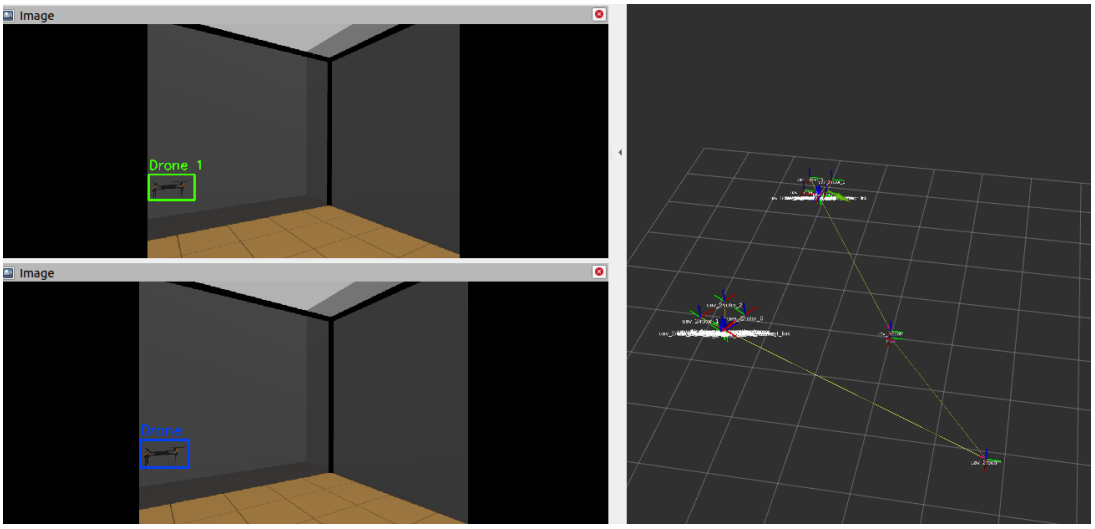

Figure 15: The detection, tracking, and position estimation working jointly in simulation.

To measure the accuracy of the vision-based position estimation, we make the observer drone fixed and make the target drone fly in a 3D grid pattern, which makes it cover a three by four by four meters volume. Then we aggregate the results by 1 meter ranges based on the distance between the target and observer drone. Using this, we can see if the distance between the two drones has an impact on the accuracy of the position estimation. Figure 16 shows the error or the distance between the estimated target drone position and the actual target drone position.

As can be seen on this figure, the farther the object the higher the error, this is shown by the median value increasing for each distance range. Additionally, we can also see that the spread of the error distribution is also getting larger as the distance between the drones increases. One of the reason behind this behavior is linked to the RealSense. The Realsense is not cost-effective 3D sensor, which means that it lacks precision. Another is that this error also includes the position error of the observer drone, which means that the total position error was most of the time around 0.2m which is comparable to an RTK GPS.

For completeness, we also show in figure 17 the trajectory of the target drone along with the trajectory estimated by our algorithm. As can be seen, the estimated trajectory is noisy and could use some sort of filtering. However, we do not apply a Kalman filter on top of it as it will be processed inside one later one. If we added this extra filtering step, it could introduce some delay, deteriorating the accuracy of the collaborative pose estimation algorithms.

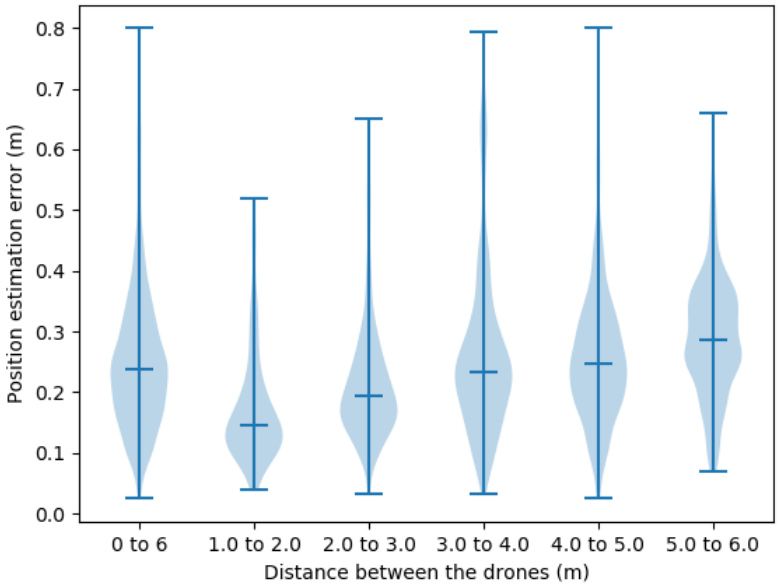

Figure 16: The error in meters of the visual position estimation algorithms. The blue shapes show the sample distribution, when the middle bars show the median.

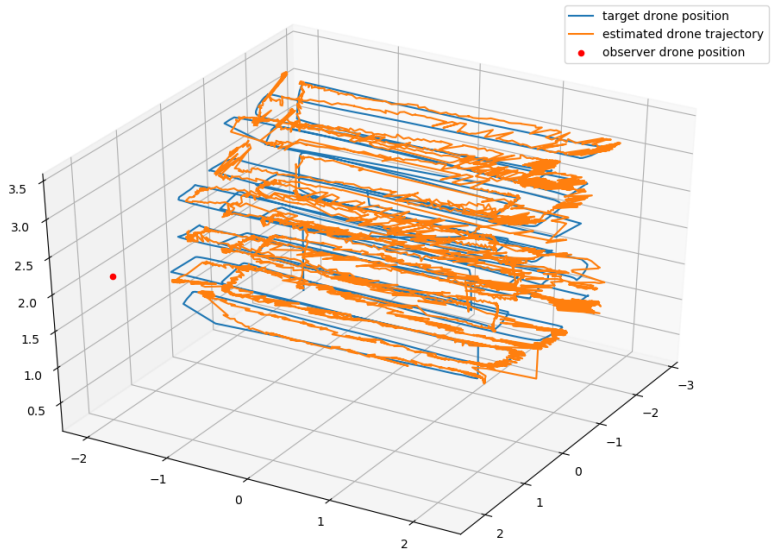

Figure 17: The position of both drones during the experiment. In blue, the trajectory followed by the target drone; in orange, its trajectory estimated by our algorithms.

Overall, our algorithms are accurate and fast, taking on average less than 7.3ms to run the full vision stack on a laptop. Most of this time is taken by the inference of the neural-network, 7ms, with the tracking and position estimation taking 75μs and 50μs on average. The initial preprocessing of the image takes about 150μs. On the Jetson, only the neural-network inference time changes significantly, with 33ms on average.

5.2.3 Obstacle detection experiments

To evaluate the obstacle detection experiments, we apply our algorithms to different types of objects, here we apply them to rocks. The distance to the rocks is significantly easier to measure, and there, instead of using the minimal distance, we use the center distance. All the rocks are detected and placed at their correct place in the global frame.

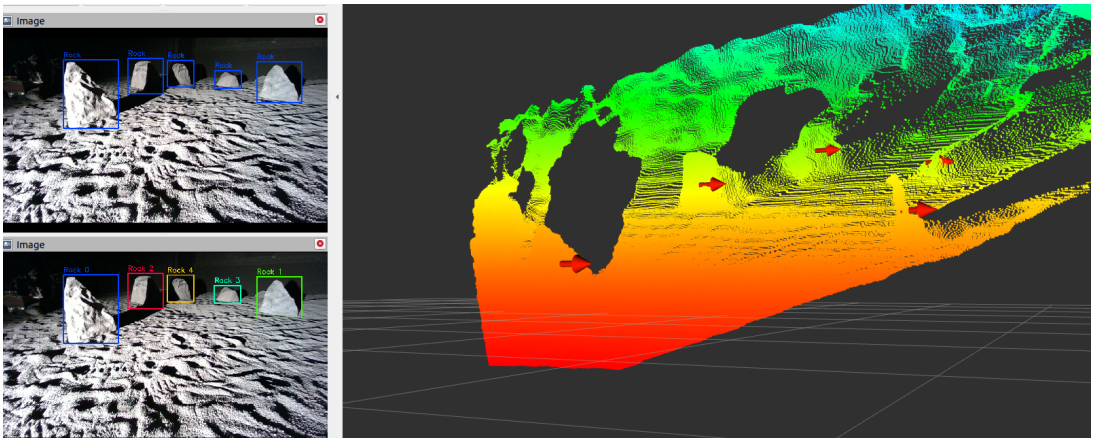

Figure 18: The detection, tracking, and position estimation working jointly to estimate the position of rocks. The red arrows show the position of the rocks in the global frame.

5.2.4 Experiments with Simulated Relative Position

The four state estimation algorithms mentioned in the previous section 3.7 are first tested on a public dataset. The EuRoC Micro Aerial Vehicle (MAV) Dataset [68] is a collection of visual-inertial datasets acquired from on-board MAVs. Each dataset contains stereo images, synchronized IMU measurements, and accurate motion and structure ground-truth. To use this dataset to evaluate our sensor fusion algorithm, we select V2_02_Medium as the target drone and V1_02_Medium as the detector drone. To simulate our MRS, the detector drone uses image and IMU data to run its VIO algorithm, as well as a noisy groundtruth to that it uses like a GPS. For the target drone, we use the IMU and a noisy groundtruth. This scenario simulates a loss of a positioning sensor on the target drone that is replaced by a noisy estimates of its position. The relative position is calculated by using the ground truth from the two datasets. In order to simulate the real relative position measurement, different levels of noise are added to the ground truth of the relative position, from 1cm to 10cm. The results of different algorithms (EKF, MSCKF, UKF, and SWLS) are shown from the Figure 19 to 22.

EKF Figure 19 shows the localization results obtained with the EKF method and the localization error. The position in three direction is aligned with the ground truth. As expected, we can see that the localization error is increasing with the increments of relative position noise. The maximum measured error is around 35cm, with 5 large error peaks reaching an error above 25cm, otherwise, the error remains under 20cm. From Figure 19 we can see that the trajectory is mostly smooth for all noise values and remains close from the ground truth. However, we can see that on the sharpest turns, the trajectory becomes noisier. We believe this is related to the heading estimation of the drone that is acquired using the IMU only. These sensors are sensitive to noise and are known to drift over time.

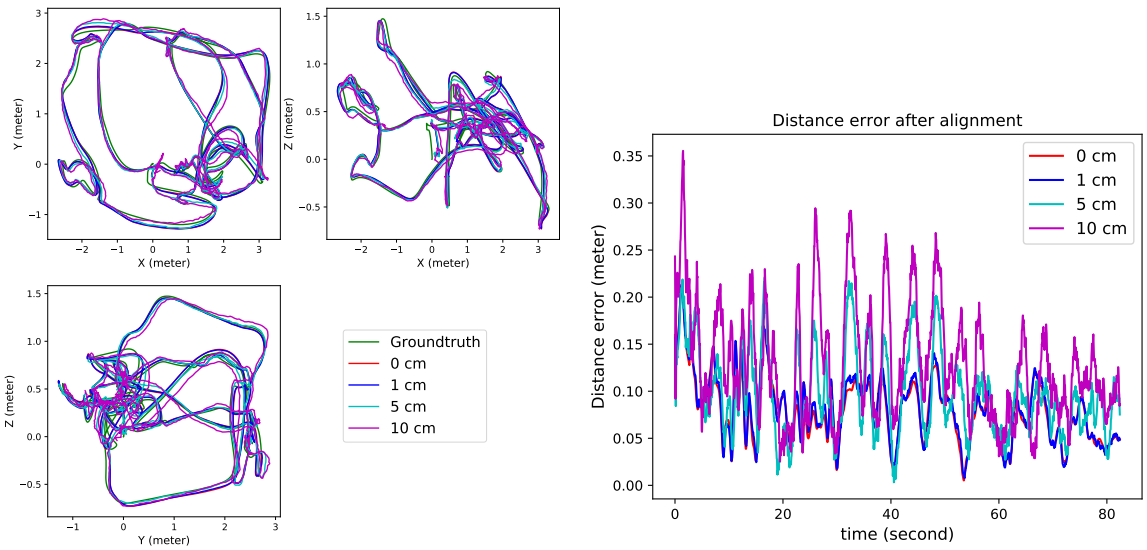

Figure 19: Left: Different views of the aligned trajectories from the EKF method evaluated with different levels of simulated relative position noise. Right: The distance error of the aligned trajectories for the EKF method evaluated with different levels of simulated relative position noise.

MSCKF Figure 20 shows the localization results obtained with the MSCKF method and the position error. The MSCKF is a variant of EKF, so their results should be similar in terms of trajectory smoothness and localization error. Nonetheless, the maximum error is smaller than the one of the EKF, with the first error peak not exceeding 30cm. Compared to the EKF, with its five peaks above 25 cm, here, only one other error peak is reaching above 25cm. Overall, the MSCKF seems to have a more consistent error than the EKF.

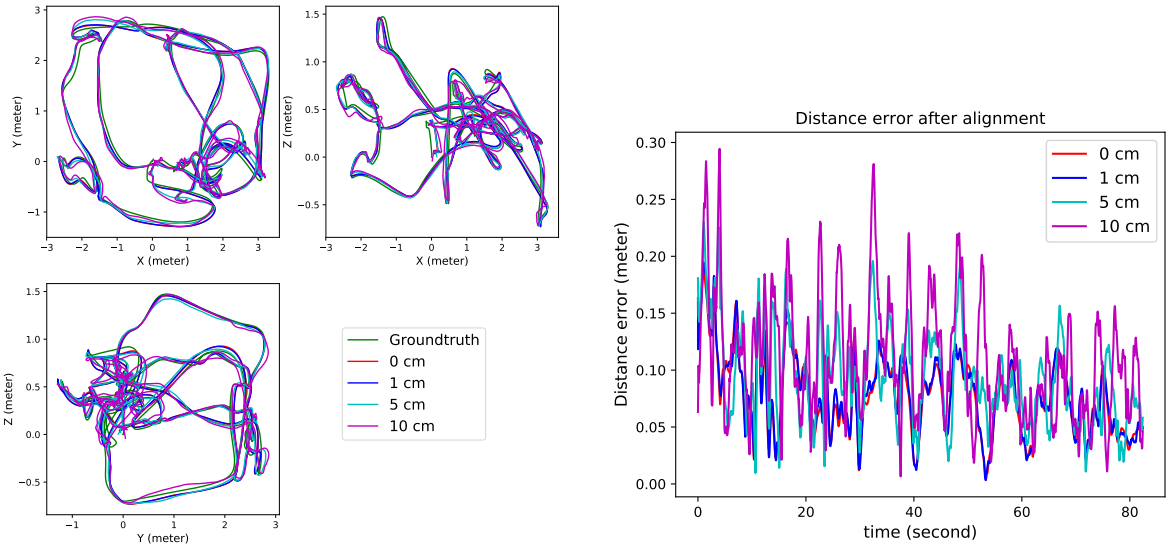

Figure 20: Left: Different views of the aligned trajectories from the MSCKF method evaluated with different levels of simulated relative position noise. Right: The distance error of the aligned trajectories for the MSCKF method evaluated with different levels of simulated relative position noise.

UKF Figure 21 shows the localization results obtained with the UKF method and the position error. Looking at the trajectory of the UKF, it can be seen that compared to the EKF the trajectory is less smooth. This is particularly visible as the noise reaches 10cm. This behavior can clearly be seen on the error graph where many narrow error peaks can be seen. In comparison, the EKF and MSCKF have fewer, but wider error peaks. The behavior of the UKF is problematic for robotics and control application as the noisy position means that

the velocity is noisy as well. This will make velocity based control more complicated than if we were using the EKF or MSCKF.

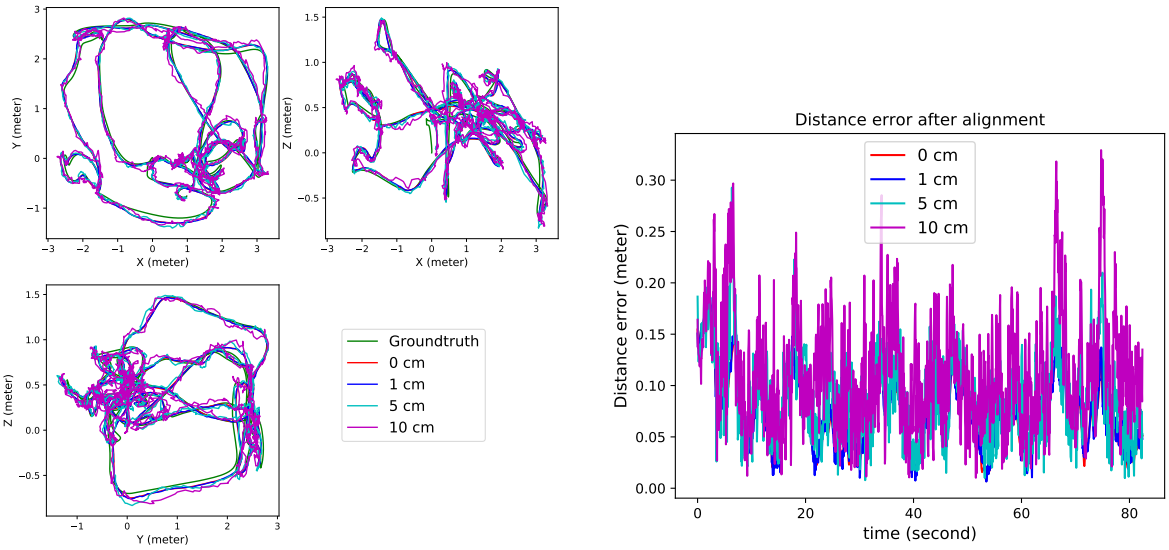

Figure 21: Left: Different views of the aligned trajectories from the UKF method evaluated with different levels of simulated relative position noise. Right: The distance error of the aligned trajectories for the UKF method evaluated with different levels of simulated relative position noise.

SWLS Figure 22 shows the localization results obtained with the SWLS method and the position error. This method seems to be giving the worst results of all the tested algorithms. The trajectory is very noisy, at both 5cm and 10cm noise. This is not surprising, as this method is particularly sensitive to the orientation of the drone. Here the orientation estimation is particularly noisy as the orientation is given by an IMU alone. In this kind of scenario, the roll and the pitch are often correct, they can be derived from the accelerometer measurements, but the yaw can be wrong, as it is only computed using the integration of the gyroscope data.

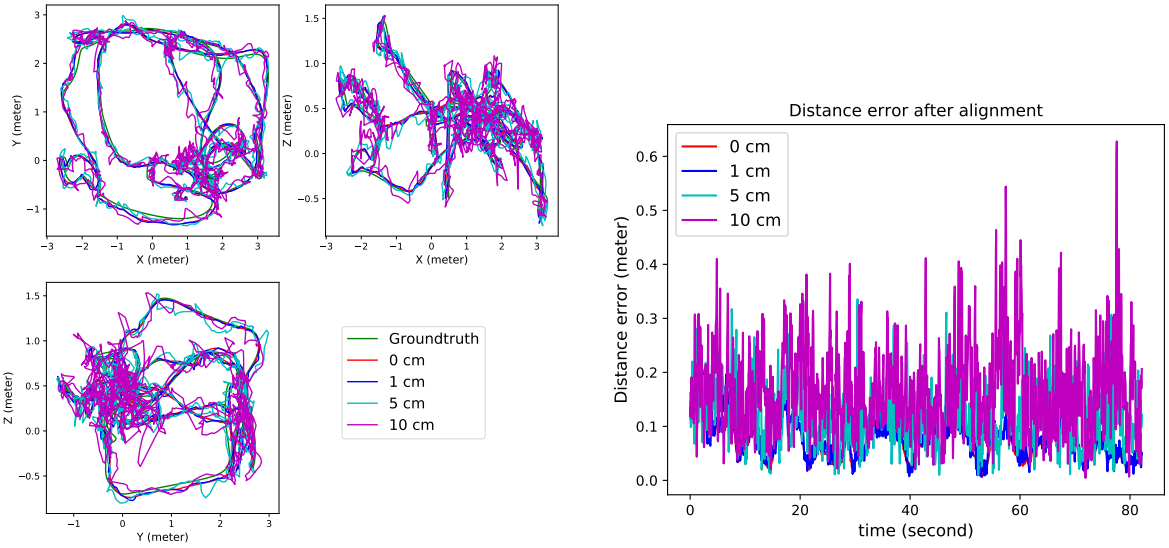

Figure 22: Left: Different views of the aligned trajectories from the SWLS method evaluated with different levels of simulated relative position noise. Right: The distance error of the aligned trajectories for the SWLS method evaluated with different levels of simulated relative position noise.

Evaluation To evaluate the accuracy of the target drone estimated position, we use the ATE. Table 3 shows the ATE of the different method for each level of noise. From this table, it can be observed that the UKF

Table 3: ATE (cm) of different algorithms (EKF, MSCKF, UKF, and SWLS) was evaluated with different levels of simulated relative position noise.

Noise	EKF	MSCKF	UKF	SWLS
0	8.665	8.434	8.119	**8.015**
1	8.830	8.599	**8.243**	8.369
5	10.740	10.163	**9.802**	13.302
10	14.354	**12.231**	12.457	18.732

achieves the best performance when relative position noise is lower than 10cm. Despite the noisy behavior of the UKF, it outclasses all other filters as long as the error stays below 10cm. The second-best filter is the MSCKF, performance wise it is the most accurate filter when the error reaches 10 cm. Unlike the UKF, it generates a smooth trajectory, which is desirable for robotics application. In practice, on outdoor drones, the error is likely be larger than 10cm. In our comparison, the accuracy of the MSCKF seems to be less sensitive to the noise than the other filters: it is the filter with the smallest error increase when the noise level is increased. Overall, we would recommend using the MSCKF, it appears to be more robust to noise than the UKF, and its trajectories are smoother.

In addition to this test, we also deploy our sensor fusion algorithm in another scenario. This set up aims to evaluate the behavior of our system when a loss of the global positioning occurs on the target drone. Initially, the target drone has all of its sensors working and is being tracked by the detector drone. The global positioning sensor of the target drone then fails, and it relies on the position given by the detector drone. After a few seconds, the target drone recovers its global positioning sensor.

Here, the global position measurement is obtained from the motion capture system. To simulate the estimated position of the target drone, we rely on the same technique as previously. The results given in Figure 23 show the position being accurate for the first 30 seconds, then the target drone sensor fails, and we can see the error in position increasing significantly. 20 seconds later, the global position is recovered and the position error quickly decreases. This example demonstrates the ability of our filter to deal with sensor losses, and recovery of said sensor.

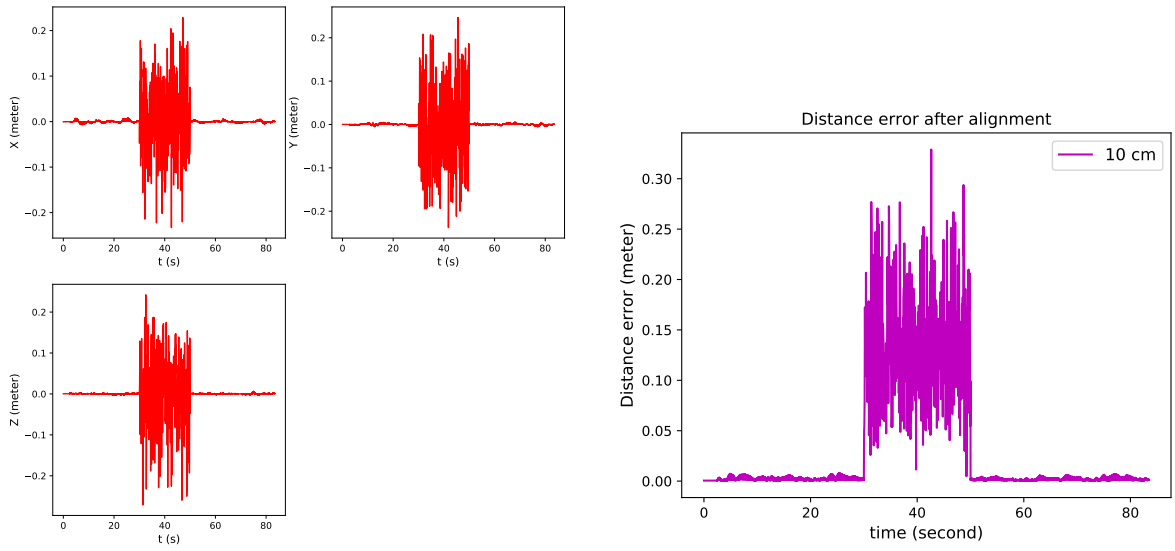

Figure 23: Left: Aligned trajectory error in three direction for EKF method evaluated with 10cm simulated relative position noise. Right: Aligned trajectory distance error for EKF method evaluated with 10cm simulated relative position noise.

5.2.5 Indoor / AeroLab Testing

To validate the perception and collaborative sensor fusion algorithm, we use motion capture system, a very accurate sensor capable of precisely recording the pose of marked objects. The pose acquired by the motion capture system are then compared to the one obtained by the onboard camera and IMU (Figure 24). The setup for this test uses an array of OptiTrack motion capture cameras (12 cameras) to determine the location of the camera and the other UAV. At the same time, the onboard computer of the detector drone records the detections and tracks obtained for the movement of the target drone.

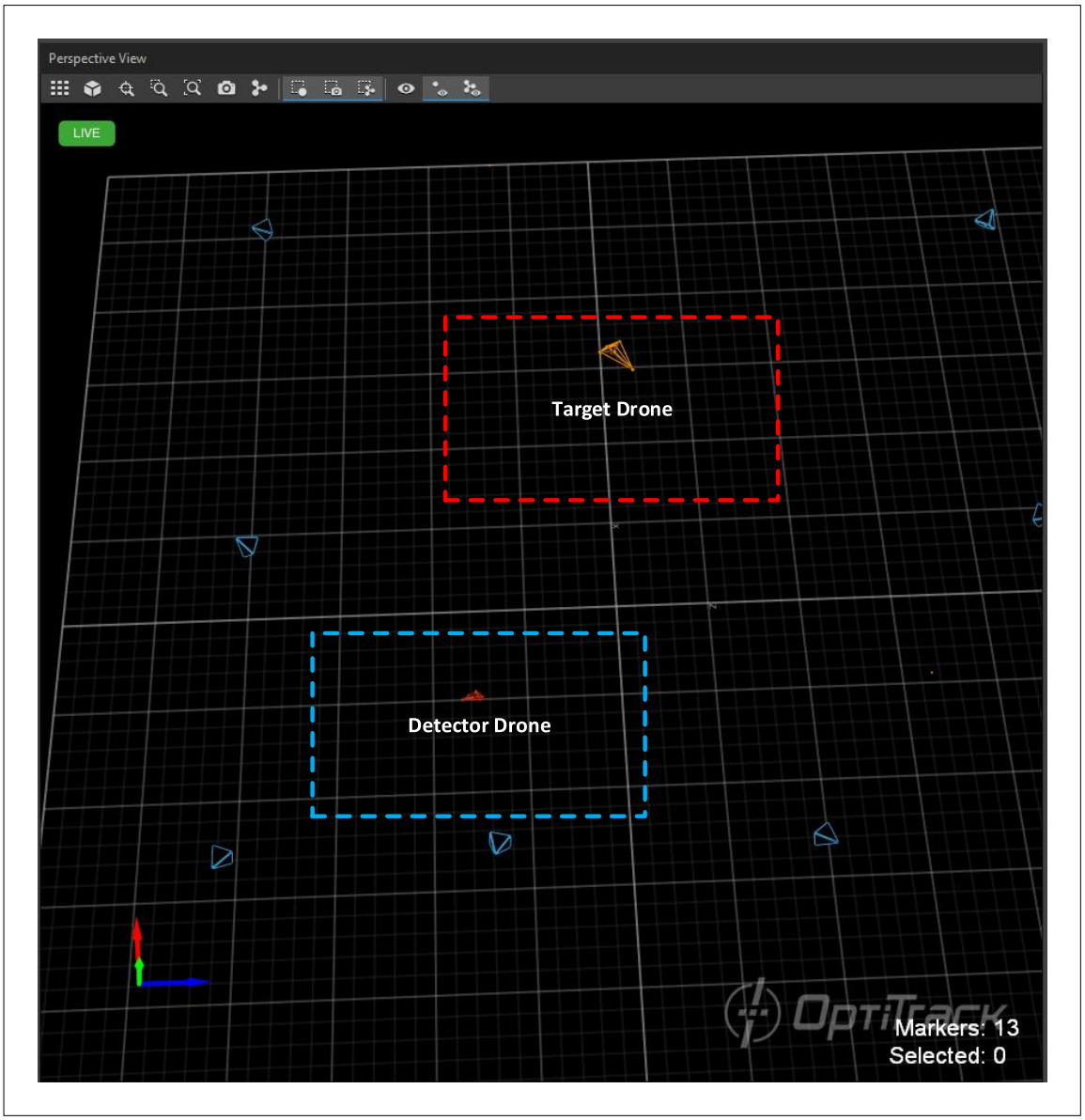

Figure 24: Motion capture system setup for the in-lab testing.

In the scenario depicted in Figure 25, the two drones were flying in the AeroLab. The detector drone is equipped with an onboard Nvidia Jetson Xavier NX and a RealSense D435i camera.

The detection results of the target drone from the proposed drone detection method in the AeroLab are shown in Figure 26.

In an indoor environment, the semantic segmentation results are shown in Figure 27. Color encoding of semantic class categories can be found in [69]. Here, we can see the ceiling in beige, the floor in brown, and

Figure 25: AeroLab for the Indoor Testing.

the walls in grey. The posters on the wall are confusing the network. When they are detected, the network classifies them as signs/signboards while others as posters, which is a legitimate confusion. However, we can see that the network is sometimes having trouble giving the same label to an object. In some of the images, the posters are half classified as half posters, and half signs. This could be due to the network being too small to extract large-scale context and maintain coherence across segmented objects.

We also tested our navigable space segmentation algorithm. Figure 28 shows the algorithm applied in an outdoor environment. On it, we can see the point cloud of the camera as well as the flyable space map. The color of each pixel in the map goes from 0 to 100. A cell with a value of 50 means that the highest point in the cell is at the same height as the drone. A value under 50 (white) is lower than the drone, while a value above 50 is higher than the drone (black). Overall, the algorithm is fast and works well outdoor. However, it does not work indoor because the ceiling is confused as an obstacle. Instead, for indoor environments, we would recommend setting a maximum altitude, or rely on an octomap like representation.

Figure 26: Sample Target Drone Detection Results.

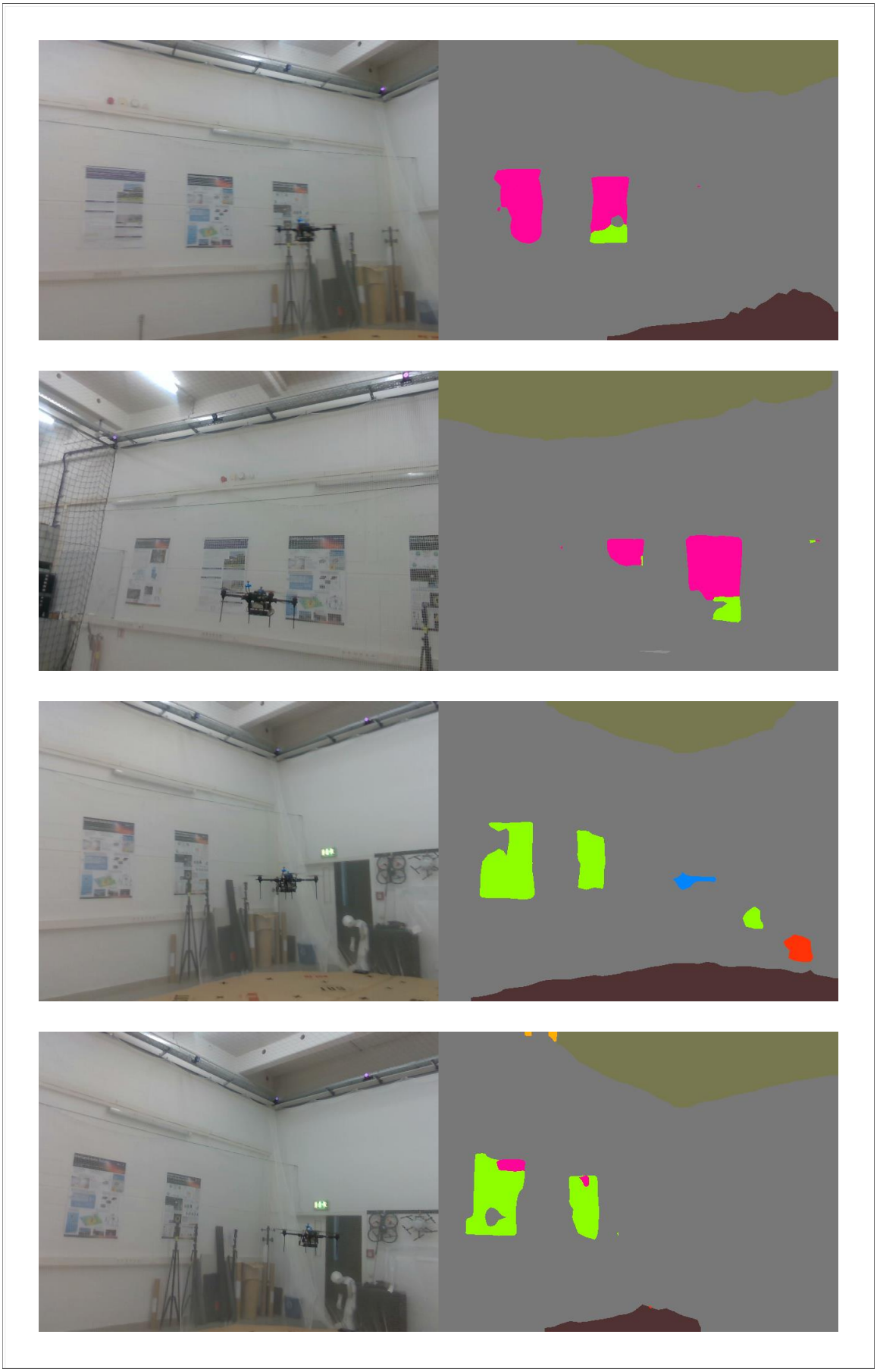

Figure 27: Sample Semantic Segmentation Results.

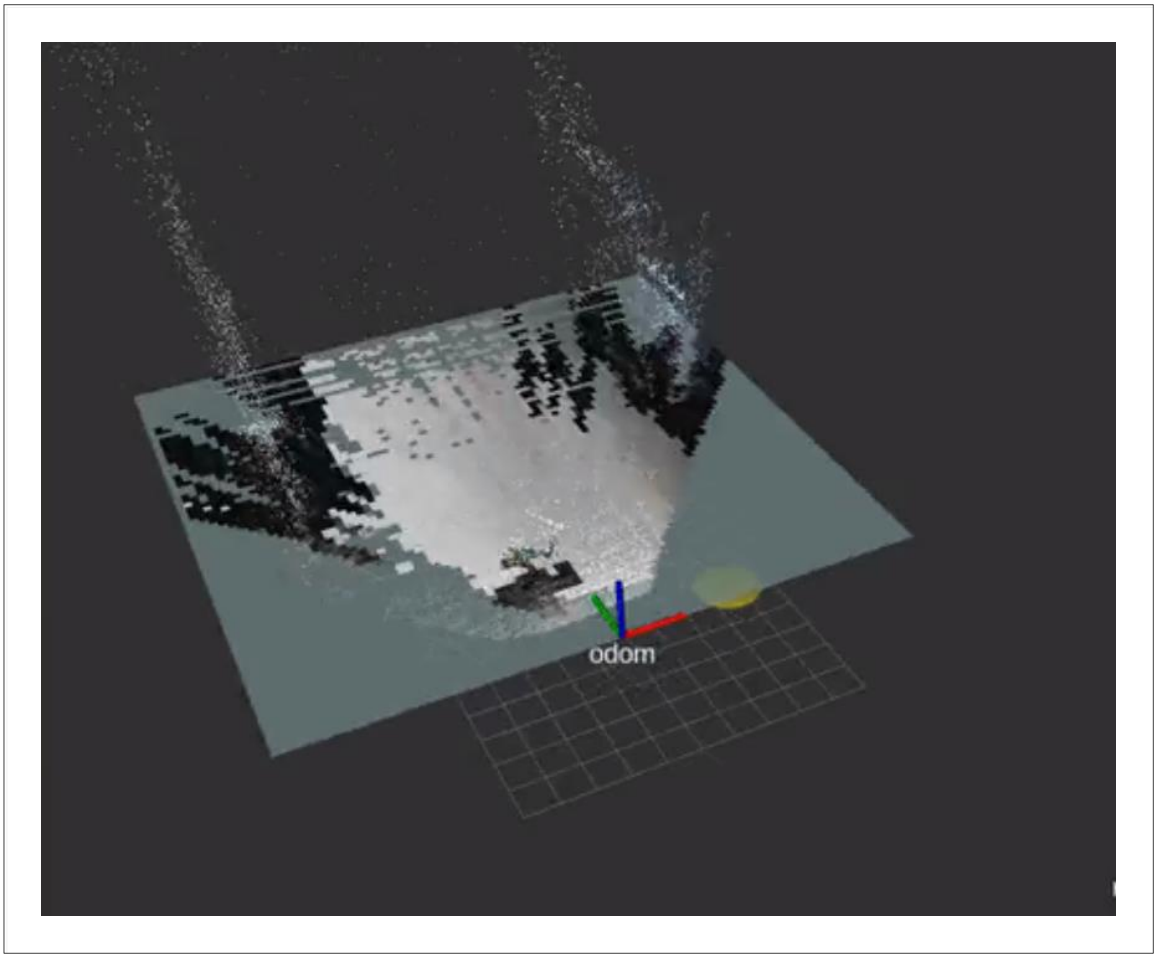

Figure 28: Flyable space segmentation map.

6 Conclusions

This report describes the modules of the perception and collaborative sensor fusion within the SESAME project. It defines their interfaces, input/output data, and the primary methods and algorithms of the modules. The development of these modules is drawn directly from the capabilities described in deliverable D2.1, "Specification of MRS Capabilities." Among all the use cases defined by SESAME, we selected the viticulture use case using drones to demonstrate applicability of our algorithms in complex scenarios. The development was divided into two parts. The first part described a perception, which forms the essential elements of the drone detection, position estimation, and semantic segmentation. The second part detailed the sensor fusion, which is central to the collaborative sensor fusion. Both parts are the description of the perception and collaborative sensor fusion meant to accomplish the missions and tasks in the WP2 (Task 2.3) as part of the SESAME project. They have been tested in simulation, and are in the process of being deployed jointly on real robots in the AeroLab. To conclude, this report serves as a technical guideline for researchers and developers to implement, integrate, deploy, and test core functionalities in perception and collaborative sensor fusion modules along with the execution of this project.

References

[1] A. W. Stroupe and T. Balch, "Mission-relevant collaborative observation and localization," in *Multi-Robot Systems: From Swarms to Intelligent Automata*, pp. 31–40, Springer, 2002.

[2] S. Dong, K. Xu, Q. Zhou, A. Tagliasacchi, S. Xin, M. Nießner, and B. Chen, "Multi-robot collaborative dense scene reconstruction," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 4, pp. 1–16, 2019.

[3] B. D. Gouveia, D. Portugal, D. C. Silva, and L. Marques, "Computation sharing in distributed robotic systems: A case study on slam," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 410–422, 2014.

[4] M. Vasic and A. Martinoli, "A collaborative sensor fusion algorithm for multi-object tracking using a gaussian mixture probability hypothesis density filter," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pp. 491–498, IEEE, 2015.

[5] R. Bajcsy, "Active perception," *Proceedings of the IEEE*, vol. 76, no. 8, pp. 966–1005, 1988.

[6] T. Zwęgliński, "The use of drones in disaster aerial needs reconnaissance and damage assessment–three-dimensional modeling and orthophoto map study," *Sustainability*, vol. 12, no. 15, p. 6080, 2020.

[7] C. Gomez and H. Purdie, "Uav-based photogrammetry and geocomputing for hazards and disaster risk monitoring–a review," *Geoenvironmental Disasters*, vol. 3, no. 1, pp. 1–11, 2016.

[8] M. Hayeri Khyavi, "Rescue network: Using uavs (drones) in earthquake crisis management," *arXiv e-prints*, pp. arXiv–2105, 2021.

[9] W. Blake and I. Burger, "Small drone detection using airborne weather radar," in *2021 IEEE Radar Conference (RadarConf21)*, pp. 1–4, IEEE, 2021.

[10] S. Dogru and L. Marques, "Drone detection using sparse lidar measurements," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 3062–3069, 2022.

[11] Q. Gao, A. Parslow, and M. Tan, "Object motion detection based on perceptual edge tracking," in *Proceedings Second International Workshop on Digital and Computational Video*, pp. 78–85, IEEE, 2001.

[12] J. Lai, L. Mejias, and J. J. Ford, "Airborne vision-based collision-detection system," *Journal of Field Robotics*, vol. 28, no. 2, pp. 137–157, 2011.

[13] S. R. Ganti and Y. Kim, "Implementation of detection and tracking mechanism for small uas," in *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 1254–1260, IEEE, 2016.

[14] J. Li, D. H. Ye, T. Chung, M. Kolsch, J. Wachs, and C. Bouman, "Multi-target detection and tracking from a single camera in unmanned aerial vehicles (uavs)," in *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 4992–4997, IEEE, 2016.

[15] Y. Wu, Y. Sui, and G. Wang, "Vision-based real-time aerial object localization and tracking for uav sensing system," *IEEE Access*, vol. 5, pp. 23969–23978, 2017.

[16] M. Saqib, S. D. Khan, N. Sharma, and M. Blumenstein, "A study on detecting drones using deep convolutional neural networks," in *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pp. 1–5, IEEE, 2017.

[17] C. Aker and S. Kalkan, "Using deep networks for drone detection," in *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pp. 1–6, IEEE, 2017.

[18] D. H. Ye, J. Li, Q. Chen, J. Wachs, and C. Bouman, "Deep learning for moving object detection and tracking from a single camera in unmanned aerial vehicles (uavs)," *Electronic Imaging*, vol. 2018, no. 10, pp. 466–1, 2018.

[19] S. Arunachalam.T, "Flying object detection and classification using deep neural networks," *International Journal of Engineering Trends and Technology*, vol. 67, no. 03, pp. 124–130, 2019.

[20] M. Nalamati, A. Kapoor, M. Saqib, N. Sharma, and M. Blumenstein, "Drone detection in long-range surveillance videos," in *2019 16th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pp. 1–6, IEEE, 2019.

[21] D. K. Behera and A. B. Raj, "Drone detection and classification using deep learning," in *2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS)*, pp. 1012–1016, IEEE, 2020.

[22] Q. Shi and J. Li, "Objects detection of uav for anti-uav based on yolov4," in *2020 IEEE 2nd International Conference on Civil Aviation Safety and Information Technology (ICCASIT*, pp. 1048–1052, IEEE, 2020.

[23] M. W. Ashraf, W. Sultani, and M. Shah, "Dogfight: Detecting drones from drones videos," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7067–7076, 2021.

[24] D. T. W. Xun, Y. L. Lim, and S. Srigrarom, "Drone detection using yolov3 with transfer learning on nvidia jetson tx2," in *2021 Second International Symposium on Instrumentation, Control, Artificial Intelligence, and Robotics (ICA-SYMP)*, pp. 1–6, IEEE, 2021.

[25] B. K. Isaac-Medina, M. Poyser, D. Organisciak, C. G. Willcocks, T. P. Breckon, and H. P. Shum, "Unmanned aerial vehicle visual detection and tracking using deep neural networks: A performance benchmark," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1223–1232, 2021.

[26] S. Singha and B. Aydin, "Automated drone detection using yolov4," *Drones*, vol. 5, no. 3, p. 95, 2021.

[27] H. Liu, K. Fan, Q. Ouyang, and N. Li, "Real-time small drones detection based on pruned yolov4," *Sensors*, vol. 21, no. 10, p. 3374, 2021.

[28] A. Coluccia, A. Fascista, A. Schumann, L. Sommer, A. Dimou, D. Zarpalas, M. Méndez, D. De la Iglesia, I. González, J.-P. Mercier, *et al.*, "Drone vs. bird detection: Deep learning algorithms and results from a grand challenge," *Sensors*, vol. 21, no. 8, p. 2824, 2021.

[29] H. Zhu, F. Meng, J. Cai, and S. Lu, "Beyond pixels: A comprehensive survey from bottom-up to semantic image segmentation and cosegmentation," *Journal of Visual Communication and Image Representation*, vol. 34, pp. 12–27, 2016.

[30] J. Shotton, J. Winn, C. Rother, and A. Criminisi, "Textonboost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context," *International journal of computer vision*, vol. 81, no. 1, pp. 2–23, 2009.

[31] Y. Wei, Z. Wang, and M. Xu, "Road structure refined cnn for road extraction in aerial image," *IEEE Geoscience and Remote Sensing Letters*, vol. 14, no. 5, pp. 709–713, 2017.

[32] M. Tschannen, L. Cavigelli, F. Mentzer, T. Wiatowski, and L. Benini, "Deep structured features for semantic segmentation," in *2017 25th European Signal Processing Conference (EUSIPCO)*, pp. 61–65, IEEE, 2017.

[33] K. Chen, K. Fu, X. Sun, M. Weinmann, S. Hinz, B. Jutzi, and M. Weinmann, "Deep semantic segmentation of aerial imagery based on multi-modal data," in *IGARSS 2018-2018 IEEE International Geoscience and Remote Sensing Symposium*, pp. 6219–6222, IEEE, 2018.

[34] S. Wei, S. Ji, and M. Lu, "Toward automatic building footprint delineation from aerial images using cnn and regularization," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 58, no. 3, pp. 2178–2189, 2019.

[35] B. Zhou, H. Zhao, X. Puig, T. Xiao, S. Fidler, A. Barriuso, and A. Torralba, "Semantic understanding of scenes through the ade20k dataset," *International Journal of Computer Vision*, vol. 127, no. 3, pp. 302–321, 2019.

[36] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba, "Semantic understanding of scenes through the ade20k dataset," *arXiv preprint arXiv:1608.05442*, 2016.

[37] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba, "Scene parsing through ade20k dataset," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[38] C. Chen, H. Zhu, M. Li, and S. You, "A review of visual-inertial simultaneous localization and mapping from filtering-based and optimization-based perspectives," *Robotics*, vol. 7, no. 3, p. 45, 2018.

[39] S. Lynen, M. W. Achtelik, S. Weiss, M. Chli, and R. Siegwart, "A robust and modular multi-sensor fusion approach applied to mav navigation," in *2013 IEEE/RSJ international conference on intelligent robots and systems*, pp. 3923–3929, IEEE, 2013.

[40] S. Weiss, M. W. Achtelik, S. Lynen, M. Chli, and R. Siegwart, "Real-time onboard visual-inertial state estimation and self-calibration of mavs in unknown environments," in *2012 IEEE international conference on robotics and automation*, pp. 957–964, IEEE, 2012.

[41] S. Shen, Y. Mulgaonkar, N. Michael, and V. Kumar, "Multi-sensor fusion for robust autonomous flight in indoor and outdoor environments with a rotorcraft mav," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4974–4981, IEEE, 2014.

[42] T. Qin, J. Pan, S. Cao, and S. Shen, "A general optimization-based framework for local odometry estimation with multiple sensors," *arXiv preprint arXiv:1901.03638*, 2019.

[43] T. Qin, S. Cao, J. Pan, and S. Shen, "A general optimization-based framework for global pose estimation with multiple sensors," *arXiv preprint arXiv:1901.03642*, 2019.

[44] G. Huang, "Visual-inertial navigation: A concise review," in *2019 international conference on robotics and automation (ICRA)*, pp. 9572–9582, IEEE, 2019.

[45] P. Geneva, K. Eckenhoff, W. Lee, Y. Yang, and G. Huang, "Openvins: A research platform for visual-inertial estimation," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4666–4672, IEEE, 2020.

[46] W. Xu, Y. Cai, D. He, J. Lin, and F. Zhang, "Fast-lio2: Fast direct lidar-inertial odometry," *IEEE Transactions on Robotics*, 2022.

[47] R. Jung and S. Weiss, "Scalable recursive distributed collaborative state estimation for aided inertial navigation," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1896–1902, IEEE, 2021.

[48] R. Jung, C. Brommer, and S. Weiss, "Decentralized collaborative state estimation for aided inertial navigation," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4673–4679, IEEE, 2020.

[49] R. Jung and S. Weiss, "Modular multi-sensor fusion: A collaborative state estimation perspective," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 6891–6898, 2021.

[50] P. Zhu, Y. Yang, W. Ren, and G. Huang, "Cooperative visual-inertial odometry," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 13135–13141, IEEE, 2021.

[51] P. Zhu, P. Geneva, W. Ren, and G. Huang, "Distributed visual-inertial cooperative localization," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 8714–8721, IEEE, 2021.

[52] H. Xu, Y. Zhang, B. Zhou, L. Wang, X. Yao, G. Meng, and S. Shen, "Omni-swarm: A decentralized omnidirectional visual-inertial-uwb state estimation system for aerial swarm," *arXiv preprint arXiv:2103.04131*, 2021.

[53] J. Sola, "Quaternion kinematics for the error-state kalman filter," *arXiv preprint arXiv:1711.02508*, 2017.

[54] YOLOv5, "Yolov5," 2022. Last accessed April 2022.

[55] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[56] M. C. V. team, "Semantic segmentation for ros in pytorch," 2022. Last accessed April 2022.

[57] M. Larsson, E. Stenborg, L. Hammarstrand, M. Pollefeys, T. Sattler, and F. Kahl, "A cross-season correspondence dataset for robust semantic segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9532–9542, 2019.

[58] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Niessner, M. Savva, S. Song, A. Zeng, and Y. Zhang, "Matterport3d: Learning from rgb-d data in indoor environments," *arXiv preprint arXiv:1709.06158*, 2017.

[59] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, "Scannet: Richly-annotated 3d reconstructions of indoor scenes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5828–5839, 2017.

[60] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[61] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, 2013. Software available at `https://octomap.github.io`.

[62] K. Sun, K. Mohta, B. Pfrommer, M. Watterson, S. Liu, Y. Mulgaonkar, C. J. Taylor, and V. Kumar, "Robust stereo visual inertial odometry for fast autonomous flight," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 965–972, 2018.

[63] C. Hertzberg, R. Wagner, U. Frese, and L. Schröder, "Integrating generic sensor fusion algorithms with sound state representations through encapsulation of manifolds," *Information Fusion*, vol. 14, no. 1, pp. 57–77, 2013.

[64] T. Qin, P. Li, and S. Shen, "Vins-mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018.

[65] Pixhawk, "Pixhawk 4," 2022. Last accessed April 2022.

[66] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, Japan, 2009.

[67] XTDrone, "Xtdrone," 2022. Last accessed April 2022.

[68] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, vol. 35, no. 10, pp. 1157–1163, 2016.

[69] M. C. V. team, "Color coding semantic segmentation classes," 2022. Last accessed April 2022.