ECOLE
POLYTECHNIQUE
DE BRUXELLES

ULB

UNIVERSITÉ LIBRE DE BRUXELLES

Sensor fusion in swarm robotics
Enhancing perception capabilities of swarm robots

Mémoire présenté en vue de l'obtention du diplôme
d'Ingénieur Civil en Informatique à finalité spécialisée

Saïd Belaroussi

Director
Prof. Mauro Birattari

Supervisor
Ir. Miquel Kegeleirs

Service
IRIDIA

Academic year
2023 - 2024

ULB UNIVERSITÉ LIBRE DE BRUXELLES

ULB UNIVERSITÉ LIBRE DE BRUXELLES

Exemplaire à apposer sur le mémoire ou travail de fin d'études,
au verso de la première page de couverture.

Réservé au secrétariat :	**Mémoire réussi***	**OUI**
		NON

CONSULTATION DU MEMOIRE/TRAVAIL DE FIN D'ETUDES

Je soussigné

NOM : BELAROUSSI

PRENOM : SAÏD

TITRE du travail : SENSOR FUSION IN SWARM ROBOTICS
ENHANCING PERCEPTION CAPABILITIES OF SWARM ROBOTS

AUTORISE*

~~**REFUSE***~~

la consultation du présent mémoire/travail de fin d'études par les utilisateurs des bibliothèques de l'Université libre de Bruxelles.

Si la consultation est autorisée, le soussigné concède par la présente à l'Université libre de Bruxelles, pour toute la durée légale de protection de l'œuvre, une licence gratuite et non exclusive de reproduction et de communication au public de son œuvre précisée ci-dessus, sur supports graphiques ou électroniques, afin d'en permettre la consultation par les utilisateurs des bibliothèques de l'ULB et d'autres institutions dans les limites du prêt inter-bibliothèques.

Fait en deux exemplaires, **Bruxelles, le** 18/08/2024

Signature

Formulaire version 04/2015

** Biffer la mention inutile*

** Biffer la mention inutile*

Title: Sensor fusion in swarm robotics
Author: Saïd Belaroussi
Master's degree: Master en ingénieur civil en informatique, à finalité spécialisée
Academic year: 2023 - 2024

Abstract

The sensor fusion topic comes naturally with the recent development of a new robotic platform at the IRIDIA laboratory that utilizes a new set of sensors. Sensor fusion has the potential to enhance the perception capabilities of each robot and open the way to new possibilities in swarm robotics experiments.

The main objective of this Master's thesis is to develop a sensor fusion technique, specially designed for this new robotic platform and able to merge data from different sensors to increase the accuracy of the robot in detecting and estimating the position of other robots in the swarm, and obstacles.

The camera and LiDAR of the robot are selected to be part of the sensor fusion scheme after analysis of the different perception sensors, and a data fusion method is implemented based on the literature review and selected sensors. The algorithm combines the Hungarian algorithm to associate data between the sensors and the Kalman filter to fuse the data. Three variants of this algorithm are based on different LiDAR detection methods: a standalone Euclidean clustering method, Euclidean clustering based on detected bounding boxes from the camera, and a fully assisted method using the position estimate from the camera. The methods are evaluated in three different scenarios to assess their performance.

The experiments showed that the standalone Euclidean clustering LiDAR method is the most suited LiDAR detection method for the sensor fusion algorithm. The experimental results were discussed and several adjustments were made to improve this variant of the fusion algorithm. The algorithm also showed great resiliency in scenarios where one or both sensors are faulty.

The sensor fusion algorithm developed in this work proves efficient at merging data from the camera and the LiDAR. The algorithm has better results than both sensors taken individually and the ability of the robot to sense other robots in the swarm is demonstrated.

Keywords: Sensor fusion, Swarm robotics, Kalman filter, Camera, 2D LiDAR

Acknowledgements

I would like to thank my promoter, Professor Mauro Birattari for proposing the subject of this Master's thesis, "Sensor fusion in swarm robotics" and for trusting me to research this topic that deeply interested me.

I would like also to thank my supervisor, Miquel Kegeleirs, for his continuous guidance throughout the academic year. The many discussions we had and his valuable feedback helped me broaden my research, find solutions, and advance.

Many thanks to Jeanne Szpirer and Edwige Loems, their collaboration permitted me to go further in the experiments proposed in this work.

Finally, thanks to everyone who supported me and helped me during difficult times throughout this thesis.

Contents

Chapter 1

Introduction

The idea of this Master's thesis comes from the development of a new robotic platform at IRIDIA suited for swarm robotics experiments. The new robots have multiple sensors that capture different data and overlapping data. The sensors of robots used in swarm robotics experiments are usually very limited in precision and coverage [1].

Sensor fusion is worth studying in any system equipped with multiple sensors, as it improves the overall ability of the system to sense its environment. Since the sensors are already there and providing their own data, their combination comes essentially for "free" or, more precisely, at the cost of computing hardware and power, which is minimal compared to the cost of the sensors. Plus, most of the time, the computing hardware is already there to collect data from the different sensors.

Sensor fusion is a broad term that encompasses a lot of techniques and applications. Sensor fusion in its widest definition can refer to any method that consists of combining data coming from several sensors, either to increase the global accuracy of some system measuring some state or associate measurements of different states or both [2] [3].

The goal of this work is to investigate how sensor fusion can be designed for such a robotic platform to enhance the perception capabilities of the robots. I will try to reach this goal by developing a sensor fusion method based on the state of the art and analysis of the individual capabilities of the present sensors and their potential in a sensor fusion scheme.

1.1 Main contributions

I focused mainly on developing and implementing sensor fusion techniques between a camera device composed of an RGB camera and a stereo pair, and 2D 360° LiDAR to improve each robot's ability to detect other robots in the swarm and estimate their positions in real-time. These techniques can detect and compute the relative position of not only the robots but any kind of object of interest or obstacles if the camera algorithm is trained to detect them.

All the software was developed as a ROS package since many components of the robot already use ROS and it is the best way to integrate this work on the robots. The package is available on GitHub[1].

[1] `https://github.com/Said-belaroussi/mercator_sensor_fusion_ros_pkg`

ROS is an open-source framework designed for robotics and autonomous systems development. It includes a huge set of tools and libraries that make it easier to create and maintain software components for robotics. In practice, ROS provides a publisher/subscriber messaging system to permit communication between different software modules.

I used the publisher/subscriber system to its fullest in this work to separate the software into multiple independent components that can be activated or deactivated easily depending on usage.

Concretely, I developed two sensor fusion techniques with the first one using 3 variants of LiDAR detection:

- High-level and modular fusion: Each sensor performs detection and position estimation of the robots separately before fusing the data using a Kalman Filter and Hungarian algorithm. This method is not specific to the sensors used on the robots and can be used with any number of sensors. Its performance was evaluated when coupled with 3 different LiDAR detection methods:
  - Standalone LiDAR detection: The LiDAR performs detection by itself using Euclidean clustering and then estimates the positions of the detected clusters.
  - Partially assisted LiDAR detection: The LiDAR uses the detections performed by the camera to know in which directions to look for targets and then estimates their positions.
  - Fully assisted LiDAR detection: The LiDAR uses all the data provided by the camera including its position estimation of the targets. The LiDAR will search for the closest position to each target in its readings within a certain radius.
- Complementary fusion: The camera is used to detect the robots, while only the LiDAR provides depth data to estimate position. This method has the advantage of saving battery power since the stereo pair of the camera is not used. It also provides a baseline to compare the improvement in accuracy for the previous method.

Additionally, I selected a method to calibrate the stereo camera and the LiDAR together to ensure the alignment of data from the sensors and increase the accuracy of the fusion methods.

I developed a protocol to evaluate the accuracy of these techniques based on similar works and created a dataset for this purpose. This dataset can be used to:

- Evaluate future techniques and compare them to the ones proposed in this work.
- Perform regression testing to ensure any modifications in the implemented techniques do not lower their quality.

I did assess the performance of each sensor fusion method in real experiments, discussed the results, and implemented improvements based on them.

1.2 Outline

This thesis is structured as follows:

- Chapter 2 is dedicated to the literature review of swarm robotics and sensor fusion in general.

- Chapter 3 is about the introduction of the robotic platform and analysis of the different perception sensors that are present.
- Chapter 4 contains all the methodology used and implemented in this work, including the core of the fusion algorithm, and based on it, the camera and LiDAR detection methods.
- Chapter 5 presents and summarizes the different sensor fusion techniques developed in this work.
- Chapter 6 is dedicated to all the experiments done including, the setup, evaluation metrics, results discussion, and improvements.
- Chapter 7 offers a conclusion to this Master's thesis and possible future developments.

Chapter 2

Related work

Sensor fusion techniques have gained a lot of popularity in the Automotive industry in recent years and are a vector of advancements in Advanced Driver-Assistance Systems (ADAS) and autonomous driving because of the need to increase vehicles' perception capabilities of their environment [3]. This is done initially by using different kinds of sensors that complement each other and can detect other vehicles, obstacles, road markings, traffic signs, etc. The data from the individual sensors is relevant in itself but additional information is gained by combining it.

In ADAS and self-driving, the most used sensors for perception are camera, LiDAR (Light Detection and Ranging), and Radar (Radio Detection and Ranging). The camera is used to detect and classify objects, LiDAR provides accurate depth sensing that can be used to estimate the distance to detected objects, and Radar can sense objects and their velocities [4]. By combining the data from the three sensors, not only is their complementarity great and permits the detection, classification, and measurement of distance to objects, and their velocities, but also it profits from redundant information. For example, both the camera and radar are detecting objects (not necessarily the same detections but can be overlapping) and both LiDAR and RADAR provide positions of the detected objects.

The sensors are not limited to those described above. The camera can be a mono or stereo camera (which adds depth capability), and the LiDAR can be 3D or 2D, 360° Field of View or less. There are also proximity sensors which measure distance to obstacles but are limited in Field of View, Radio Frequency Identification which provides position and identification using a physical RFID tag [5], ultrasonic sensors which rely on the speed of sound to measure distances [6], and many others.

2.1 Sensors in swarm robotics

Before discussing the usage of sensors in swarm robotics, let's first see what differentiates a swarm of robots from a simple group of robots or a single robot. In [7], constraints that a swarm of robots has to respect are listed as follows:

- Robots are autonomous.
- Robots are situated in the environment and can act to modify it.
- Robots' sensing and communication capabilities are local.
- Robots do not have access to centralized control and/or to global knowledge.

- Robots cooperate to tackle a given task.

These constraints are not there to only constrain the robot system but to guarantee some nice properties such as:

- Not having a single point of failure since all robots are autonomous, the swarm will continue its mission even when losing one or more robots [8].
- Ability to do missions in areas not covered by GPS (or any other form of global knowledge) [9].
- Losses of robots due to failures are less costly since the total cost of the robot system is split between all the robots, compared to a system with a single robot.

Many advancements were made recently in the swarm robotics field and the sensors typically used for robots in swarm experiments are inexpensive and of lower quality than those for standalone robots [10] [11] [12]. This is not a problem but, in fact, an advantage. Of course, robust and high-quality sensors are always preferred, but the trade-off is their higher price. One of the advantages that swarm design brings to robotics is enabling several robots of lower cost to work together towards a shared objective. While the same objective could likely be accomplished by a single high-cost robot, it comes with a higher risk of mission failure and/or damage and loss of the robot [13].

At first glance, sensor fusion in swarm robotics appears to be an unusual combination of topics because, in swarm robotics, it is accepted that robots are individually less capable of sensing their environment than they are as a collective swarm [14]. Thus, the focus should be on enhancing swarm techniques rather than individual robot sensing through sensor fusion. However, sensor fusion between sensors on the same robot shares a similar philosophy with swarm robotics.

Regarding perception in swarm robotics, the goal is to enhance the quality of what the swarm perceives locally [15] [16]. Improving the perception accuracy of individual robots will directly enhance the local perception of the swarm. Moreover, sensor fusion techniques developed in this context can also be applied to data from different robots, as local communication between robots is possible.

2.2 Classification of sensor fusion methods

Sensor fusion methods are numerous and depend on the sensors used, available computational power, and objectives. Nonetheless, the sensor fusion methods used to increase the accuracy in detecting objects can be classified into three main classes [17]:

- Low-level fusion: fusion is performed at the lowest possible level of processing of the data. The raw data from the sensors is fused and detection/classification of objects is done on the fused data. Doing fusion at this level has the advantage of preserving the maximum of information possible. Machine learning techniques are often found in this class because deep neural networks are really suited to extract the maximum of information from raw data.
- Middle-level fusion: fusion is rather performed after some preprocessing of the data coming from each sensor. The preprocessing can go from simple denoising to high-level feature extraction. it has the advantage of using less computational power, and less data has to be communicated if the sensor can do the preprocessing with its hardware.

- High-level fusion: each sensor data is preprocessed separately to generate a list of detected objects. Depending on the sensor, they can include additional information like the bounding box of the detection, position in 2D/3D coordinate frame, velocity, orientation, etc. The fusion is done on the provided object lists to produce a more accurate list of the detected objects. It can go from simply joining all the objects detected in the lists if the sensors give strictly complementary information (no overlapping in their environment perception) to the increase in the quality of the detections, by better estimation of the confidence in the detection, more accurate position, velocity, etc (the sensors overlap their environment perception).

2.3 Sensor fusion in robotics for localization

For the localization problem of mobile robots, the extended Kalman filter (EKF) is used as the basis of most of the algorithms because of its ability to fuse data for nonlinear combinations of sensors and have equivalent accuracy in real-world scenarios to the Unscented Kalman Filter while being less computationally hungry, allowing to save battery power [18].

In [19], the localization problem for an autonomous dam surveillance robot where the robot needs accurate localization to plan its surveillance route indoors and outdoors and ensure the dam safety. To improve the localization, a first extended Kalman filter loop is used to fuse wheel encoders, IMU, and LiDAR odometry, the result of this fusion is then used in a second extended Kalman filter loop to fuse with GNSS data.

In [20], the data from IMU, wheel encoders, visual odometry, and laser-based mapping are fused to provide reliable localization for robots in Urban Search and Rescue contexts. The fusion is also done with an extended Kalman filter loop.

In [21], the data from an indoor static radar infrastructure, ultrasonic sensor, and robot odometry are fused using an extended Kalman filter to provide absolute 2D localization of the robot.

In [22], extended Kalman filter is used to improve simultaneous localization and mapping (SLAM) by fusing first, the data from IMU and LiDAR SLAM in a first EKF loop and then with Stereo SLAM data in a second EKF loop.

2.4 Sensor fusion for object detection

Object detection does not require tracking, but it becomes necessary in the context of sensor fusion when several sensors can detect objects. The goal of tracking is to correct the estimation of the state of an object based on observations from multiple sensors. When tracking multiple objects, data association is another necessary step to associate new observations of different objects with previous observations and more than that, to associate detected objects from different sensors together.

In [23], a Mono Camera and 3D LiDAR are mounted on a vehicle and the sensor fusion technique is based only on the complementarity of the two sensors. Before anything, a calibration step is done to align correctly the perception of the camera and LiDAR. A neural network is trained to detect different road users and traffic signs used on the images from the camera and produces detections in the form of bounding boxes having a certain height, width, and type. The data fusion step is based on the previously done calibration to associate the LiDAR point cloud to the 2D camera image which makes it

possible to determine the distance to any detected object from the vehicle as shown in Figure 2.1. This is a good example of how to align sensors measuring different variables of the environment and combine them.

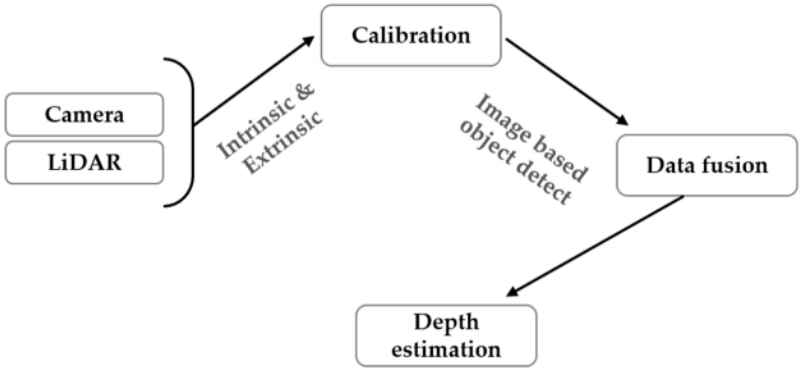


Figure 2.1: LiDAR and Camera Fusion Approach for Object Distance Estimation in Self-Driving Vehicles [23]

In [6], A neural network is trained and used to fuse data from the stereo camera, 2D 360° LiDAR, and sonars. The goal of the fusion is to provide a more accurate distance to obstacles estimation for mobile robots, and also take advantage of the ability of some sensors to detect materials that others cannot. For example, LiDAR is bad for detecting glass because the emitted light beam of the LiDAR refracts and does not return to it, while the sonar has no difficulty detecting refractive materials.

In [24], a machine learning approach is used to fuse data from the camera, LiDAR, and RADAR to detect and classify 3D objects in real-time. A region-based convolutional neural network is trained taking as input a bird's eye view of the 3D LiDAR data and the RADAR data to produce depth data around the vehicle which is then combined with the bounding boxes produced by YOLOv3 convolutional neural network processing images from the camera.

In [25], a Siamese neural network is trained and used to fuse images from the camera and depth maps from 3D LiDAR to obtain the positions of the detected objects.

In [26], a Particle filter is used to fuse pedestrian detections from 2D Camera and 3D LiDAR based on a proposed motion model and data association model.

In [27], a camera and RADAR are placed at a Traffic intersection to detect and track vehicles. The camera detects vehicles using YOLOv4 and provides their positions. RADAR provides positions and velocities. Kalman Filter is used for sensor fusion and the Hungarian algorithm is used for data association between the sensors.

In [28], LiDAR and RADAR are fused using an extended Kalman filter to track the position and velocity of a moving object. Only one object is tracked, so no data association algorithm was used.

In [29], Kalman Filter is used to fuse data from LiDAR and RADAR to perform more accurate vehicle detections and tracking based on a constant turn rate and velocity dynamic model.

In [17], a high-level and modular fusion is described in detail, it has the advantage of being able to deal with any number of sensors and is resilient to sensor failure. Each sensor

needs to produce its own object track list and the different lists are fused to produce a final and more accurate list of the tracked objects. The algorithm uses the Unscented Kalman Filter for data fusion and the Hungarian algorithm for data association.

In [30], a camera, 3D LiDAR, and RADAR are mounted on a vehicle and were fused to accurately detect, estimate position, and track other vehicles. Camera detection was performed using YOLOv3 neural network coupled with RADAR to estimate the positions. LiDAR performed detection and position estimation. The Data was fused using the extended Kalman filter, and the association between data coming from the sensors and currently tracked vehicles was done using the Hungarian algorithm.

2.5 Sensor fusion for higher accuracy depth sensing

In [31], depth maps from 3D LiDAR and stereo camera are fused. The LiDAR has the advantage of having accurate measurements but sparse data, while the camera has a much higher resolution but loses in accuracy. The fusion scheme keeps all the values from LiDAR and interpolates the values between, taking the camera depth map as a reference and computing the missing values with a weighted sum of the values from LiDAR and camera, using some well-found fitting weights.

In [32], depth maps from sparse 3D LiDAR and dense stereo camera are also fused into a more accurate dense depth map. Similarly to [31], all values from LiDAR are kept and the missing values are interpolated from LiDAR and camera depth maps via a probabilistic approach.

In [33], the fusion between sparse 3D LiDAR and dense stereo camera depth maps is done using a deep convolutional neural network which has the advantage of having a compact architecture that can be run on mobile robots. KITTI dataset [34] was used to train and validate the model.

2.6 Summary

After going through all the previously cited applications of sensor fusion techniques in different domains and with different sensors, it becomes clear that the choice of the most suitable technique depends heavily on the kind of sensors we have, their accuracy, and how they can complement each other.

Machine learning methods are mostly suited to fuse raw data at low levels with little preprocessing or no preprocessing at all to preserve all of the features in the data.

High-level fusion methods are based on a combination of state estimators and data association algorithms. Kalman filter, extended Kalman filter, unscented Kalman filter, and particle filter are commonly used as state estimator methods. The Hungarian algorithm is commonly used as a data association method in object detection tasks.

In a following section, the different sensors of the Mercator robot are listed along with their characteristics to discuss which sensor fusion techniques would be worth considering.

Chapter 3

Robotic platform sensors analysis

The robotic platform used for this Master's thesis is the Mercator robot [35]. The Mercator is the newly developed robotic platform at IRIDIA to run swarm robotics experiments. The robot has a lot of actuators and sensors.

In this chapter, I analyze the sensors that are present on the Mercator robot, but not all the sensors are taken into account for this analysis, only the sensors used for environment perception. The goal is to determine whether including a sensor in a sensor fusion scheme would enhance the ability of the robot to detect other robots in the swarm or not.

3.1 Camera

The camera is the Oak-D W stereo camera [36], shown in Figure 3.1, which is much more than a simple camera; it is a complete computer vision module. It consists of a high-resolution RGB camera (12 MP) placed at the center of the module and a stereo pair used for depth sensing with a resolution of 1280x800 and a range between 0.2 m and 10 m. Additionally, these sensors are linked to the Robotics Vision Core 2 (RVC2), which is a System-on-Chip capable of efficiently running various neural network architectures with dedicated hardware.

Figure 3.1: Oak-D Wide camera[1]

In previous work, I conducted a comparison of several edge AI options for computer vision, and the Oak-D W has proved to be the best choice for the Mercator robot due to its efficiency and low power consumption. Additionally, a YOLO neural network was trained to detect the robots, but it was for the first version of the Mercator. Since the experiments in this Master's thesis are done with a new prototype that is visually

[1] `https://shop.luxonis.com/products/oak-d-w?variant=43905772519647`

completely different from the previous one, the neural network needs to be retrained to detect the new prototype. The protocol used to collect the dataset, annotations, and training will be described in a following section.

The camera module is actually two sensors instead of one: the RGB camera used for robot detection and the stereo pair providing a depth map. Since the alignment of these two sensors is done at the firmware level, as well as the computation of the position of the detected objects [37], it can also be considered a single sensor detecting the robots and giving their positions in the camera frame. In the next sections, the camera will sometimes be considered a single sensor or two sensors depending on the fusion technique.

3.2 LiDAR

A 2D 360° Light Detection and Ranging (LiDAR) sensor, shown in Figure 3.2, is mounted on top of the robot. This sensor provides very accurate distance estimation around the entire robot in the xy-plane.

- Min range: 0.12 m
- Max range: 10 m
- Angle resolution: Between 0.43° and 0.86° (depending on frequency, 6 Hz to 12 Hz). Typically runs with 0.50° resolution at 7 Hz frequency, corresponding to 720 distance values around the entire robot.

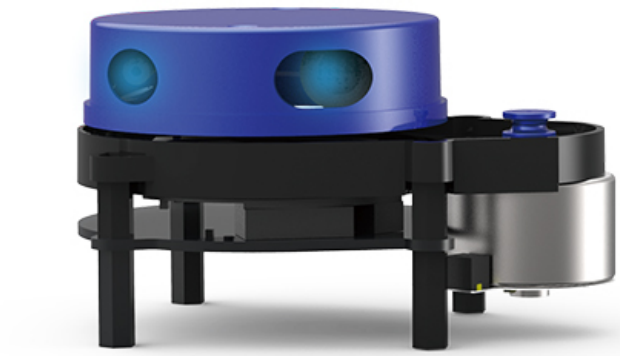


Figure 3.2: YDLIDAR X4[2]

The distance data can be fused by simply looking at the corresponding angles for a robot detected by the RGB camera and computing the position of that robot. Since all robots

[2] https://www.ydlidar.com/products/view/5.html

are the same, they all have their LiDAR at the same height. So, the distance to other robots is, in reality, the distance to their LiDAR (which is on top and centered on the robot). This means that position values will always be consistent regardless of the detected robots' orientation. This fusion method takes advantage of the complementary nature of detections done by the RGB camera and the distance values from the LiDAR.

The LiDAR data can also be fused using a low-level fusion technique by combining the raw distance data with any other sensor providing the same kind of data if there is overlap in their field of view (FoV). Regarding the fusion with the camera, there is a line of depth values in the depth map generated by the stereo camera that overlaps with 95° of FoV from the LiDAR.

The LiDAR can perform detections of objects and robots in addition to estimating their positions using clustering methods [38]. Object detection at the sensor level is necessary in high-level fusion methods where each sensor needs to produce its own list of detected objects before fusion.

3.3 Proximity sensors

Eight proximity sensors [39] are mounted around the robot, with the majority located at the front. These sensors are used only for obstacle avoidance and provide distance estimation to obstacles with a maximum range of 1.10 m within a limited field of view (20°).

Since each proximity sensor provides distance estimation at a known angle, they could theoretically be used to estimate the position of robots detected by the camera. However, due to the very limited range and low resolution (one depth value for a 20° field of view), the proximity sensors are not considered for sensor fusion at this moment.

3.4 RGB ground color sensor

An RGB ground color sensor that detects the color of the floor that is just below the robot. This information can be used individually and will not be included in the sensor fusion schemes in this Master's thesis because it does not add directly any value to the data coming from the other sensors to help the robot detect other robots or estimate their positions.

3.5 Ambient light sensor

A sensor that senses the intensity of ambient light. Not considered for sensor fusion for the same reason as the RGB Ground Color Sensor.

3.6 Summary

The sensors that will be used for sensor fusion are the camera and the LiDAR.

- The camera can provide detections of the robots and the depth map of the environment in front of the robot. These two types of data will either be considered separately or combined by estimating the position of the detected robot using the depth information.

- The LiDAR provides accurate depth information in the 2D xy plane all around the robot. The LiDAR can also perform detection of the robots using clustering methods.

Chapter 4

Methodology

4.1 Data fusion method

The literature review showed that several high-level sensor fusion methods exist and the best choice depends on the goal of the sensor fusion and the sensors used. Methods based on Particle filter, extended Kalman filter, and Unscented Kalman filter are all well-fitted for sensor fusion in non-linear dynamic systems.

In the case of this thesis, the fusion is performed on data from a stereo camera and LiDAR, where each sensor provides positions for detected robots. Estimating the position for a moving robot where the measurements are also positions is a linear problem, as I show in the following section. This means that the linear Kalman filter is the most suitable variant in the case of Kalman filtering.

Machine learning methods are also often used and can estimate linear or non-linear dynamic systems. Those methods have several disadvantages:

- Need for dedicated hardware to run inference.
- Optimal architectures of neural networks depend on the type of sensors used and their quality and there is no guarantee that the same architecture would bring the same accuracy with different sensors [6].
- Sensitivity to the training dataset is a recurrent problem with deep neural networks. It can be mitigated by providing a huge and diversified dataset.

Since the position data of the robots from the stereo camera and LiDAR evolves based on a linear dynamic system and the Kalman filter is the best linear estimator for such problems [40]. I chose to implement the linear Kalman filter, which is often simply called Kalman filter.

4.1.1 Kalman filter: General explanation

The Kalman filter [41] is used as the core of the high-level sensor fusion algorithm proposed in this Master's thesis. The Kalman filter is often used as a state estimator of some process that has a known evolution model over time with noisy measurements. Besides the initialization of the different parameters of the algorithm, a Kalman filter consists of 2 main steps: a prediction step and an update step [42].

State transition equation

The state of the system is represented by a state vector $\mathbf{x}_k$ and is assumed to evolve according to the following linear dynamic model, at each time step k:

$$\mathbf{x}_k = \mathbf{F}_k \mathbf{x}_{k-1} + \mathbf{w}_k$$

Where:

- $\mathbf{F}_k$ is the state transition matrix that describes how the state evolves from time $k-1$ to k.
- $\mathbf{w}_k$ is the assumed process noise with covariance matrix $\mathbf{Q}_k$.

State relation to the measurements

The system produces a measurement $\mathbf{z}_k$ related to the estimated state $\mathbf{x}_k$ through a measurement matrix $\mathbf{H}_k$, at each time step k :

$$\mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k$$

Where:

- $\mathbf{H}_k$ is the measurement matrix that maps the state space to the measurement space.
- $\mathbf{v}_k$ is the assumed measurement noise with covariance matrix $\mathbf{R}_k$.

Estimation and measurements fusion

For each time t, the Kalman filter will predict the next state of the filter based on the previous state and process noise using the state transition matrix, this is the prediction substep. After that, the filter updates its state estimation based on the latest measurements and measurement noise using the measurement matrix.

The prediction step estimates the state of the system at the next time step based on the current state and the system dynamics:

$$\hat{\mathbf{x}}_k^- = \mathbf{F}_k \hat{\mathbf{x}}_{k-1} + \mathbf{B}_k \mathbf{u}_k$$

$$\mathbf{P}_k^- = \mathbf{F}_k \mathbf{P}_{k-1} \mathbf{F}_k^T + \mathbf{Q}_k$$

Where $\hat{\mathbf{x}}_k^-$ and $\mathbf{P}_k^-$ are the predicted state estimate and state covariance matrix respectively. The "super minus" indicates that the estimate is based only on the previous knowledge of the filter without new measurements.

The update step corrects the predicted state based on the current measurements:

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}_k^T (\mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^T + \mathbf{R}_k)^{-1}$$

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k (\mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_k^-)$$

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_k^-$$

Where $\hat{\mathbf{x}}_k$ and $\mathbf{P}_k$ are the updated state estimate and state covariance matrix respectively, and $\mathbf{K}_k$ is the Kalman gain.

4.1.2 Kalman filter implementation

My implementation is built upon this Kalman filter Python library [43]. In the following, I will break down the different choices made. The robots are assumed to be moving according to the constant velocity model [28] [29], this is true most of the time during swarm experiments, except when the robot is changing directions. In this model, the state variables are the x and y positions and their derivatives (x and y velocities):

$$\mathbf{x} = \begin{pmatrix} x \\ y \\ x' \\ y' \end{pmatrix}$$

The state variables evolve in the following way based on the transition matrix $\mathbf{F}$ where Δt represents the time elapsed between each prediction step and $\mathbf{w}_k$ is the process noise with mean 0 and covariance matrix $\mathbf{Q}$:

$$\mathbf{x}_{k+1} = \begin{pmatrix} x_{k+1} \\ y_{k+1} \\ x'_{k+1} \\ y'_{k+1} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_k \\ y_k \\ x'_k \\ y'_k \end{pmatrix} + \mathbf{w}_k$$

The measurement variables here are ($x1$, $y1$) which corresponds to the position measured by the first sensor and ($x2$, $y2$) measured by the second sensor, and so on. The following shows how measurements from two sensors are related based on the $\mathbf{H}$ matrix to the state variables and the measurement noise with mean 0 and covariance matrix $\mathbf{R}$:

$$\mathbf{z}_k = \begin{pmatrix} x1_k \\ y1_k \\ x2_k \\ y2_k \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_k \\ y_k \\ x'_k \\ y'_k \end{pmatrix} + \mathbf{v}_k$$

Additionally, the process noise covariance matrix $\mathbf{Q}$ needs to be defined for the robot detection problem. In [38], the $\mathbf{Q}$ is defined based on an acceleration noise factor $\mathbf{q}$ since we consider a constant-velocity model. The value for $\mathbf{q}$ was arbitrarily set to 0.0001 and I chose to go with the same value before doing any fine-tuning of the Kalman filter parameters. Note that the diagonal elements of a covariance matrix are the variances of the variables and since the variables in our system are independent, all other values in the matrix are put to 0.

$$Q = \begin{bmatrix} Q_x & 0 & 0 & 0 \\ 0 & Q_y & 0 & 0 \\ 0 & 0 & Q'_x & 0 \\ 0 & 0 & 0 & Q'_y \end{bmatrix} = q \times \begin{bmatrix} \frac{1}{2}\Delta t^2 & 0 & 0 & 0 \\ 0 & \frac{1}{2}\Delta t^2 & 0 & 0 \\ 0 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & \Delta t \end{bmatrix}$$

Where $\frac{1}{2}\Delta t^2$ represents the contribution of the acceleration noise to the noise of the position and Δt, the contribution of the acceleration noise to the noise of the velocity.

The measurements noise covariance matrix $\mathbf{R}$ can be easily computed based on the accuracy of the sensors that can be found on their datasheets [38].

- The datasheet for the stereo camera can be found on [44]. The relative error in the distance estimation between the camera and an object depends on the distance:
  - Below 4m: less than 2%
  - 4m - 7m: less than 4%
  - 7m - 10m: less than 6%
- The datasheet for the LiDAR can be found on [45]. The relative error in the distance estimation between the LiDAR and an object depends on the distance:
  - Below 1m: absolute error of +- 0.02m
  - Further than 1m: less than 3.5%

Remember that the measurement variables are the position given by the camera (x1, y1) and the position given by the LiDAR (x2, y2). Let's look at how to compute the error in the position variables based on the error in the distance. For some angle θ, x and y in 2D coordinates are directly proportional to the distance to the object:

$$\begin{cases} x = d \cdot \cos\theta \\ y = d \cdot \sin\theta \end{cases}$$

The absolute error on the distance can be written as $d \cdot \epsilon_d$ where ϵ_d is the relative error on the distance. Let's write the absolute errors on x and y:

$$\begin{cases} x \cdot \epsilon_x = (d \cdot \epsilon_d) \cdot \cos\theta \\ y \cdot \epsilon_y = (d \cdot \epsilon_d) \cdot \sin\theta \end{cases}$$

$$\begin{cases} \epsilon_x = \frac{(d \cdot \epsilon_d) \cdot \cos\theta}{x} \\ \epsilon_y = \frac{(d \cdot \epsilon_d) \cdot \sin\theta}{y} \end{cases}$$

$$\begin{cases} \epsilon_x = \frac{d \cdot \epsilon_d \cdot \cos\theta}{x} \\ \epsilon_y = \frac{d \cdot \epsilon_d \cdot \sin\theta}{y} \end{cases}$$

$$\begin{cases} \epsilon_x = \epsilon_d \\ \epsilon_y = \epsilon_d \end{cases}$$

The relative errors on x and y are equal to the relative error on the distance. The values needed in a covariance matrix are not the relative errors but the absolute variances. The absolute variance of some variable is the square of the absolute error. The datasheets of the sensors showed that the error is not fixed but relative to the actual values of the measurements. This means that the measurement noise covariance matrix needs to be updated for each new set of measurements before updating the Kalman filter with the measurements [46]. The measurement noise covariance matrix $\mathbf{R}$ is computed as follows before each update step, where ϵ_{camera} is the relative error for the stereo camera and ϵ_{LiDAR} is the relative error for the LiDAR.

$$\text{R}=\begin{bmatrix} (x1\cdot\epsilon_{camera})^2 & 0 & 0 & 0 \\ 0 & (y1\cdot\epsilon_{camera})^2 & 0 & 0 \\ 0 & 0 & (x2\cdot\epsilon_{LiDAR})^2 & 0 \\ 0 & 0 & 0 & (y2\cdot\epsilon_{LiDAR})^2 \end{bmatrix}$$

The last matrix to initialize for the Kalman filter is the state noise covariance matrix $\mathbf{P}$. The best values for this matrix need to be determined by experimenting with different values but for the moment, I take the same approach as done in the article [38] where the detection and tracking are performed on pedestrians with constant velocity model, the initial value for matrix $\mathbf{P}$ is seen as the distance uncertainty of the detection based on the average width a human body and taking half of it as error radius of the detection. I do the same thing but with the average radius of the robot r:

$$\text{P}=\begin{bmatrix} r & 0 & 0 & 0 \\ 0 & r & 0 & 0 \\ 0 & 0 & \frac{r}{\Delta t} & 0 \\ 0 & 0 & 0 & \frac{r}{\Delta t} \end{bmatrix}$$

Let's summarize:

- When a robot is detected, a Kalman filter is initialized with transition matrix $\mathbf{F}$, measurements mapping matrix $\mathbf{H}$, process noise covariance matrix $\mathbf{Q}$ and state noise covariance matrix $\mathbf{P}$.
- After Δt time has elapsed, perform a round of prediction, update measurements noise covariance matrix $\mathbf{R}$ and update the Kalman filter with the new measurements.
- How to fix Δt in the transition matrix $\mathbf{F}$? There are several ways to handle measurements coming from different sensors:
  - Update the Kalman filter with measurements when they are new ones from any sensor and update the transition matrix $\mathbf{F}$ with the time elapsed since the last prediction.
  - Update the Kalman filter at a fixed frequency with the latest measurements from all sensors. There is also no need to update the transition matrix $\mathbf{F}$ since the frequency of the Kalman filter itself is fixed.
  - Update the Kalman filter when measurements arrive from a designated sensor (could the one with the highest frequency or the most reliable) and use the latest measurements from the other sensors. In this case, since the frequency of the sensor is fixed, the transition matrix $\mathbf{F}$ is fixed when the Kalman filter is initialized.

  I chose the last method over the first one since the stereo camera is more reliable than the LiDAR at detecting robots and has a much higher frequency (at least 4 times the frequency of the LiDAR), so using the first method would not increase the accuracy. The second method is basically the same as the last one since the camera has a fixed frame rate.

4.1.3 Data association problem

In the context of multiple object detections with detections coming from different sensors, detections of the same object but from different sensors need to be associated together using a data association algorithm to be fused. Furthermore, as explained in a previous section, the sensor fusion algorithm chosen for the high-level fusion is the Kalman filter. The incoming detections need also to be associated with the corresponding Kalman filter.

The data association algorithm used in this work is the Hungarian algorithm [47]. The Hungarian algorithm was first designed to solve the assignment problem between workers and jobs but it is widely used in the domain of sensor fusion to associate data from different sensors and current state estimate [17] [27] [30].

The Hungarian algorithm implementation used is from the SciPy Python library [48] which is based on [49].

To stay generic and fit both the use cases explained above for the Hungarian algorithm in this work, the data to be associated are two lists of 2D positions in (x,y) coordinates, each position corresponds to a detected robot (if the list is coming from a sensor) or current position estimation of some robot (coming from the currently used Kalman filters).

The algorithm takes as input a rectangular cost matrix where rows correspond to the positions from the first list, and columns to the positions from the second list. The output consists of two lists: the first one contains the indices of the positions from the first list that were matched in ascending order, and the second one contains the indices of the positions from the second list that were matched and ordered to correspond with the indices from the first list.

4.2 Robots detection with camera

4.2.1 Previous work

In my previous work on the Master 1 project, I developed software to detect Sphero RVR robots using the Oak-D camera. The software used a YoloV5 neural network that ran directly on the camera. The neural network was trained on a dataset that I built from images of the previous prototype of the Mercator robot. The newest prototype has a new shell that covers nearly the entirety of the robot, as shown in Figure 4.1, making the previously trained neural network unable to detect it and obsolete.

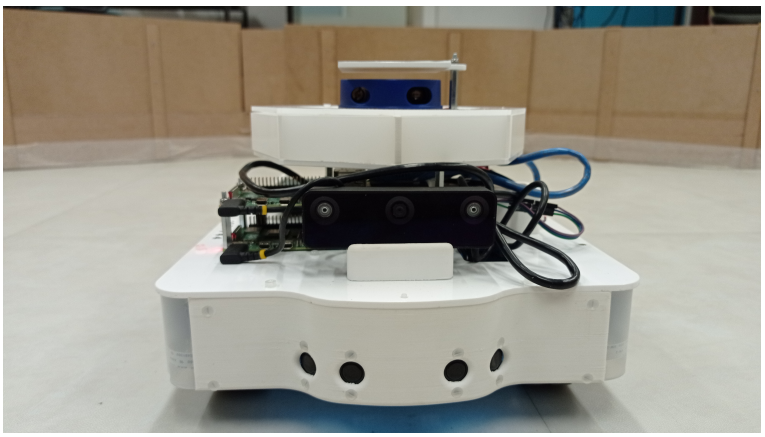


Figure 4.1: Current version of the Mercator robot

Furthermore, the previous neural network was trained using the trial version of the Roboflow cloud training service. The disadvantages were that the training was limited and could not further train the neural network with additional images or objects to detect.

For the previous reasons, the neural network training method had to be changed and redone. The following sections will cover, the dataset building, the training of the neural network, and the results.

4.2.2 Dataset building

The dataset must contain relevant images of the robots in the environment where the experiments will occur. This means that:

- The images must be taken while the robots are in the arena.
- Must contain various images where the robots are moving across the arena, staying still, aggregated, etc.
- The images must be taken with the cameras mounted on the robots. This was not the case with the previous dataset since the camera was not present on the previous prototype.

3 robots were set to move randomly inside the arena (for more information about the random walk algorithm used, see appendix A.1) for 3 minutes with their cameras recording. This was done 3 times to get a total of 9 videos. The videos were recorded with robots moving at a speed of 0.3 m/s.

Those videos are then sampled to get a total of 200 images where robots and obstacles are present. An example is shown in Figure 4.2.

The images were then annotated and the dataset was augmented (to a total of 500 images) by adding images with the following augmentations:

- Horizontal flip.
- Brightness ranging from -15% and 15%.
- Blur up to 2px.

Figure 4.2: Image taken from a moving robot of 2 other robots and an obstacle

4.2.3 YOLOv8 training

The neural network model trained is YOLOv8n[1]. The model was trained locally on GPU for 183 epochs (early stopping to avoid overfitting since no improvement was seen 100 epochs later) with an input resolution of 256x256.

The obtained Pytorch model was compiled to blob format that can be used on the Oak-D camera with the tool provided by the manufacturer[2].

4.2.4 Results

The performance metrics of the trained neural network are shown in table 4.1. Precision measures the ratio of true positives to all detected positives, while recall measures the ratio of true positives to all annotated positives. mAP50 and mAP50-95 are more complex measures based on both precision and recall that assess the model performance overall[3]. The focus of this work is not the maximize the accuracy of the neural network used but just to ensure that it performs sufficiently to be used in a sensor fusion scheme.

	Robot	Obstacle
Precision	0.988	0.909
Recall	0.875	0.766
mAP50	0.916	0.929
mAP50-95	0.807	0.797

Table 4.1: Neural network performance metrics on test set.

4.3 Robots detection with LiDAR

LiDAR is a less suitable sensor for the object detection task than a camera is, mostly because of the higher resolution of cameras and the ability to include colors in the extracted features. In case of 3D LiDAR, detection is mostly done by Neural Networks [17][24][26][50][51][52] but also by clustering methods [30][38].

In the case of the Mercator robot, the LiDAR is a 2D 360° LiDAR, which makes it harder to perform detection using only the data of the LiDAR than with a more expensive 3D LiDAR. Unlike 3D LiDAR, deep learning based methods are less popular in 2D due to the lack of input data, unless it is used together with other sensors as done in [6] where the neural network takes as input all the data from the 2D LiDAR, sonar, and stereo camera to perform detection. It is also possible to use a clustering method to detect targets of interest, this is well done in [38] where Euclidean clustering is used to detect pedestrians with 2D LiDAR and Kalman filter for tracking.

Knowing the possible sensor fusion schemes with the camera/stereo camera, this section focuses on the potential detection and position estimation of the targets using the data from LiDAR alone or alongside the data from the cameras.

[1] https://github.com/ultralytics/ultralytics

[2] http://tools.luxonis.com/

[3] `https://docs.ultralytics.com/guides/yolo-performance-metrics/#class-wise-metrics`

4.3.1 Standalone LiDAR detection

The first possibility is to try to detect the targets with only the data from the LiDAR. The method I choose for this task is the same Euclidean clustering algorithm from [38] since the targets here are the other robots and they have fixed widths. More precisely, the LiDAR of each robot is at the same height so the LiDAR has to detect other LiDARs. The main problem with clustering is that obstacles with similar widths to the robots may be confused for them, thus detecting more robots than in reality. Keeping in mind that the output of this detection algorithm will be used in a sensor fusion scheme, it is possible to filter out false positives later with the data from the other sensors.

Here is in detail the Euclidean clustering algorithm implemented:

- **Initialization:**
  - All the depth data from the LiDAR is converted to 2D cartesian points.
  - A KDTree data structure is initialized with the points. KDTree can facilitate the search for neighbor points within a certain radius for multidimensional points.
  - A set of unvisited points is initialized with the indices of all the points.
  - An empty clusters list is initialized to store clustering results.
- **Forming a cluster:**
  - Take an unvisited point and add it to the current cluster.
  - Add the point to the search queue for the current cluster.
  - Search neighbors for all points in the search queue till there are no more points in the queue:
    - Find the neighbors for the current point within the specified distance threshold using the KDTree structure.
    - For each unvisited neighbor:
      - Remove the neighbor from the unvisited set.
      - Add the neighbor to the current cluster.
      - Add the neighbor to the search queue to continue searching for more potential neighbors to add to the cluster.
  - When there are no more points in the search queue, the cluster is formed but still needs to satisfy a few conditions to be a valid cluster:
    - It must Contain a minimum number of points and less than a maximum number of points.
    - The cluster's centroid is computed along the distance of the furthest point of the cluster from the centroid. This distance must be less than a certain threshold determined by the target's characteristics (0.04m for Mercator robot).
- Repeat the **forming a cluster** step till there are no more points in the unvisited set.

4.3.2 Camera assisted LiDAR detection

The following detection methods using the data from the LiDAR but assisted by data from the camera can be considered sensor fusion methods since data from different sensors is used to enhance the overall output. They are presented in this section since they are also used as a level of abstraction in the more complete sensor fusion schemes in this work.

As discussed before in section 4.2.1, the camera on the robot is actually two sensors in one, where the central RGB camera is used to perform detections and the stereo pair generates depth maps of the environment that can be used to estimate the 3D position of detected targets.

The partially assisted LiDAR detection method uses only the data from the central camera while the fully assisted LiDAR detection method uses both the data from the central camera and the stereo pair.

Partially assisted LiDAR detection

The goal here is to use the bounding boxes around the targets generated by the camera detection to guide the LiDAR at which angle to look for targets and measure the distance to them.

By limiting the interesting depth data of the LiDAR to the values inside the ranges defined by the bounding boxes, there are still tens or hundreds of values to choose from to determine the more precisely possible distance from the LiDAR to the target. The strategy is to reuse the same clustering algorithm as in the standalone LiDAR detection 4.3.1 but this time only on values of the LiDAR inside the bounding box.

- Each time the LiDAR readings are received:
  - Check if there are currently detected targets by the camera.
  - For each detected target, convert the bounding box to a pair of minimum and maximum angles to delimit the relevant LiDAR readings.
  - Apply the Euclidean clustering algorithm to those readings and extract clusters of points.
  - If there are several clusters for a single bounding box, the cluster that is the closest to the center of the bounding box should be chosen since the LiDAR can only see the LiDARs of other robots which happen to be always at the center of the robots.

Fully assisted LiDAR detection

The most complete method of detection for the robot, in the sense that it uses all the available data from the camera and stereo camera to estimate the distance with the LiDAR.

Going further than just limiting the relevant LiDAR readings to the one delimited by the bounding box like in the previous method, here, the closest reading from that range to the position estimated by the camera is chosen as the LiDAR position estimation. To avoid having positions estimated by the LiDAR that are too far away from reality in the case the LiDAR is unable to detect the target and would select the closest obstacle instead, a distance threshold is set. The distance threshold depends on both the precision of the camera and the LiDAR and should be determined empirically.

4.4 Camera LiDAR calibration

In this work, the camera and the LiDAR are considered to be already calibrated to function individually and focus on how to calibrate them to each other or in other words, how to ensure that the data coming from both of them is aligned.

The method used to calibrate the camera to each other is the Least-square fitting method [53] that is used to calibrate a set of multiple sensors pair-wise in [54]. The method used here will be limited to only 2 sensors and 2D.

The method consists of calculating a transformation matrix that will be used to transform all the positions estimated by a sensor to fit the frame of the other sensor.

In the case of this work, the sensor fusion methods will be evaluated against ground truth data from a tracking system. To avoid redoing calibration a second time between the sensor and the tracking system, the transformation matrix will be computed once for each sensor with the data from the tracking system. This still accomplishes the goal of aligning the two sensors on top of facilitating later experiments.

4.4.1 Calibration algorithm

The transformation matrix calculation can be separated into multiple steps:

Positions re-centering

The centroids of both sets of poses coming from the sensor and the tracking system are computed separately. The centroid is the average position (x,y) of all the points in one set:

$$\mathbf{c} = \left(\frac{1}{n} \sum_{i=1}^{n} x_i, \frac{1}{n} \sum_{i=1}^{n} y_i \right)$$

Where n is the number of points in one set.

After that, For all the points in both sets, subtract the centroid to re-center the data with the centroid as origin:

$$\mathbf{S} = \mathbf{s}_i - \mathbf{c}_{\text{sensor}}$$

$$\mathbf{G} = \mathbf{g}_i - \mathbf{c}_{\text{ground\_truth}}$$

Compute rotation and translation

Compute the covariance matrix $\mathbf{H}$ and apply Singular Value Decomposition method (SVD):

$$\mathbf{H} = \mathbf{G} \cdot \mathbf{S}^T$$

$$\mathbf{U}, \mathbf{\Sigma}, \mathbf{V} = \text{SVD}(\mathbf{H})$$

Compute 2x2 Rotation matrix:

$$\mathbf{R} = \mathbf{V} \cdot \mathbf{U}^T$$

Compute 2x1 translation vector will align the sensor's positions centroid to the ground truth's positions centroid:

$$\mathbf{t} = \mathbf{c}_{\text{sensor}} - \mathbf{R} \cdot \mathbf{c}_{\text{ground\_truth}}$$

Combine into the final transformation matrix

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix}$$

4.4.2 Data for calibration

The data needed for calibration must have a one-to-one correspondence, each position from the sensor set must correspond to one position from the ground truth set. To ensure this correspondence and that a position in the sensor frame is the same correspondent position but in the ground truth frame (or another sensor frame, if the calibration goal is to align to another sensor), a target that can be detected by both the sensor and the tracking system is needed.

The target here is simply a robot since the camera and the LiDAR are both able to detect it. The tracking system does not directly detect the robot but can detect ArUco markers, this is explained in detail in the experiments section 6.1.

Additionally, there are two important points to take into account:

- To ensure no ambiguity in target correspondence, only one target robot will be used at a time to collect the calibration data.
- To ensure higher coverage of the possible positions that the targets can take in front of the sensors, the target will take a pre-determined path, since letting the robot move randomly would take much more time and give more weight to some positions where it might get stuck for a moment.

Chapter 5

Sensor fusion techniques

5.1 Complementary fusion

Complementary fusion takes advantage of the fact that different sensors gather different data to merge. This is the way to go for systems where each sensor is specialized in some type of data but in the case of the Mercator robot, the Oak-D camera has a stereo camera pair that computes the depth of the environment, just like the 2D LiDAR. So, doing complementary fusion is an underuse of available sensors.

In the case of the Mercator robot, the camera detects the target and then the LiDAR estimates the distance to the target, knowing in which direction the target is thanks to the camera.

Complementary fusion between the camera and the LiDAR is still interesting to implement on the Mercator robot to perform sensor fusion for the following reasons:

- The stereo pair of cameras of the Oak-D can be turned off/on at will and is not necessary for the mono RGB camera to detect robots/objects. Turning it off can save up to 0.5 W (the base consumption of the mono camera + AI model is 4.5 W)[1].
- Processing power is freed to increase the framerate at which the detections are done.
- The method is used as a baseline for comparison with other methods to evaluate the improvement that the stereo pair brings to the sensor fusion system.

In practice, the complementary fusion between the mono RGB camera and LiDAR is explained in detail in the section about LiDAR detection and position estimation assisted by the camera 4.3.2. It is interesting to note that the Oak-D camera itself already uses a complementary fusion scheme between the Mono RGB camera and the stereo pair of cameras. The algorithm is provided by the manufacturer [37] and does the following:

- The mono camera detects objects and computes the size and location of the bounding boxes on the image.
- The stereo pair of cameras computes a depth map that is aligned to the mono camera.
- The algorithm runs on the Oak-D camera itself. The bounding box determines a region of interest on the depth map, all out-of-range depth values are discarded,

[1] https://docs.luxonis.com/hardware/platform/environmental-specifications/power-consumption/

and the remaining values are averaged to provide the depth of the detected object as shown in Figure 5.1.

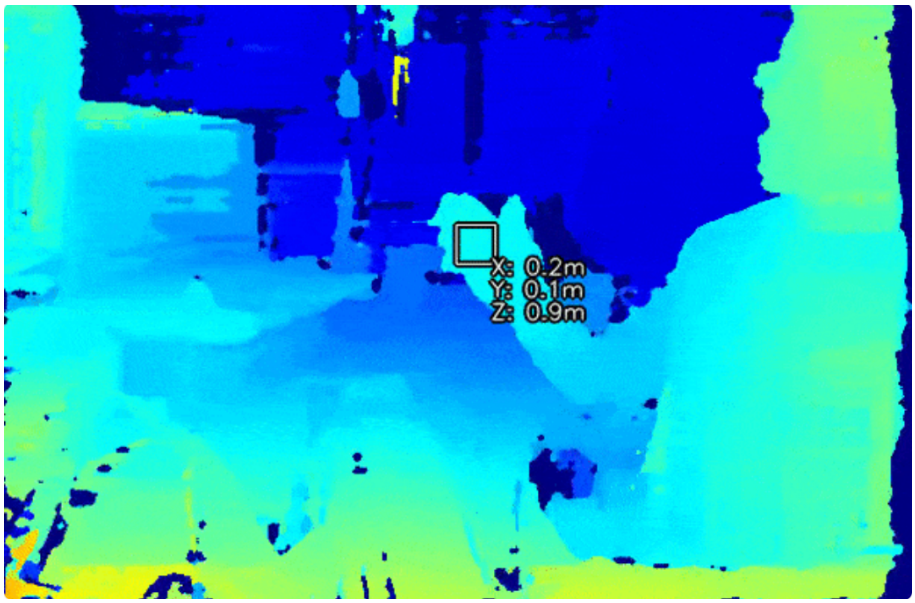


Figure 5.1: Illustration of the depth of some region of interest with Oak-D camera [37]

5.2 High-level and modular fusion

This fusion technique assumes that each sensor will perform the detection of the robots and estimate their relative position individually. The scheme is modular because it can process input data from any number of sensors. The maximum number of sensors needs to be specified but the scheme assumes that any sensor can fail at any time.

Practically, each sensor produces a list of several positions in the form of a ROS "geometry_msgs/PoseArray" message in a ROS topic specific to that sensor.
The algorithm is implemented within a ROS node that subscribes to each of the sensors' topics and publishes the fused positions in a new topic as a ROS "geometry_msgs/PoseArray" message.

The following is a breakdown of the algorithm:

- Initialization:
  - Step 1: Initialization of the ROS node with the number of sensors to fuse and their topic names as ROS parameters.
  - Step 2: Initialize an empty set of Kalman filters with the same parameters as in section 4.1.2.
- For each time t:
  - Step 3: For each additional sensor than the first one, perform matching between the positions given by the first sensor and each of the other sensors, one by one, using the Hungarian algorithm (see section 4.1.3). The output for each matching step is a list of indices corresponding to the matched positions from sensor 1 and a list of indices corresponding to the matched positions from the other sensor. The first list is ordered in ascending order and the second is ordered to match the first list.

- Step 4: From the previous step, we got N-1 lists of positions from the first sensor that were matched with the other sensors. The intersection of those positions is kept.
- Step 5: Perform matching between the current states from the set of Kalman filters and the positions from the first sensor (this is arbitrary, it could be anyone since we are keeping only the detections that are agreed upon by all the sensors for a certain time t). The matching algorithm used is also the Hungarian Algorithm. For unmatched positions, initiate a Kalman filter with the position as the initial state and add it to the list of matched Kalman filters.
- Step 6: For each matched Kalman filter, perform a round of prediction and then correction using the positions computed by each sensor for the detections that were kept. Delete unmatched Kalman filters that were not matched in the last 10 iterations.

5.2.1 Camera LiDAR specific fusion algorithm

The algorithm described previously is suitable in cases where the sensors have a tendency to make a lot of false detections and if we prefer to have high confidence in the detections, but this can lead to missing some detections. There are definitely variations to this algorithm and modifications to bring to get the best possible results but this depends heavily on the type of sensors used and how reliable they are in their detections and measurements.

I will not go through all the possibilities and will only propose a variation for the current prototype of the Mercator robot, based on the following:

- The camera proved in previous work to be very reliable in detecting any robots present in its FoV and estimating their positions.
- Detecting robots with a 360° 2D LiDAR is not a trivial task due to the lack of relevant features of the robot that could be extracted from that data (only a horizontal line of 720 depth values). The method used in this work to detect the robots is a clustering method that is explained in detail in section 4.3.1. Since the LiDAR of a robot is at the same height as the other LiDARs of the rest of the robots in the swarm, the clustering method is tuned to detect the LiDARs to detect the actual robots. Obstacles of a similar size to the LiDAR can be confused with robots. This means that the LiDAR will detect more robots than there are in reality. Overall, the LiDAR is less reliable in detecting the robots than the camera due to its lower resolution, being only 2D and not 3D, and confusing some obstacles with robots.

The following are the steps of the algorithm specifically for the camera and LiDAR setup:

- Initialization:
  - Step 1: Initialization of the ROS node with the number of sensors to fuse equal to two and their topic names as ROS parameters.
  -
  - Step 2: Initialize an empty set of Kalman filters with the same parameters as in section 4.1.2.
- For each set of detections received from the camera:

- Step 3: Perform matching between the positions given by the camera and the LiDAR using the Hungarian algorithm.
- Step 4: From the previous step, keep all the positions the camera gave even if they were not matched.
- Step 5: Perform matching between the current states from the set of Kalman filters and the positions from the camera. The matching algorithm used is also the Hungarian Algorithm. For unmatched positions, initiate a Kalman filter with the position as the initial state and add it to the list of matched Kalman filters.
- Step 6: For each matched Kalman filter, perform a round of prediction and then correction using the positions from the camera, and corresponding positions from LiDAR if they were matched. Delete unmatched Kalman filters that were not matched in the last 10 iterations.

Note that with this variant, the algorithm is still modular and can fuse data from additional sensors, the only assumption made here is that the camera is way more reliable in terms of detection than the other sensors could be.

In both Hungarian algorithm matching loops, only matched positions with a distance less than 0.5m from each other are considered matched. This is to ensure that if one sensor detects target A but not target B and the other sensor detects target B and not target A, that target A will not be falsely matched with target B.

A flowchart of the High-level fusion algorithm to fuse data from the camera and LiDAR is shown in Figure 5.2.

5.3 Summary of the sensor fusion methods

In this section, I will summarize the sensor fusion methods that will be evaluated during the experiments. For each method, there is a flow chart of the main components and steps.

Three of these methods are variations of the high-level and modular fusion technique presented before but using different LiDAR detection methods.

It is debatable why the LiDAR detection methods were not evaluated separately and then the most fitting one was picked to do the experiments in this work. There are two reasons for that:

- Apart from the standalone LiDAR detection using purely Euclidean clustering, the other two methods use data from the camera to assist in performing detections for the LiDAR. This means that these methods are already sensor fusion schemes where one uses only the bounding boxes of the detected targets by the camera and the other uses also the estimated position provided by the camera. Since the purpose of this work is to enhance the perception capabilities of the robots using sensor fusion, these methods must be evaluated separately.
- Some LiDAR detection methods might perform poorly by themselves and at the same time bring the highest enhancement in results when combined with the estimated positions from the camera: For example, the standalone detection method might have less detection rate since it is not assisted by the detections from the camera but at the same time might be the most precise method among the three.

At the same time assisted LiDAR detection methods might have higher detection rates and lower precision but they can bring higher enhancement in the final result when fusing the data with the camera since even low precision measure will make the Kalman filter predict more accurately as long as its error is estimated correctly in the measurements covariance matrix.

5.3.1 High-level fusion with standalone LiDAR detection

The camera and the LiDAR perform detection and position estimation independently and then their outputs are fused together based on the algorithm described in section 5.2.1 and shown in Figure 5.2.

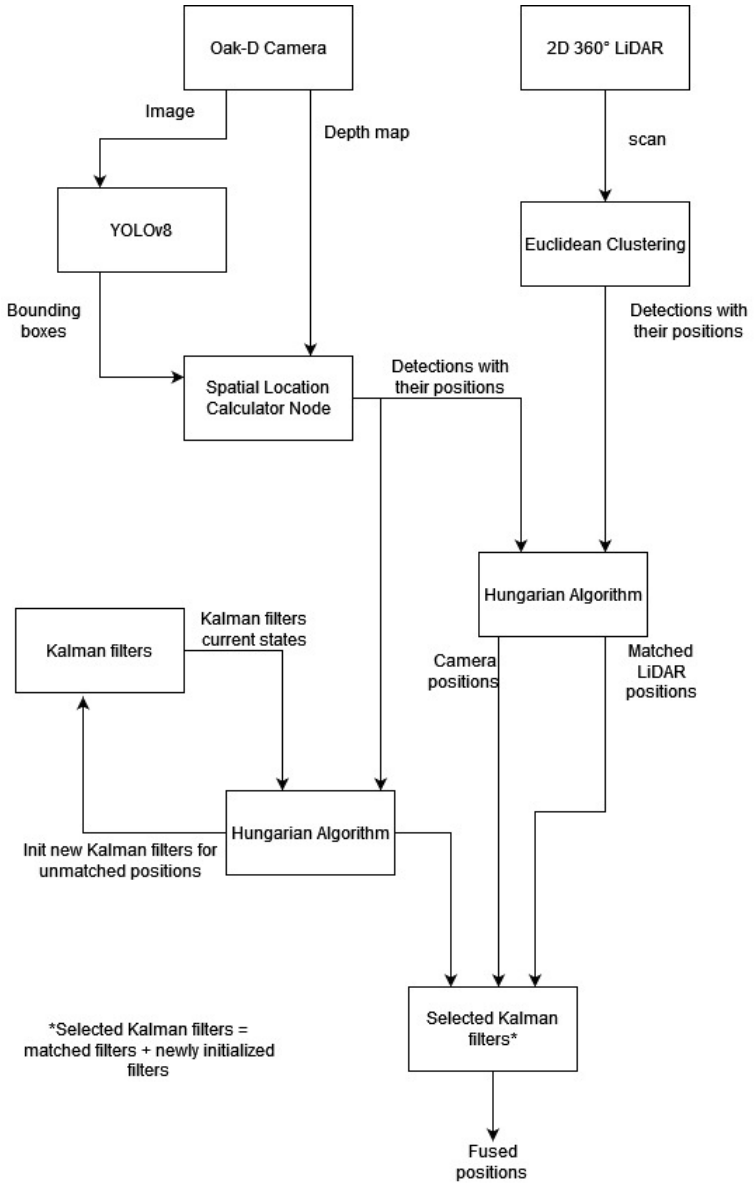


Figure 5.2: High-level fusion with standalone LiDAR detection flowchart.

5.3.2 High-level fusion with partially assisted LiDAR detection

Same core algorithm as the previous method, but now the LiDAR is assisted in its detections by focusing the clustering only on the angles defined by the bounding boxes provided by the camera. Only the modified part is shown in Figure 5.3

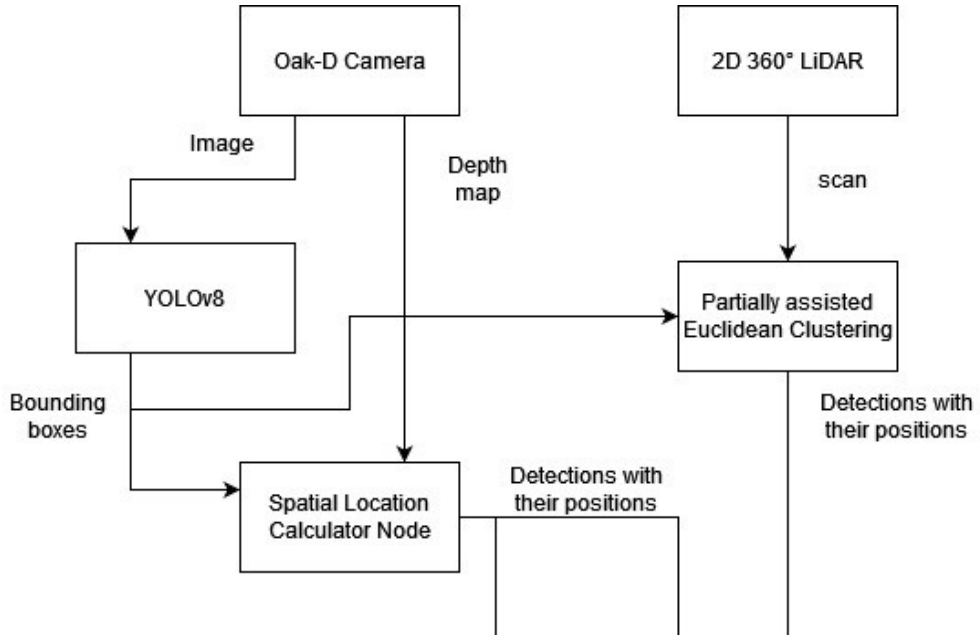


Figure 5.3: High-level fusion with partially assisted LiDAR detection flowchart.

5.3.3 High-level fusion with fully assisted LiDAR detection

One step further, LiDAR uses the positions provided by the camera to estimate its own positions by choosing the closest reading to that position in a certain radius (0.15m by default, slightly larger than the radius of the Mercator robot) if such a reading exists. The modified part is shown in Figure 5.4.

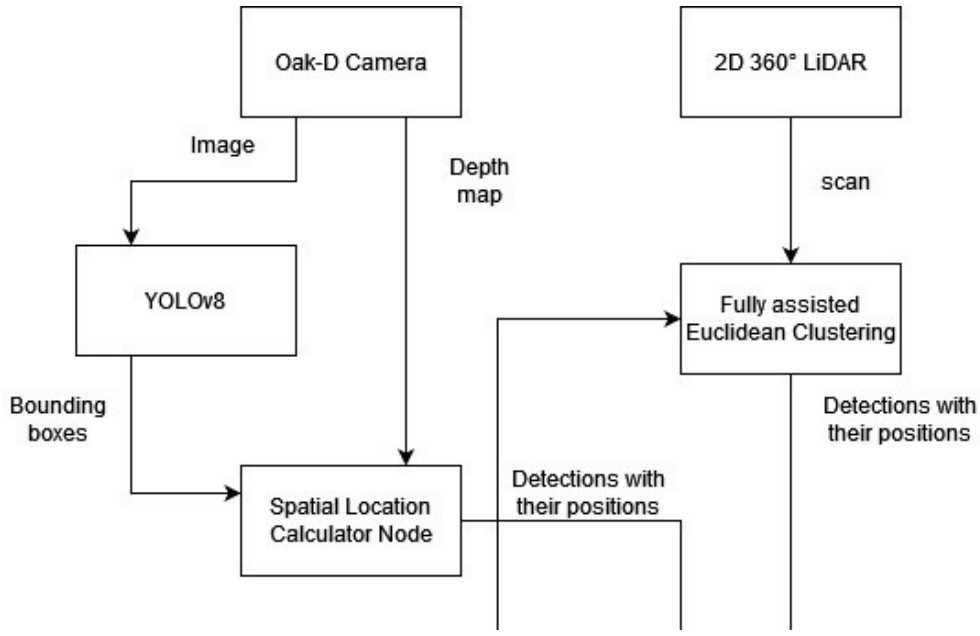


Figure 5.4: High-level fusion with fully assisted LiDAR detection flowchart.

5.3.4 Complementary fusion

In the following sections, complementary fusion and partially assisted LiDAR detection are synonymous in the sense that the high-level fusion algorithm with partially assisted LiDAR detection is complementary fusion but with the extra step of including positions

detected by the cameras using the depth maps generated by the stereo pair. Complementary fusion is the counterpart where the stereo pair of the camera is not used to save up on power.

The flowchart for this method is shown in Figure 5.5

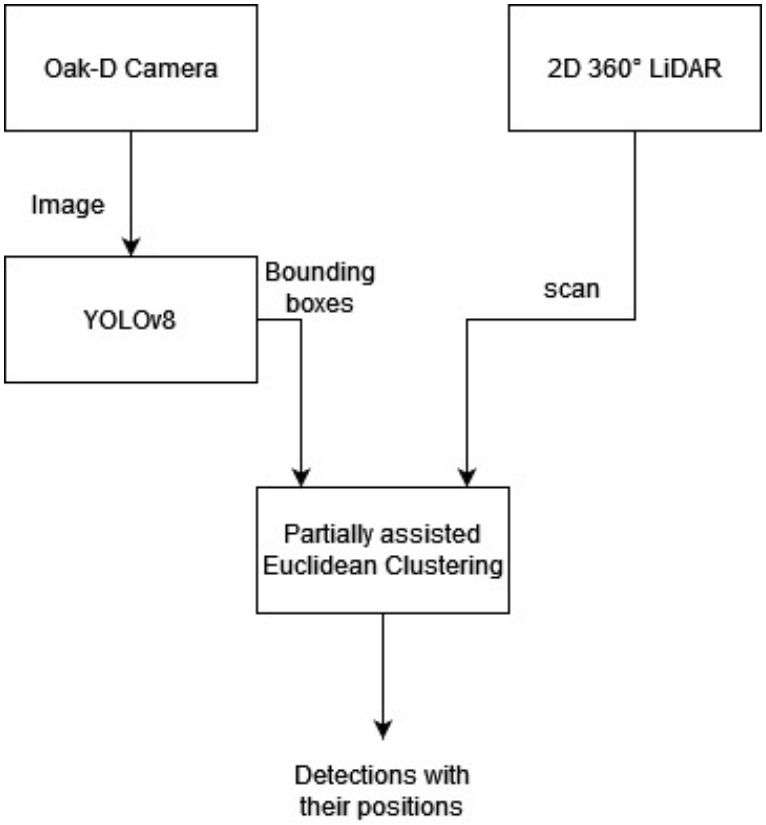


Figure 5.5: Complementary fusion flowchart.

Chapter 6

Experiments

6.1 Experiments setup

All the experiments were conducted in the IRIDIA's robots arena (shown in Figure 6.2). The arena has several fixed cameras placed on the ceiling to permit video recording of the experiments. The camera system is also able to track and estimate the positions of the robots inside the arena using ArUco markers [55] that are placed on top of each robot as shown in Figure 6.1.

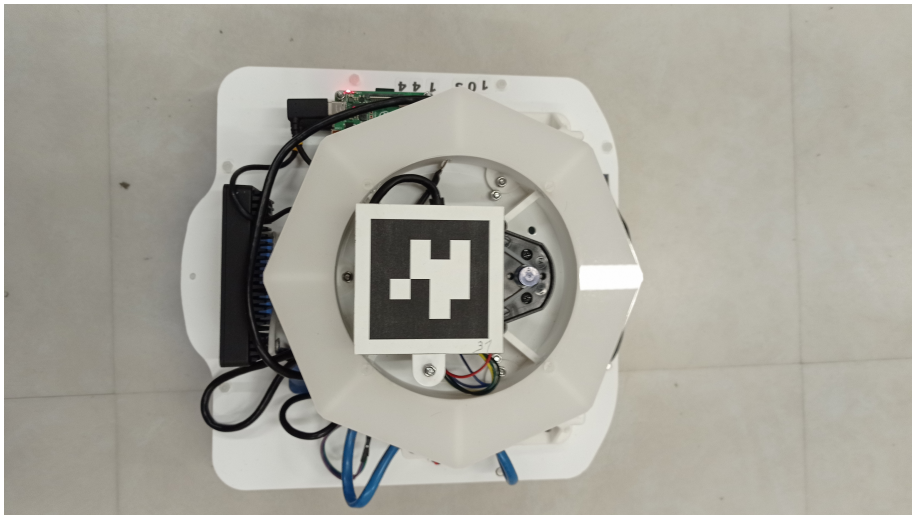


Figure 6.1: ArCuo Marker on top of a Mercator robot

6.2 Sensor fusion methods evaluation

To evaluate the accuracy and compare the different sensor fusion methods implemented in this work, it is necessary to have ground truth data as a reference for this evaluation. The robots' estimated positions from the arena tracking system [56] will be used as ground truth data during the experiments.

The arena tracking system cannot consistently track robots moving at a speed equal to or higher than 0.4 m/s. For this reason, all the experiments will be conducted with robots moving at a maximum speed of 0.3 m/s to ensure that the ground truth data does not contain any drifts or jumps in the positions of the robots.

6.2.1 Metrics

For the evaluation, similarly to other works where the precision of a sensor fusion system needs to be assessed, the metric used is the Root Mean Square Error (RMSE) of the estimated positions of each robot during an experiment [22] [28].

Additionally to the RMSE, other metrics are good to monitor to try to extract patterns or scenarios where the sensors perform better or worse than what the RMSE can tell. Here are the additional metrics used to evaluate the camera and LiDAR performances and how they reflect on the sensor fusion results:

- Cumulative error distribution plot: The goal is to see how the error is distributed by plotting the cumulative frequency of different error values. The different error values are separated into 100 bins. In this plot, it is also interesting to extract a unique value like the error such that 67% (0.67 cumulative frequency) of the errors are below that value. This is to see whether there is a minority of high errors that drag the RMSE up.
- Error over distance plot: The goal is to see the error in position estimation for different distance values. The distance here is the ground truth distance value between the sensing robot and the target. Distance values are separated into bins of 0.1 m.
- Error over angle plot: The goal is to see whether the angle between the sensing robot and the target has an impact on the error or not. Again the angle is based on the ground truth data.

Estimation of the position of a robot by another robot is only possible if that robot is directly in its field of view and successfully detected. The detection of the robots has already been evaluated in a previous section 4.2.4 using classical computer vision model metrics.

6.2.2 Metrics computation method

All the previously chosen metrics are based on the difference between the estimated position of a robot and the ground truth position from the tracking system. Before being able to compute this difference for every time t, several steps are needed:

- Convert all the positions to the same coordinates frame: The ground truth positions from the arena tracking system are computed in the arena frame while the estimated positions of other robots are computed in the robot frame. For each robot, the ground truth positions of other robots are converted to the robot frame. Arena frame coordinates system origin and an example of a robot frame origin are shown in Figure 6.2
- Since there is no way to know beforehand which estimated position of a detected robot corresponds to which ground truth position given by the tracking system, the Hungarian algorithm [49] is used to do the matching.
  - At each time t, a cost matrix is computed between the estimated and ground truth positions.
  - The Hungarian algorithm is applied to the cost matrix to compute the matching that gives the less total cost, which means, in this case, the less total difference of any matching between the estimated and ground truth positions.

– The cost of each matched pair corresponds to the difference between the estimated position of a robot and its ground truth position, which is the error in the estimation.

For more details about the algorithm, see section 4.1.3

The different metrics were computed/plotted separately for each experiment.

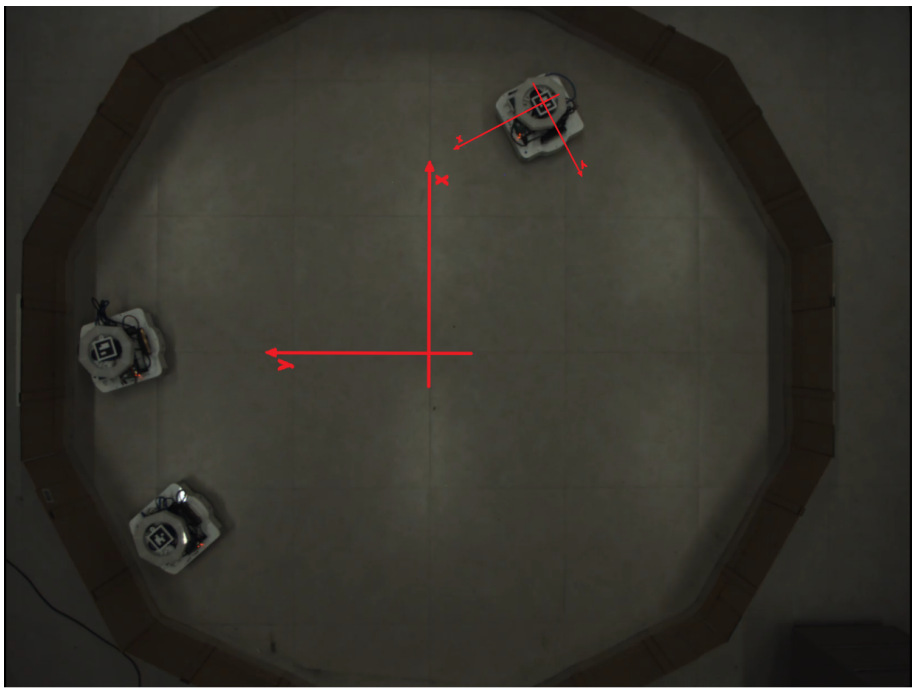


Figure 6.2: Arena and robot frame coordinate systems

6.2.3 Experiments

Three experiments are conducted to compare different scenarios and see if a more complicated scenario decreases the accuracy of the sensor fusion system. The three experiments are:

- A robot will stay still at the edge of the arena while estimating the position of a moving robot. The moving robot will move on a designated path completely inside of the field of view (FOV) of the camera of the still robot. The aim here is to have an experiment where there is a single moving robot that never goes out of the first robot FOV. The experiment duration is set to 1 minute since it is the time taken by the moving robot to complete its full path. The designated path is shown in appendix A.2.
- A robot will stay still at the edge of the arena while estimating the position of two moving robots. This time, the robots will move randomly on the entirety of the arena while trying to not bump into each other. The random walk algorithm used is explained in detail in the appendix A.1. The goal of this experiment is to evaluate the accuracy of a robot at estimating the position of multiple robots that can go in and out of its field of view at any time. The experiment duration was set to 3 minutes to ensure that the moving robots passed enough in the FOV of the static robot.
- Three robots will be moving freely and randomly inside the arena. All the robots are

estimating the position of any other robots that they detect during the experiment. The goal here is to have a more complex scenario where not only the robots to be detected are moving but also the detecting robot. The experiment duration was also set to 3 minutes like the second experiment.

For fairness between the different sensor fusion methods, they must all run on the same data from the robot. It is not possible to run all the methods at the same time on the robot, thus only the most computationally heavy method, the high-level fusion method with LiDAR standalone detection was run online, and the rest was run offline.

The first method was also run offline to see whether there was a difference in results due to hardware limitations. The results were the same and the method did not use the maximum of available resources. Since it was the heaviest method, computationally speaking, it can be concluded that the other methods would give the same results if run online, but to be confirmed.

Since the complementary fusion method corresponds to the partially assisted LiDAR detection method, its results are included in the high-level fusion scheme using this LiDAR detection method.

The results are summarized for each experiment, before a global summary of the three experiments.

6.2.4 Results

To avoid overwhelming this section with redundant plots of all the metrics in each scenario, only the RMSE and error at 0.67 cumulative frequency of error distribution are displayed in the following tables. The interesting plots are shown, but for completeness and interested readers, the full plots are available in appendix C.

It is important to note that these results have the purpose of representing real scenarios and represent real scenarios. They do not represent a theoretical performance of the fusion algorithm where there would be no latency and data from the sensors are fused instantly. In reality, each sensor has its delay in capturing the data and making it available to the fusion algorithm, the fusion algorithm has its delay due to computations, and the fused data produced at time t is a fusion of data from time t-1, which is not necessarily the same time t-1 for all sensors, as the results will show.

I will first analyze the results for each experimental scenario separately and then all the results as a whole in the discussion part.

First experiment

Among the three LiDAR detection methods, the partially assisted one performed the worst by far, having the highest RMSE and error at the 67th percentile (one order of magnitude above the two other methods as shown in Figures 6.3 and 6.4). The standalone and fully assisted LiDAR methods have similar RMSE but the standalone method gives a lower 67th percentile error as shown in Figure 6.4.

For all LiDAR detection methods, no significant improvement is seen neither in RMSE (Figure 6.5) nor 67th percentile error (Figure 6.6) values for the fused data by comparison to the camera data.

The fully assisted LiDAR method did slightly worsen the fused results but not the partially assisted method. This is because the latter method had such a high error, that most of

its estimated positions were not matched with camera detections.

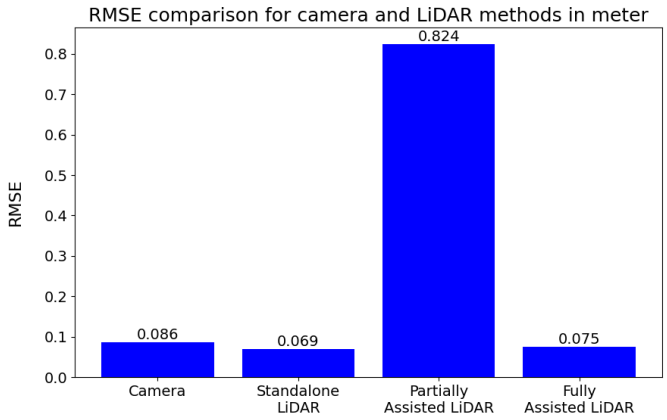


Figure 6.3: Experiment 1: RMSE comparison for camera and LiDAR methods in meter.

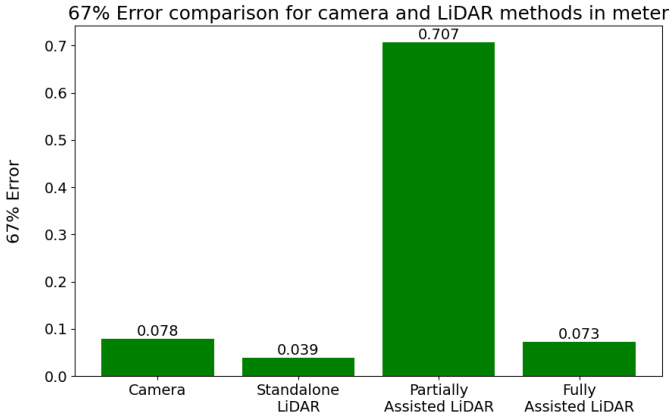


Figure 6.4: Experiment 1: Error at the 67th percentile comparison for camera and LiDAR methods in meter.

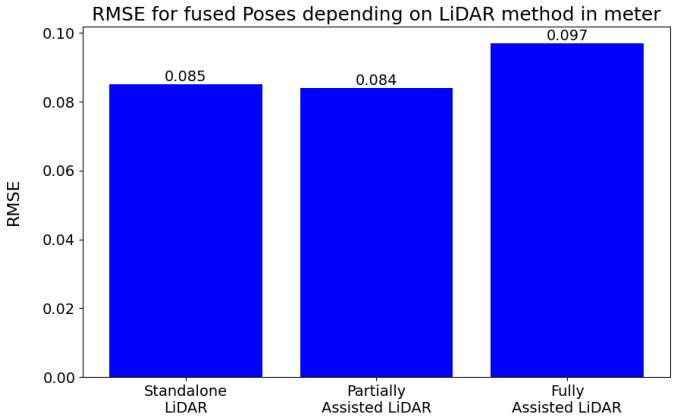


Figure 6.5: Experiment 1: RMSE for fused poses depending on LiDAR method in meter.

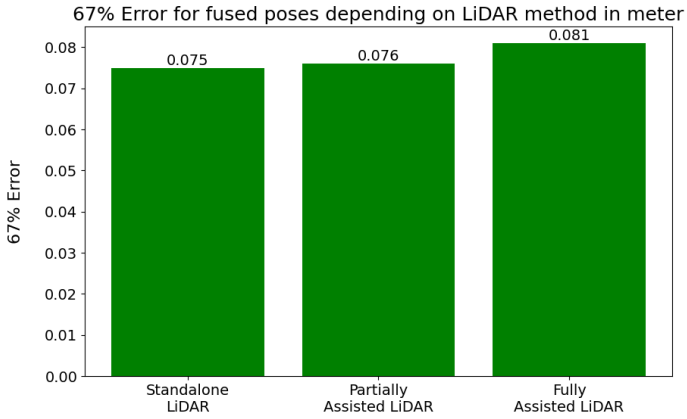


Figure 6.6: Experiment 1: Error at the 67th percentile for fused poses depending on LiDAR method in meter.

Second experiment

Similar to the first experiment, among the three LiDAR detection methods, the partially assisted one performed the worst in terms of RMSE and 67th percentile error as shown in Figures 6.7 and 6.8.

The standalone LiDAR detection method had similar RMSE to the two other LiDAR methods but way lower 67th percentile error, even lower than the camera's. This means that most of the estimated positions were really good with fewer that were worse than the estimated positions by the camera.

No improvement is seen in fused data RMSE and 67th percentile error for all the LiDAR detection methods compared to camera results in Figures 6.9 and 6.10.

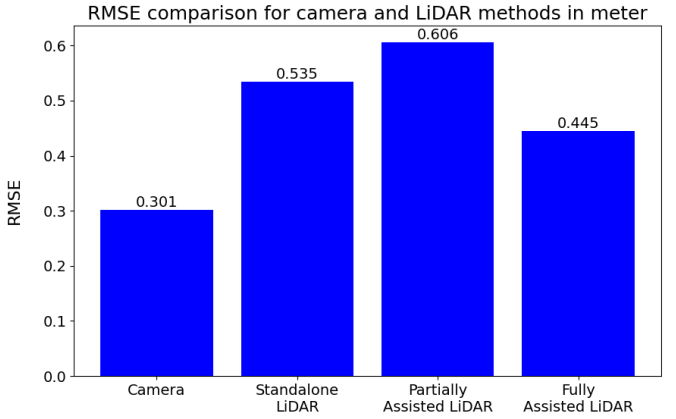


Figure 6.7: Experiment 2: RMSE comparison for camera and LiDAR methods in meter.

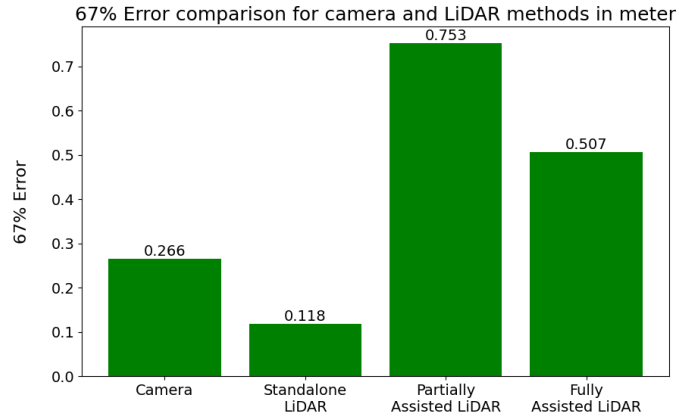


Figure 6.8: Experiment 2: Error at the 67th percentile comparison for camera and LiDAR methods in meter.

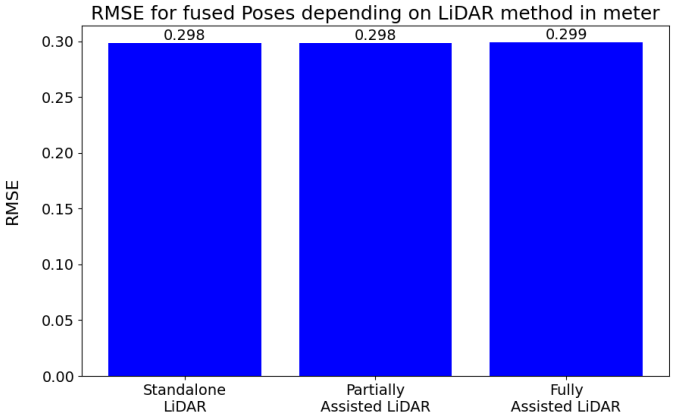


Figure 6.9: Experiment 2: RMSE for fused poses depending on LiDAR method in meter.

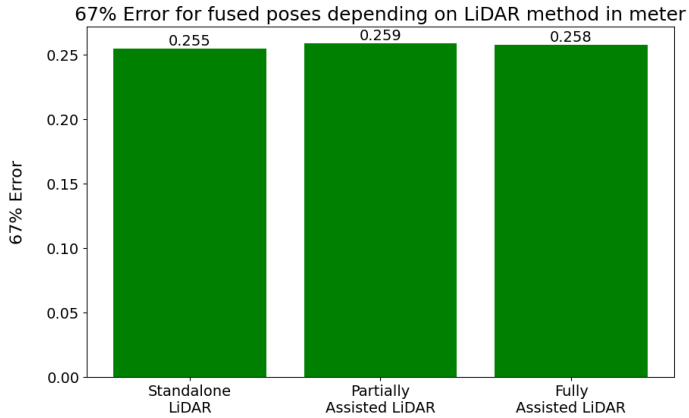


Figure 6.10: Experiment 2: Error at the 67th percentile for fused poses depending on LiDAR method in meter.

Third experiment

It is notable in this third experiment that having the robot doing detections while moving around in the arena like the other robots increased the RMSE of the camera as shown in

Figure 6.11, but it also decreased the error at the 67th percentile for the camera compared to experiment 2 (Figure 6.12. The high RMSE can be explained by the fact that the robot moving and rotating on itself, has introduced false positive detections of the other robots by the camera which is penalized by the RMSE. On the other hand, the low 67th percentile error is due to the robot moving around being closer to the other robots that it detects, rather than at the edge of the arena in experiment 2.

The worst-performing method in this experiment is by far the fully assisted LiDAR detection method, having very high RMSE and 67th percentile compared to the other two methods (Figures 6.11 and 6.12) and even worsening the fused data 67th percentile as shown in Figure 6.14.

It is difficult to tell which method performed better individually between the standalone LiDAR detection and the partially assisted one in this experiment, but the latter did improve slightly the fused data RMSE (Figure 6.13) but no improvement over the other method in Fused data 67th percentile error (Figure 6.14).

Overall, there was also no significant improvement that would make the fused data better to use than camera data in terms of RMSE and 67th percentile error for any method.

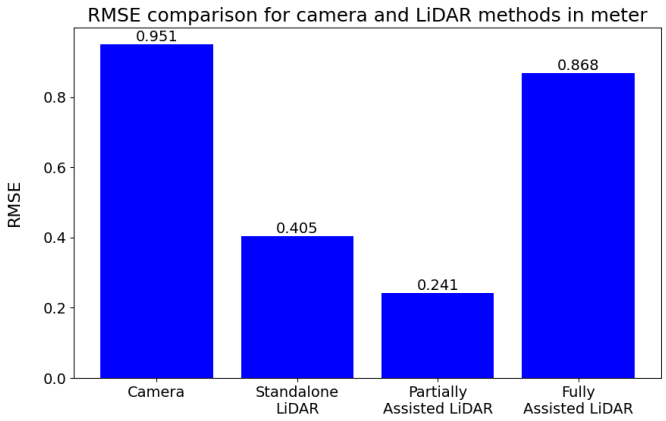


Figure 6.11: Experiment 3: RMSE comparison for camera and LiDAR methods in meter.

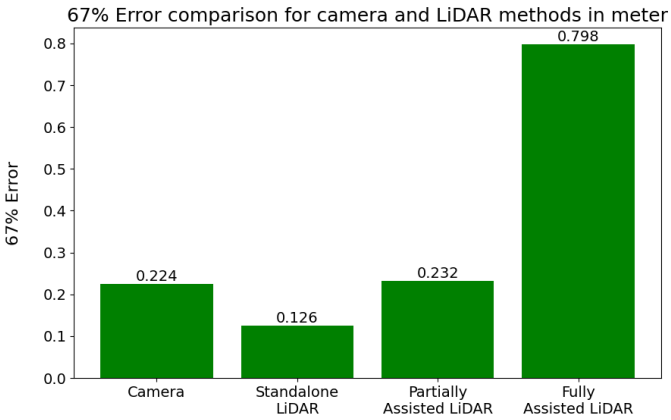


Figure 6.12: Experiment 3: Error at the 67th percentile comparison for camera and LiDAR methods in meter.

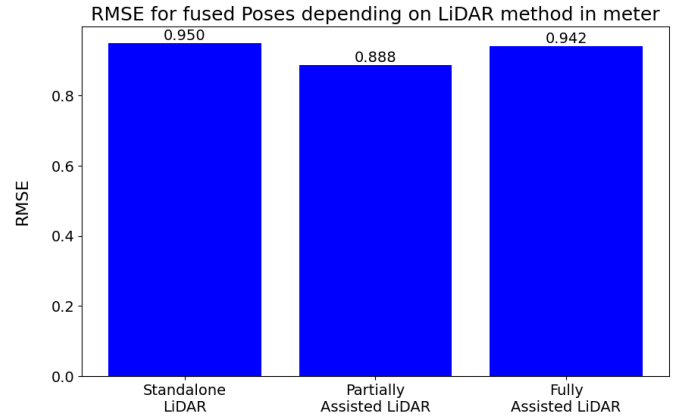


Figure 6.13: Experiment 3: RMSE for fused poses depending on LiDAR method in meter.

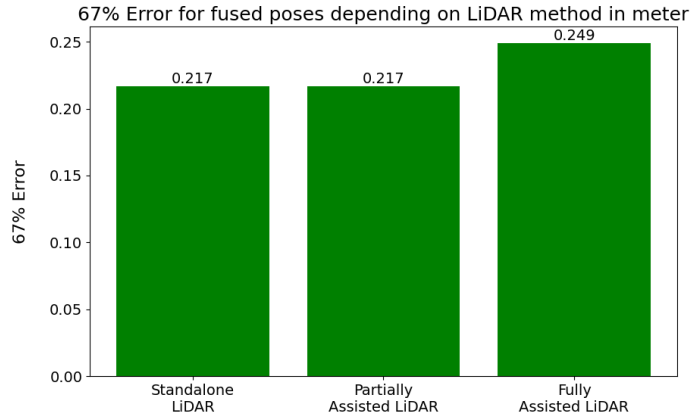


Figure 6.14: Experiment 3: Error at the 67th percentile for fused poses depending on LiDAR method in meter.

6.2.5 Experiments results discussion

The goals of this discussion are to try to determine which LiDAR detection method performed the best and should be kept for further experimentation and to see how the current algorithm can be improved to bring better results if possible.

I will go through the observations made for the three experiments and explain what they imply to improve the algorithm.

No significant improvement in fused data was observed

The fused data didn't show any significant improvement in any experiment compared to camera data, despite having LiDAR giving better estimations of the positions of the robot with the standalone method in the three experiments and with the partially assisted method in the third method. This implies that the estimated variance for the LiDAR in section 4.1.2 was overestimated and should be lowered to give more weight to the LiDAR in the fusion.

Kalman filter occasional delay

I observed during experiment 1 that the movement of the robot when it rotates on itself and takes turns according to the fused positions, has a bit of delay compared to the positions given by the camera and the LiDAR. This can either mean that the fusion computation takes too long or that the variances in the measurement covariance matrix of the Kalman filter estimated in section 4.1.2 are too high. Since I didn't observe this delay when the robot was moving straight, I assume that the measurement variances of both the camera and the LiDAR should be lowered to give more weight to the measures than the prediction of the filter to avoid the occasional delay due to the divergence between measured data and the prediction model.

Camera false positive detections

I assumed in the design of the high-level fusion algorithm that the camera had a very low false positive detection rate and that all of its detection should be trusted and kept, whether they were matched with detections from the LiDAR. This was true for experiments 1 and 2 but not for experiment 3. While this problem might be solved later by improving the neural network used on the camera, it is still worth it to have an alternative by not keeping the detections from the camera that were not matched by the LiDAR. This can be turned on/off by simply changing one parameter in the launching script of the software.

Assisted LiDAR detections poor performance

Another interesting observation is the low performance of the assisted LiDAR methods even in static scenarios in experiments 1 and 2, seeing how the standalone method performed often better than the camera, those two assisted methods should at least give approximately the same result as the camera.

The explanation is that the camera has a higher delay than LiDAR due to the need to perform neural network inference on its images to detect the robots. The consequence of the assisted LiDAR methods is that the LiDAR needs to wait for the camera data (either the bounding box for the partially assisted or the position for the fully assisted method) to search for robots in the directions given by the camera, thus looking for robots of the past that have already moved away in most cases.

To solve the issue, the LiDAR needs to keep scans of the past (depending on the actual delay between the camera and the LiDAR) and search for the detected robots from the camera in those scans.

This would have been the perfect solution but since the standalone method provides already better results in estimating the robot's positions than the camera and the assisted LiDAR methods would have an extra delay than the standalone method, it is better to use the standalone method in the fusion algorithm to compensate for the delay of the camera rather than adding an unnecessary delay to the LiDAR detections.

The problem of the camera's detection delay for the assisted LiDAR methods can also be observed in Figure 6.15 where no observation of robots where made for the partially assisted LiDAR detection outside of the FOV of the camera (since it is based on bounding box from the camera) while the camera observed robots even when the robot was looking at the opposite direction due to the delay and thanks to the robot rotating quickly on itself in experiment 3 making it easily observable.

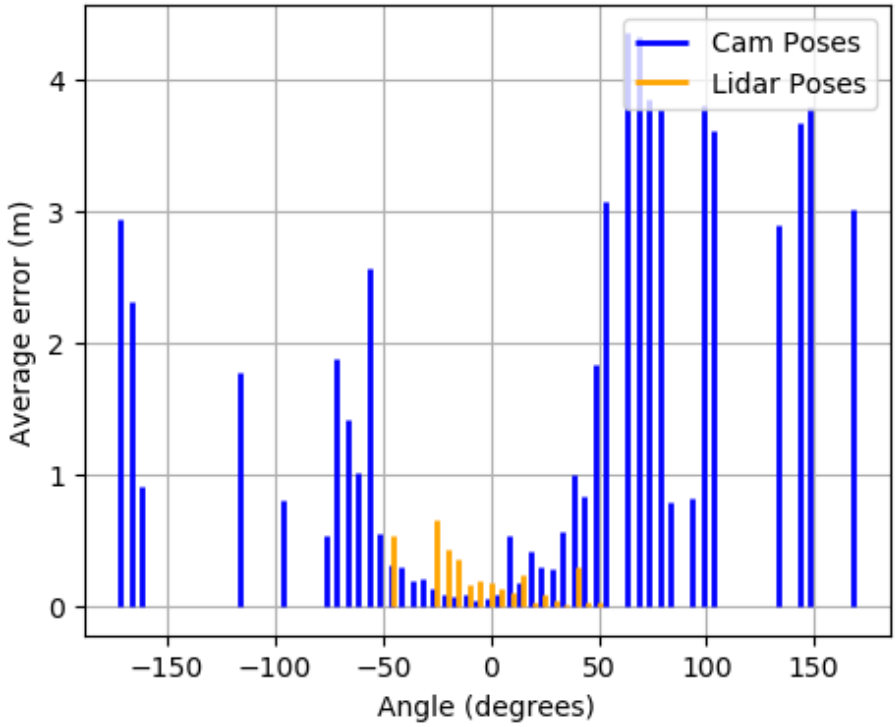


Figure 6.15: Experiment 3 with partially assisted LiDAR detection: Error over angle to target (angle from the ground truth data) for the camera and LiDAR.

6.2.6 Fusion algorithm improvements

Based on the problems observed and the possible solutions that I discussed in the previous section, I decided to only continue with the standalone method as the detection method for the LiDAR in the fusion algorithm, improve the fusion algorithm, and test it again with the recorded data of the three experiments.

Improvements made to the algorithm:

- Lowered the errors on the camera and the LiDAR used to compute the measurement covariance matrix of the Kalman filter. A good improvement was found when reduced by a factor of ten for the camera and 20 for the LiDAR.
- The detections made by the camera are no longer kept anymore if not also matched by the LiDAR.

6.2.7 Improvements results

The algorithm improved in both RMSE and 67th percentile error metrics compared to the initial implementation in all three experiments as shown in Figures 6.16 6.17 6.18.

The results also show that the RMSE of the fused data is lower than both the camera and the LiDAR data, and the 67th percentile error is similar to the LiDAR, which is the lowest of the 2 sensors.

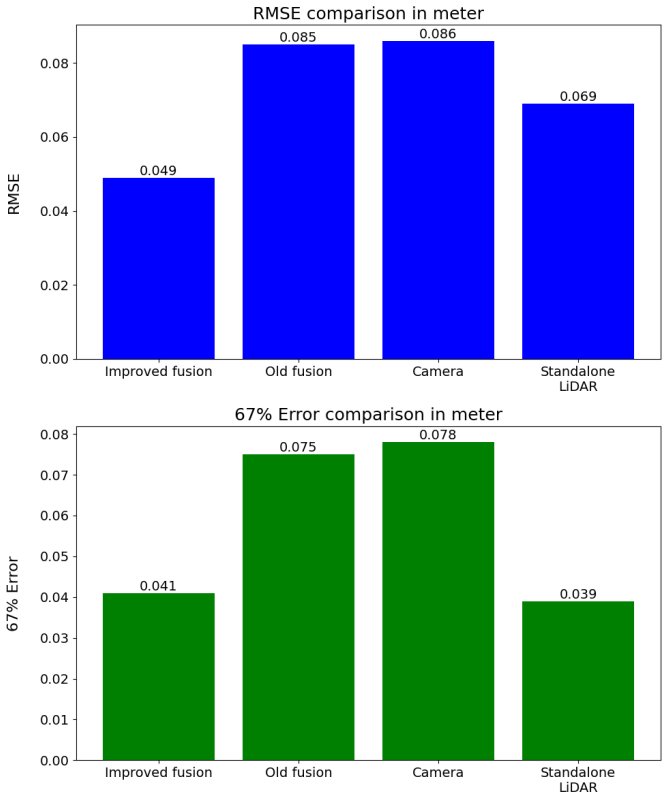


Figure 6.16: Experiment 1 comparison improved fusion, old fusion, camera, and LiDAR

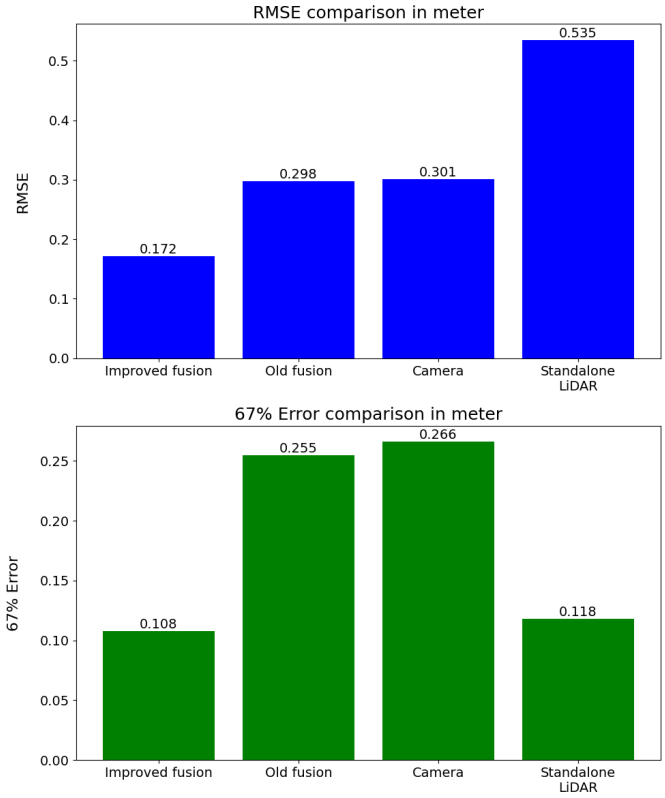


Figure 6.17: Experiment 2 comparison improved fusion, old fusion, camera, and LiDAR

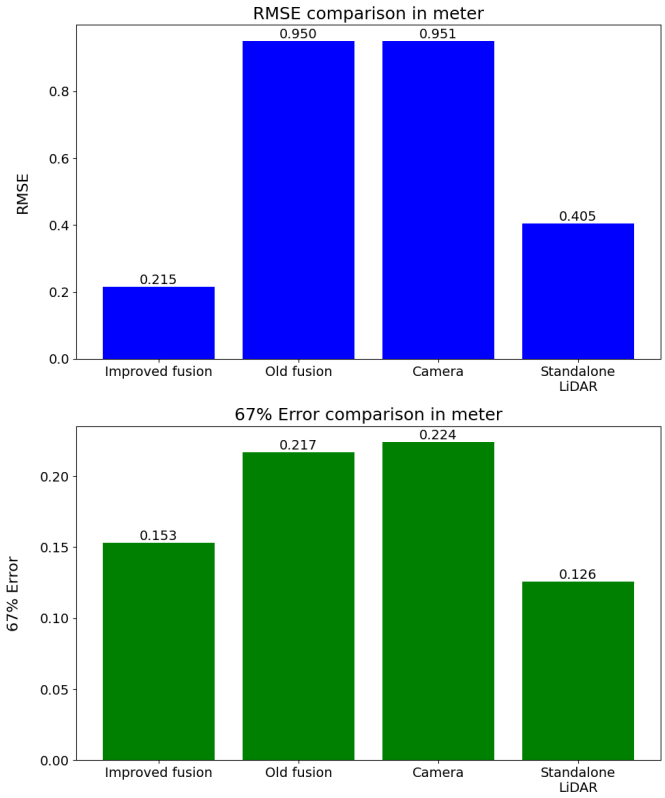


Figure 6.18: Experiment 3 comparison improved fusion, old fusion, camera, and LiDAR

6.2.8 Faulty sensors experiments

Two additional experiments were done to assess the resiliency of the fusion algorithm when low-quality data is introduced in the system from first, one sensor, and then both sensors.

The first one is with the camera performing a lot of detections of false positive robots in addition to the true positive ones, the cumulative histogram error distribution of the data from the camera, LiDAR, and the fused data are shown in Figure 6.19.

The fusion algorithm still performs well with lower RMSE and lower 67th percentile error than both of the sensors as shown in Figure 6.20

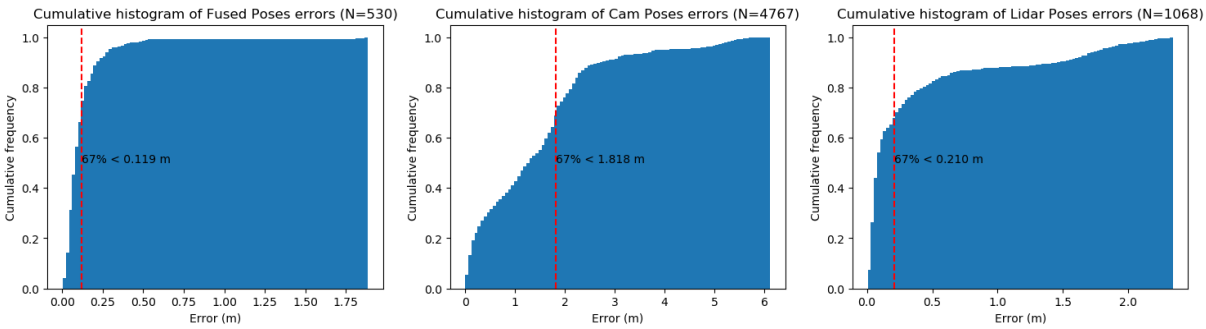


Figure 6.19: Faulty camera experiment's cumulative histogram error distributions of the data from the camera, LiDAR, and fused data.

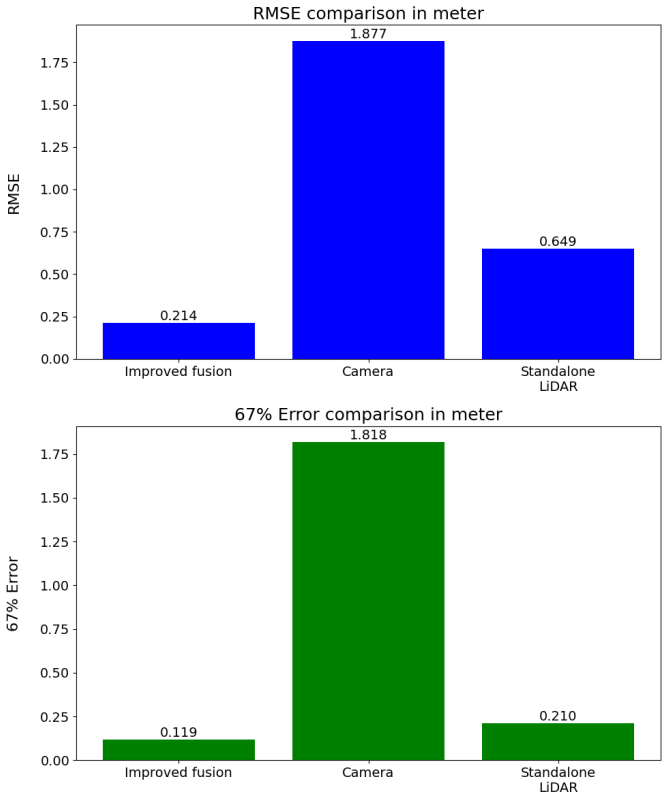


Figure 6.20: Faulty camera experiment's comparison of improved fusion, camera, and LiDAR

The second experiment was conducted with a faulty LiDAR in addition to a faulty camera. The LiDAR's clustering algorithm detects false positive robots after increasing the maximum cluster radius threshold. The cumulative histogram error distributions are shown in Figure 6.21.

The fusion algorithm gives a slightly lower RMSE than the lowest RMSE of both sensors and a much lower 67th percentile error than both sensors as shown in Figure 6.22.

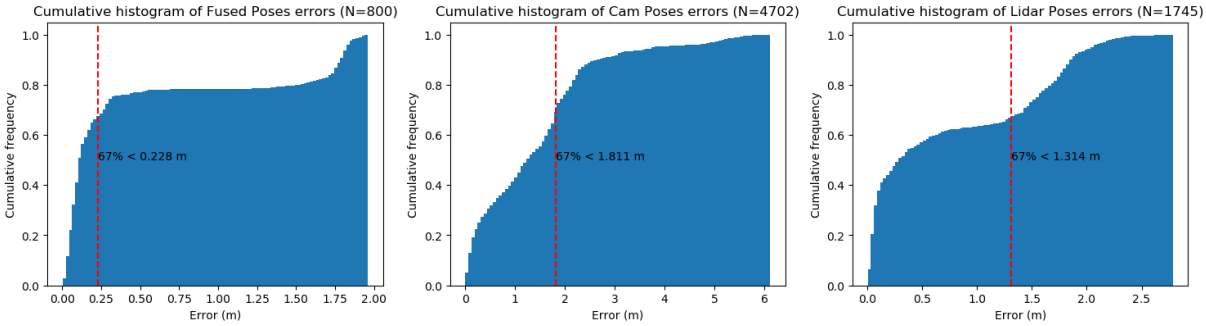


Figure 6.21: Faulty sensors experiment's cumulative histogram error distributions of the data from the camera, LiDAR, and fused data.

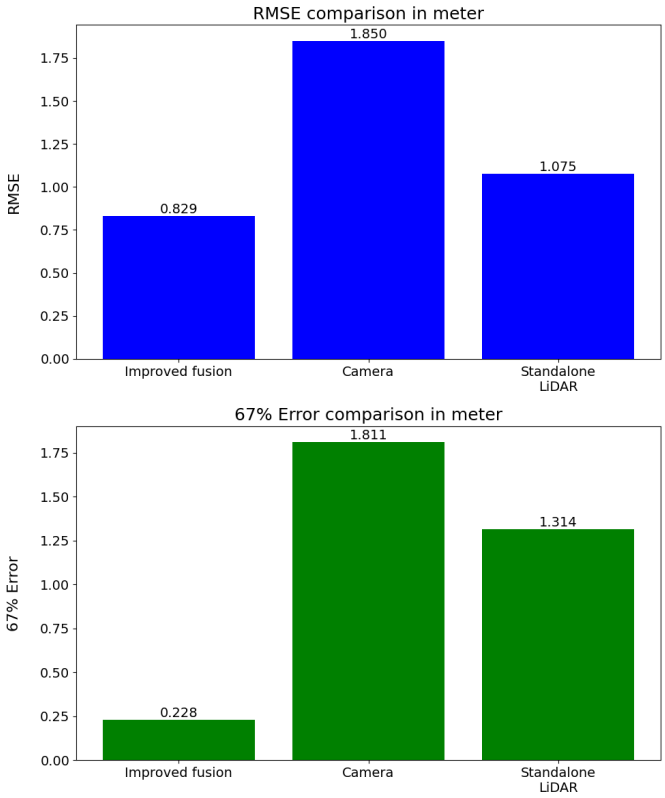


Figure 6.22: Faulty sensors experiment's comparison of improved fusion, camera, and LiDAR

6.3 Possible applications

There are a lot of swarm robotics experiments that can benefit from the robots being aware of the other robots in their surroundings and knowing their positions. A visual example of a robot detecting and computing the position of two other robots using the sensor fusion algorithm developed in this work is shown in Figure 6.23.

One experiment that I tried is doing random walk exploration mapping with the robots [57]. With the previous robots mapping with their proximity sensors, the maps were very noisy due to the robots detecting each other as obstacles while moving around to explore. One possible solution is to use the newly acquired capability of the robots to know whether the obstacles around them are robots or not, to denoise the LiDAR's reading, and filter out the detected robots from them in such a way that the space they occupy is considered free space instead of an obstacle.

I conducted an experiment with an obstacle in the arena to ensure that the obstacle is not filtered out with the robots and see whether the mapping improves with this technique. The different maps made by the robots and the merged maps with and without denoising are shown in appendix B. Unfortunately, nothing can be concluded from this experiment since the original map without filtering out the robots is already very good and with little noise compared to what was expected initially based on the mapping experiments with the previous robots.

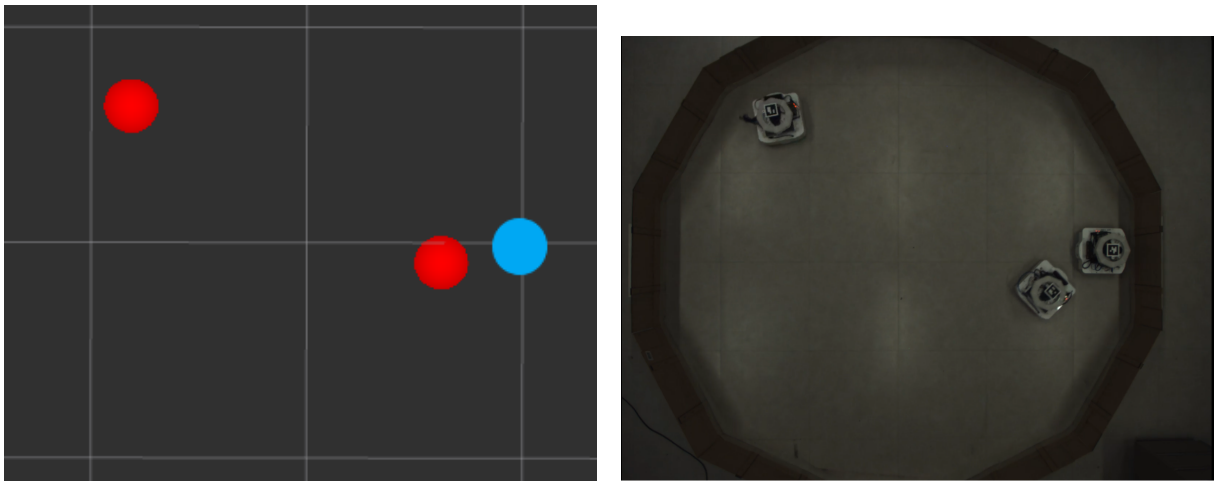


Figure 6.23: Real-time position estimations of 2 robots, the blue dot is the origin of the detecting robot frame and the two red dots are the detected robots.

Chapter 7

Conclusion

The main objective of this Master's thesis was to design and implement a sensor fusion method to enhance the perception capabilities of the robots used for swarm robotics experiments at the IRIDIA research laboratory.

The literature review on the role of sensors in swarm robotics and sensor fusion, in general, showed that many techniques and algorithms exist to fuse data from multiple sensors and the choice depends on the sensors used in the fusion scheme.

I analyzed the different perception sensors present on the robotic platform to determine which could be added to the fusion scheme to help the robot detect and estimate the positions of surrounding robots and obstacles. The camera and the 2D 360° LiDAR were selected for this work.

Based on the sensor analysis and literature review, I developed two sensor fusion methods:

- A complementary fusion algorithm where the camera detects and the LiDAR estimates the positions of the detections.
- A high-level and modular fusion algorithm where both the sensors detect and estimate the positions of the targets. This method has 3 variants based on the LiDAR detection method:
  - Standalone LiDAR detection with Euclidean clustering.
  - Partially assisted LiDAR detection (Complementary fusion).
  - Fully assisted LiDAR detection.

I conducted experiments in IRIDIA's arena to assess the different fusion methods in real scenarios. The experiments showed that the High-level fusion with standalone LiDAR detection performed the best and was selected for further improvements and experimentation.

The improved fusion algorithm outperformed the previous ones and gave better results than both the sensors taken individually, which was the initial goal of this Master's thesis. Additionally, the algorithm showed great resiliency in cases where one or both sensors were faulty.

Future development

Camera delay

The camera had a significant delay (approximately 150ms) compared to the LiDAR when it came to detecting and providing positions of robots and obstacles. This could be reduced by compiling the neural network to use all the cores available on the camera's processing unit, instead of the current where it is compiled for half of them and runs twice in parallel to increase framerate.

Even with half the delay, another solution that can improve the position estimation of the camera is to use the odometry of the robot but from the past (corresponding to the delay) to replace correctly the detected robots and obstacles in the fixed frame of the robot's odometry.

Larger training dataset

The neural network used for camera detection needs to be retrained and improved further to solve the issue of false positives appearing when the robot rotates on itself. This could be done by building a larger and more diversified dataset with more background images.

Better LiDAR alignement

The LiDAR needs to be oriented correctly and to stay parallel to the XY plane to ensure that it can detect the LiDAR of other robots, even at further distances.

Code optimization

Although the current implementation of the algorithm did not exceed the CPU resources available on the Raspberry Pi dedicated to the camera and no latency problem was encountered, it might be wise in the future, to rewrite some computationally heavy parts of the code into a more efficient language to free up resources and reduce latency further.

Appendices

Appendix A

Robot path planning

A.1 Random walk algorithm

Each robot will plan its path individually and on the spot. The robot will use its proximity sensors to detect any obstacles or other robots on its path and change direction in function of that. Since the robot can only go straight forward and turn (it can go backward but not used but it cannot go left or right without rotating to change direction), the only proximity sensors used to detect obstacles are the four in the front. The algorithm is simple:

- If no obstacle is detected, goes straight
- An obstacle is detected when the reading from any of the four front sensors gives a distance less than 0.4 m (in fact the robot will continue a bit its movement before actually detecting the obstacle, so the distance threshold must be a bit higher than the expected stopping range):
  - If the obstacle is detected by a proximity sensor on the front left, the robot will choose to go right and vice versa. If obstacles are detected by both, it will choose the turning direction randomly.
  - It will turn for a random amount of 0.2 and 0.8 seconds, with the maximum duration representing approximately half a turn.

A.2 Pre-determined path

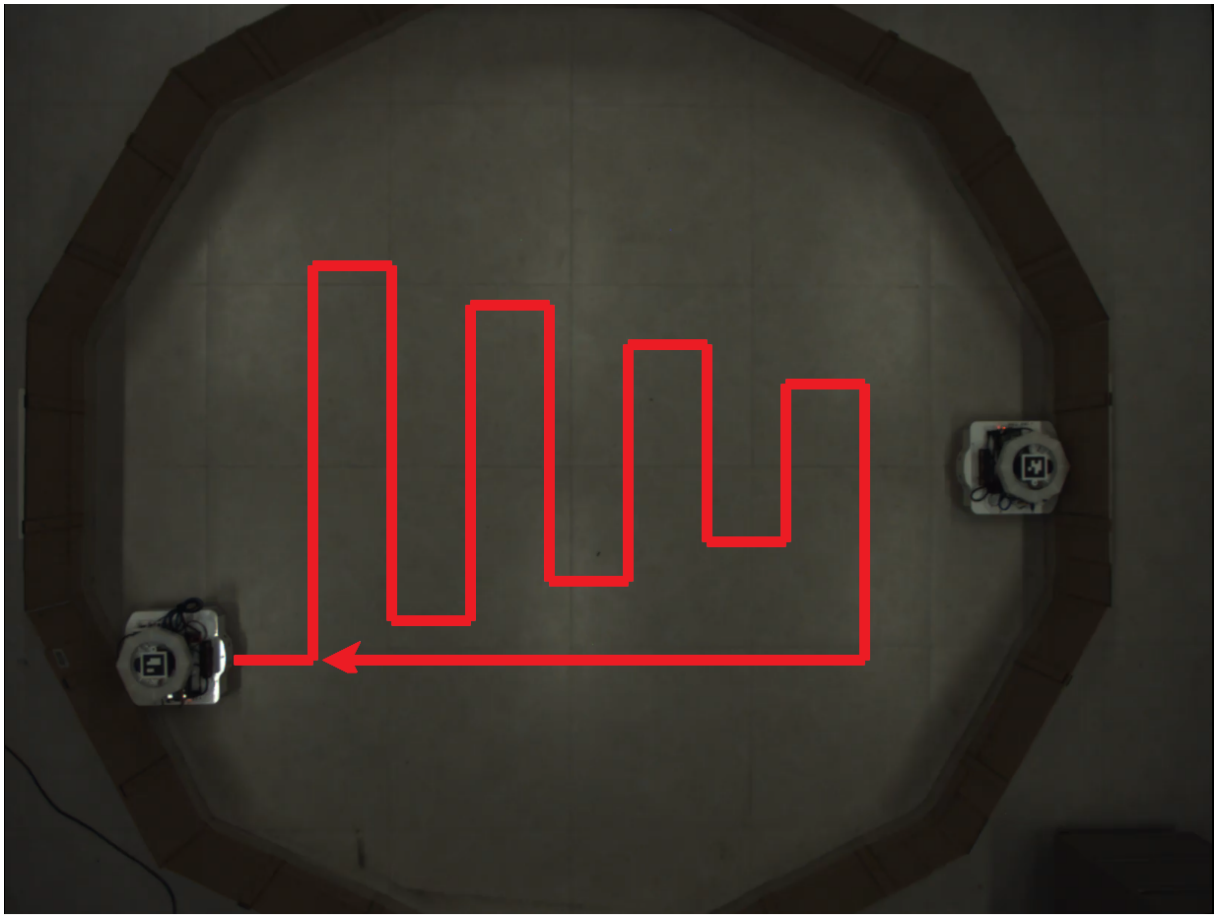
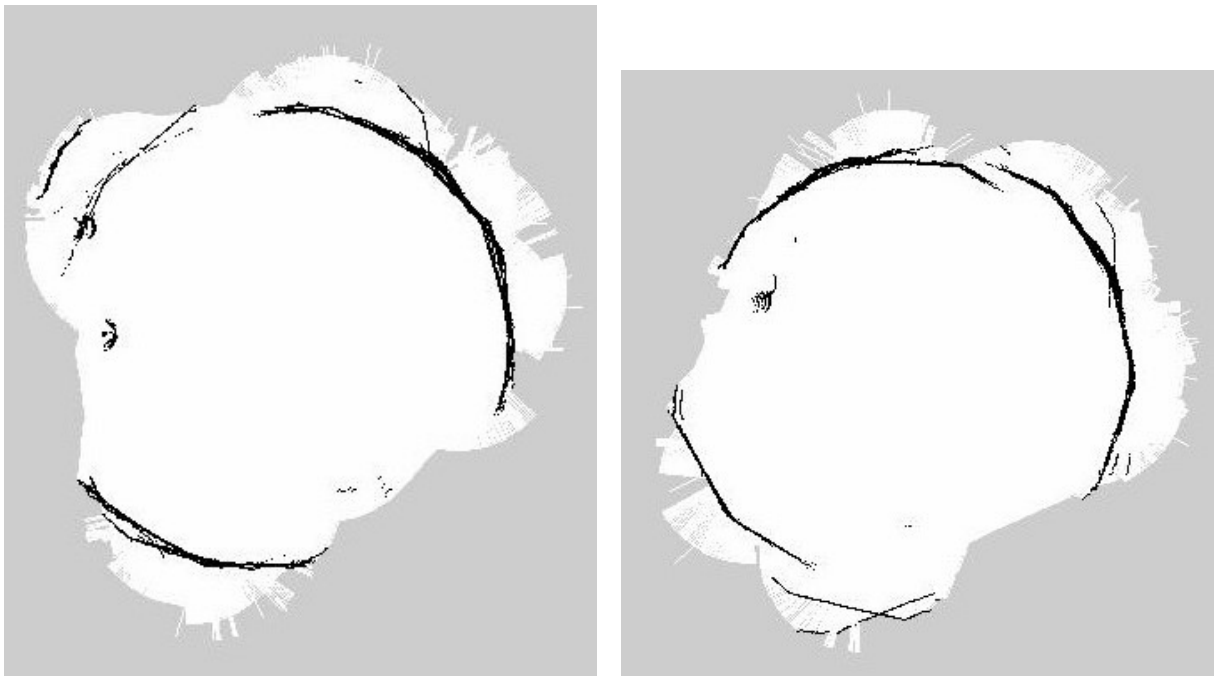


Figure A.1: Path taken by the moving robot in the first experiment.

Appendix B

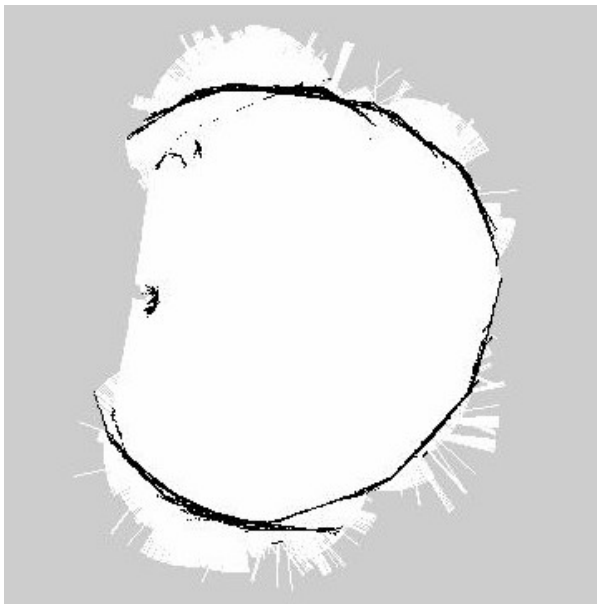
Mapping experiment

Mapping was done with the Gmapping ROS package[1] and the merging of maps was done with multirobot$_{m}ap_{m}ergeROSpackage$[2].

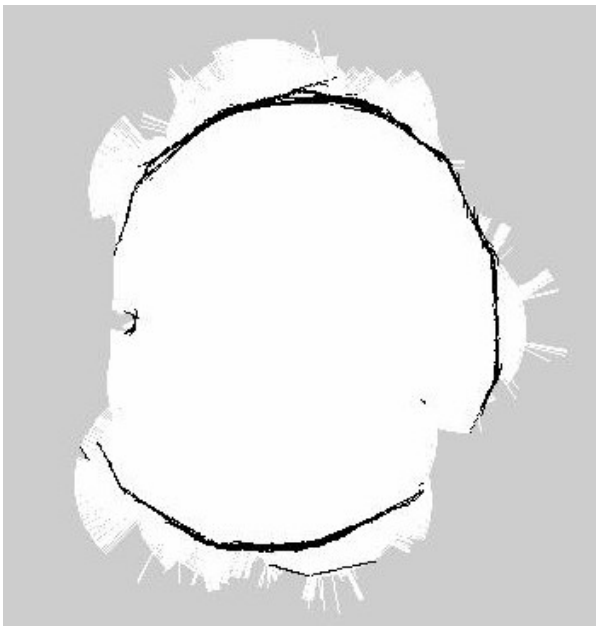


(a) Robot 1 map without filtering other robots (b) Robot 1 map with filtering other robots

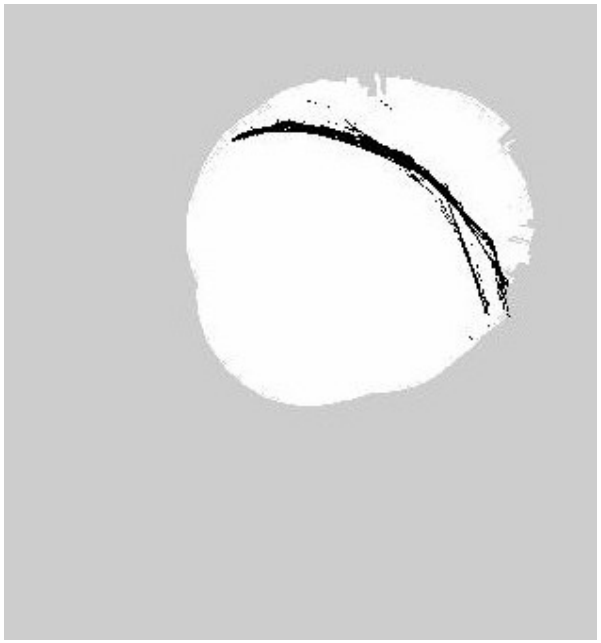
[1] `http://wiki.ros.org/gmapping`
[2] `http://wiki.ros.org/multirobot_map_merge`



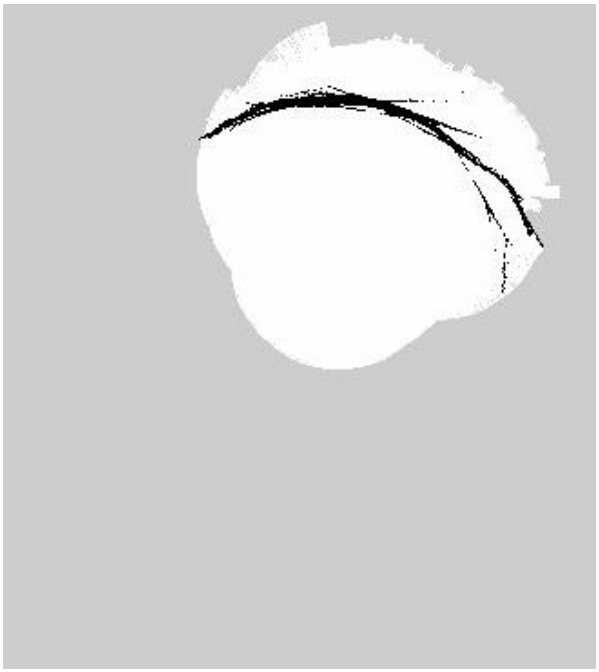
(a) Robot 2 map without filtering other robots



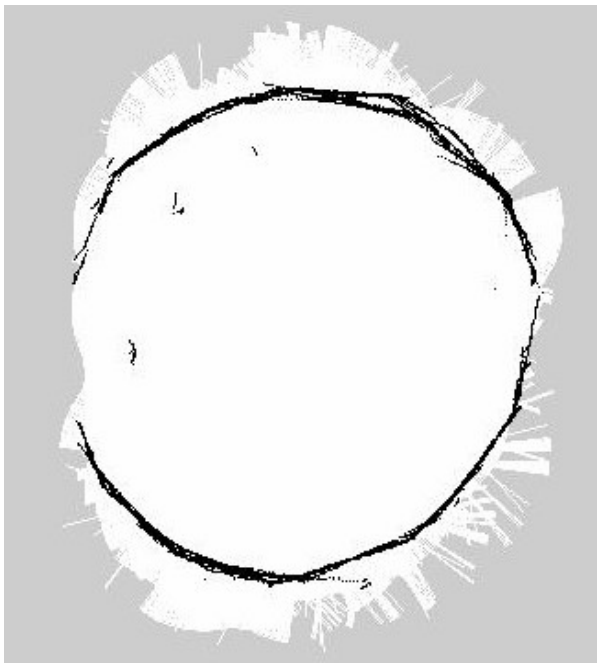
(b) Robot 2 map with filtering other robots



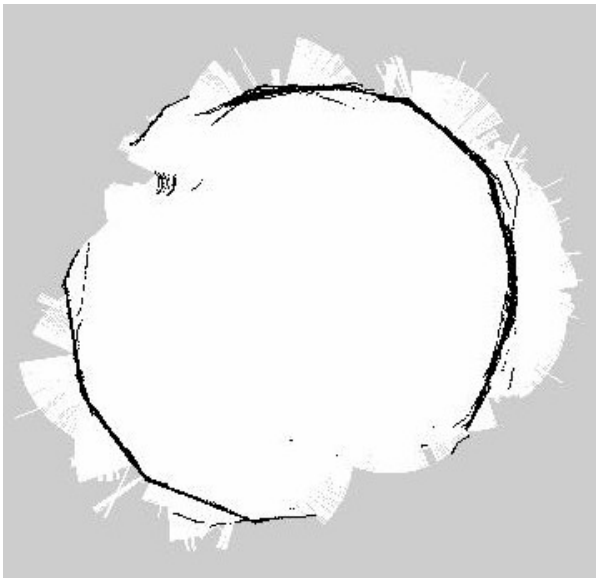
(a) Robot 3 map without filtering other robots



(b) Robot 3 map with filtering other robots



(a) Merged map without filtering other robots



(b) Merged map with filtering other robots

Appendix C

Sensor fusion methods evaluation plots

C.1 First experiment

C.1.1 High-level fusion with standalone LiDAR detection

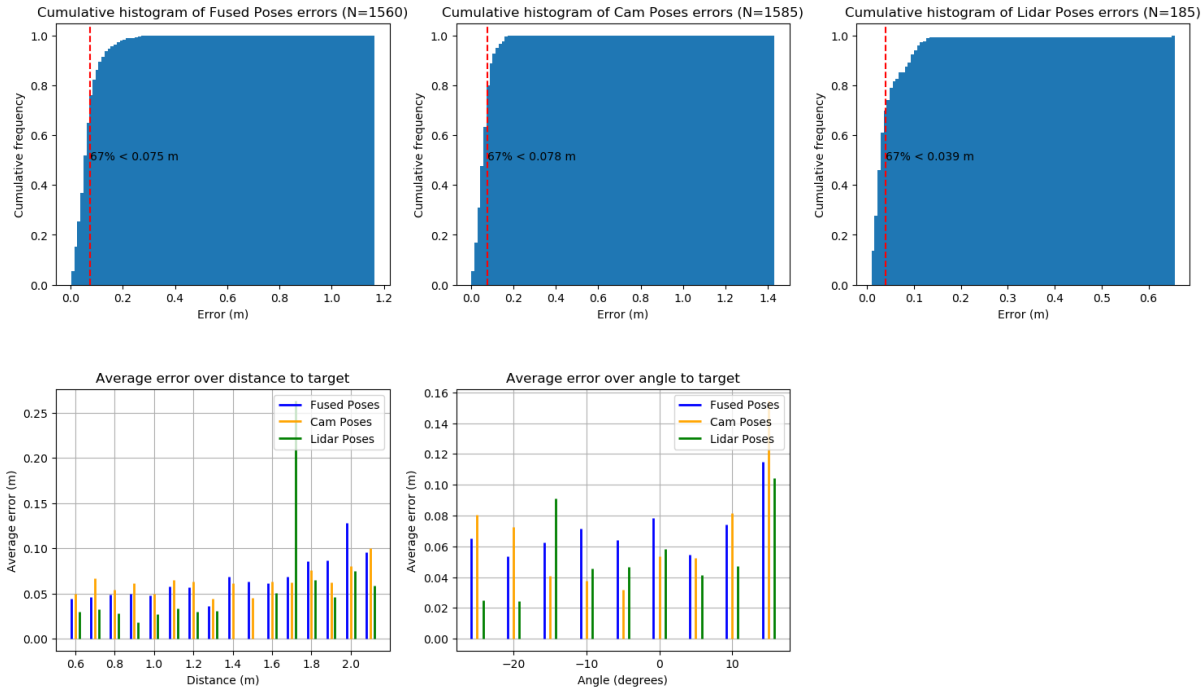


Figure C.1: First experiment, High-level fusion with standalone LiDAR detection: Errors Evaluation metrics.

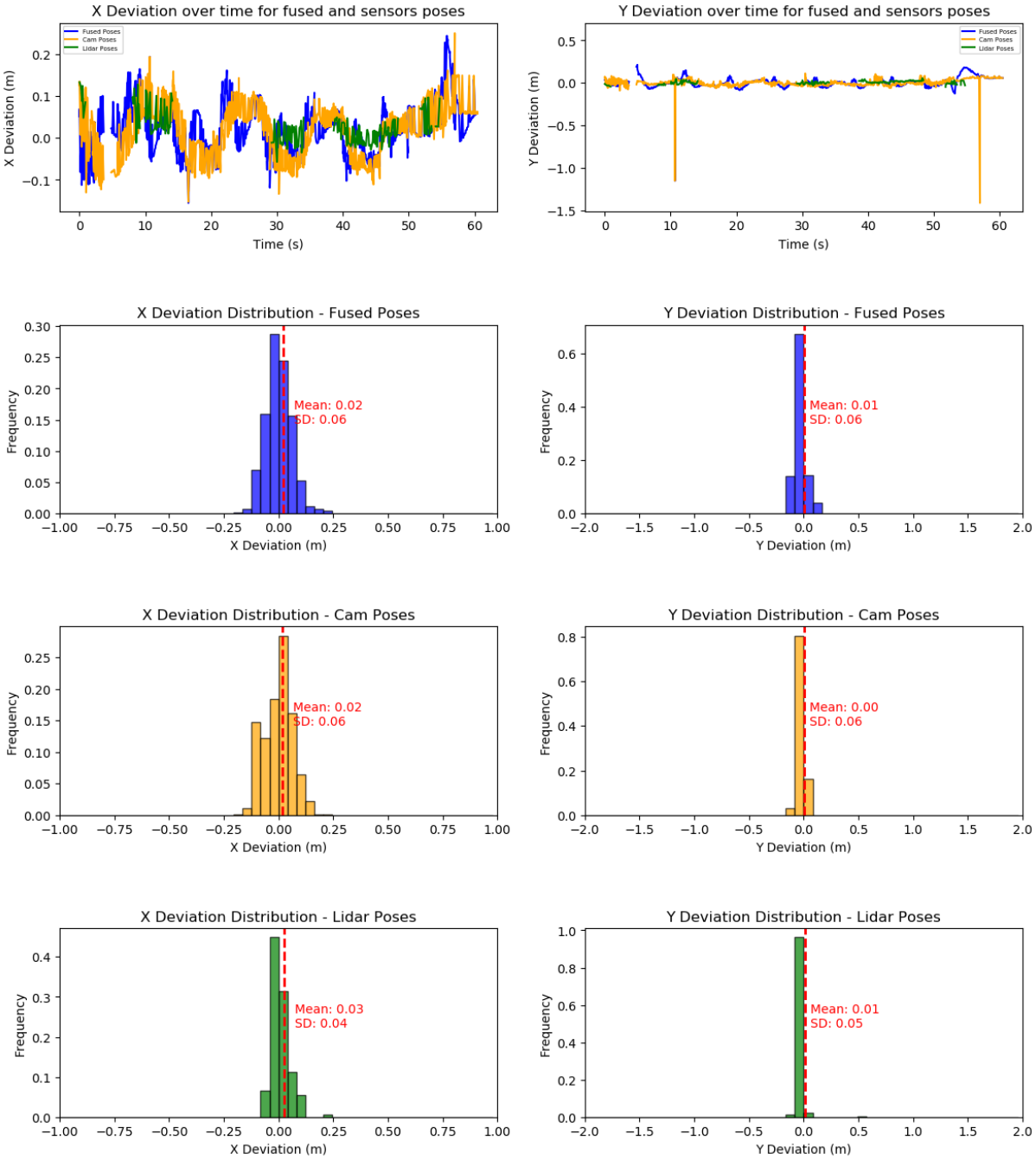


Figure C.2: First experiment, High-level fusion with standalone LiDAR detection: X and Y deviations metrics.

C.1.2 High-level fusion with partially assisted LiDAR detection

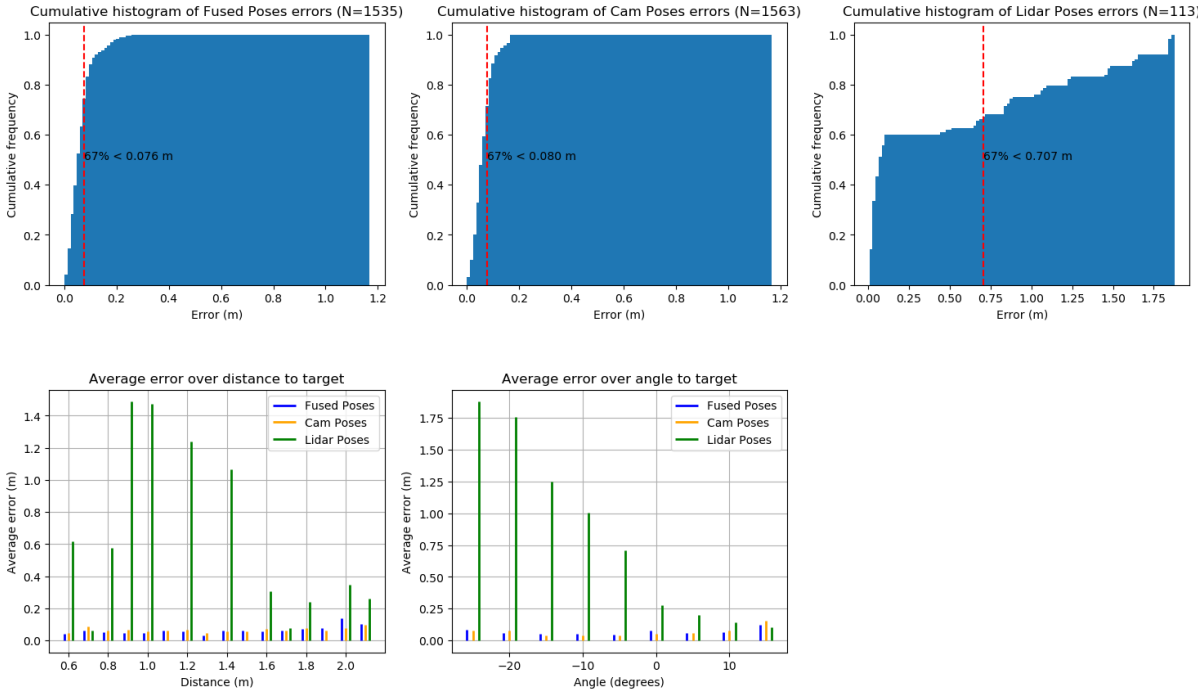


Figure C.3: First experiment, High-level fusion with partially assisted LiDAR detection: Errors Evaluation metrics.

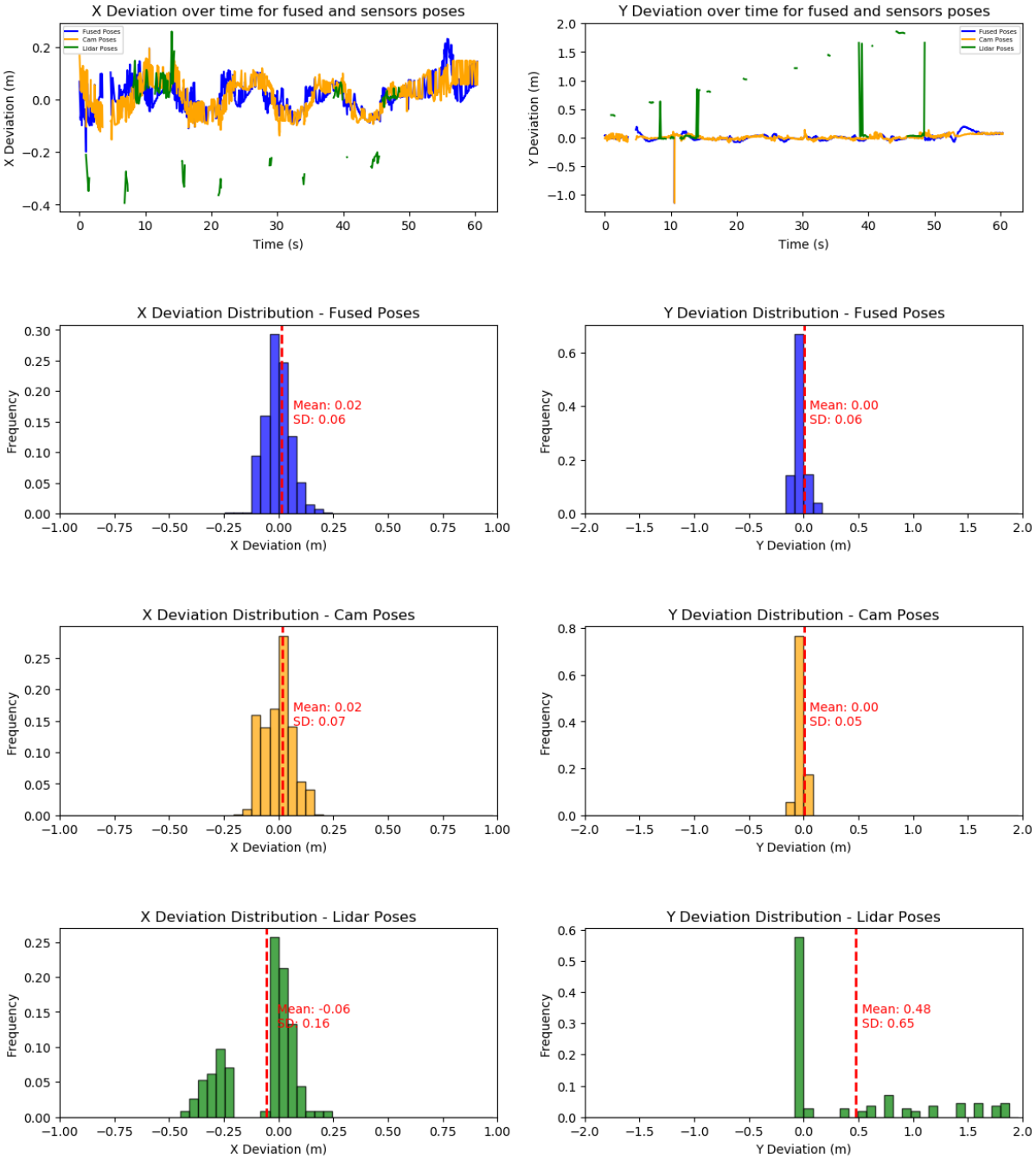


Figure C.4: First experiment, High-level fusion with partially assisted LiDAR detection: X and Y deviations metrics.

C.1.3 High-level fusion with fully assisted LiDAR detection

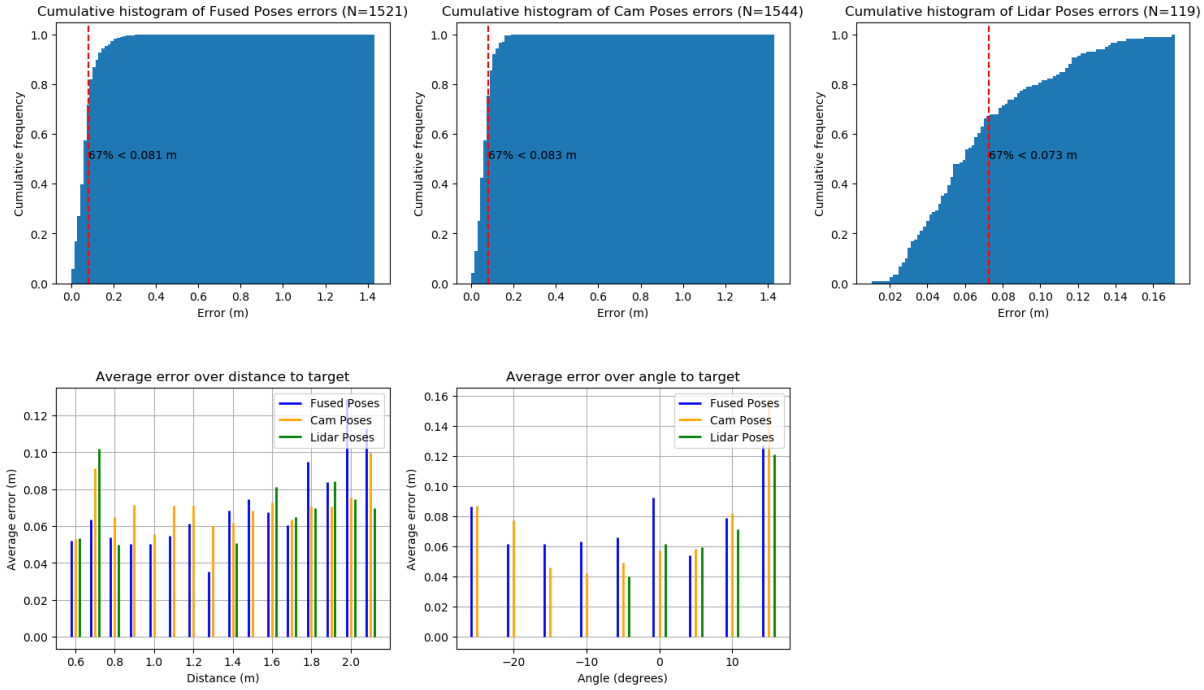


Figure C.5: First experiment, High-level fusion with fully assisted LiDAR detection: Errors Evaluation metrics.

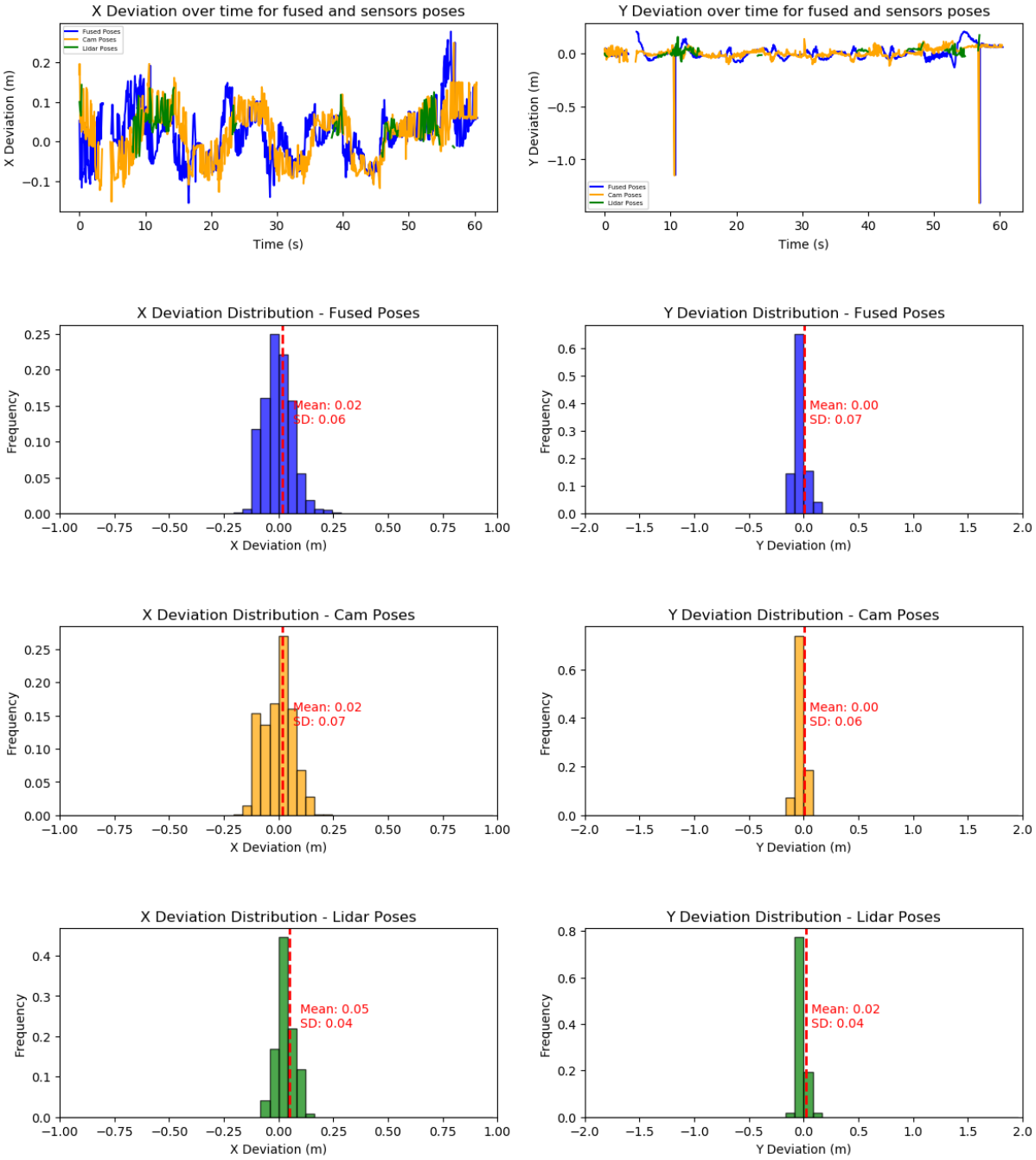


Figure C.6: First experiment, High-level fusion with fully assisted LiDAR detection: X and Y deviations metrics.

C.2 Second experiment

C.2.1 High-level fusion with standalone LiDAR detection

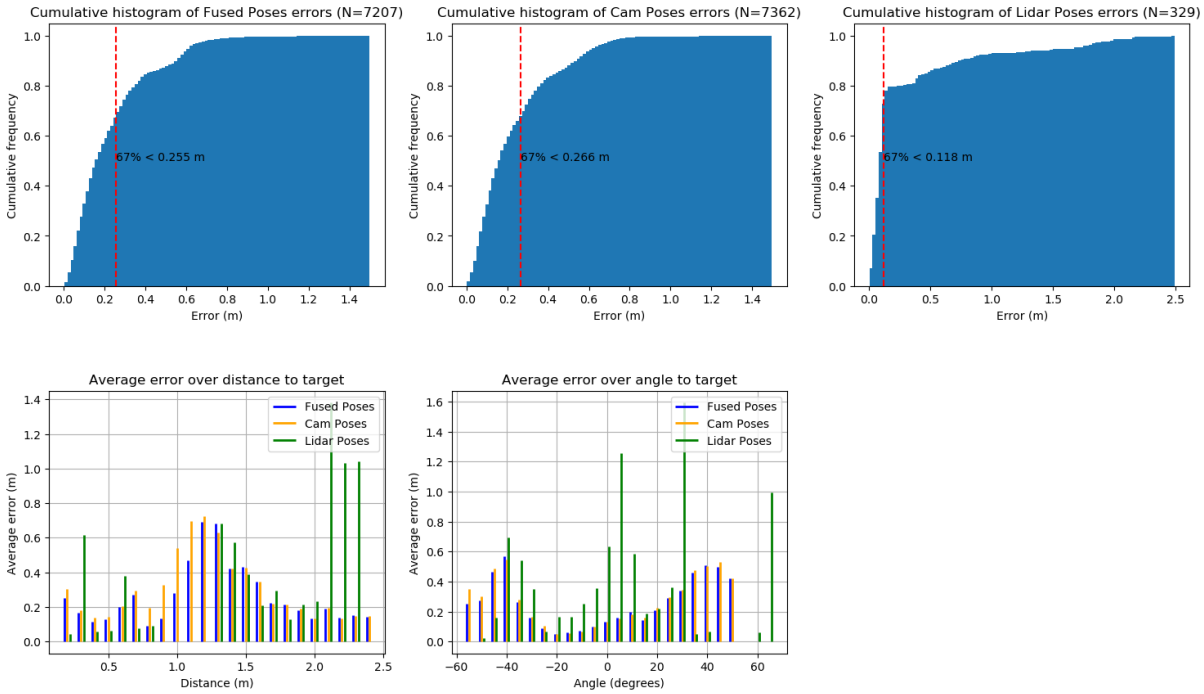


Figure C.7: Second experiment, High-level fusion with standalone LiDAR detection: Errors Evaluation metrics.

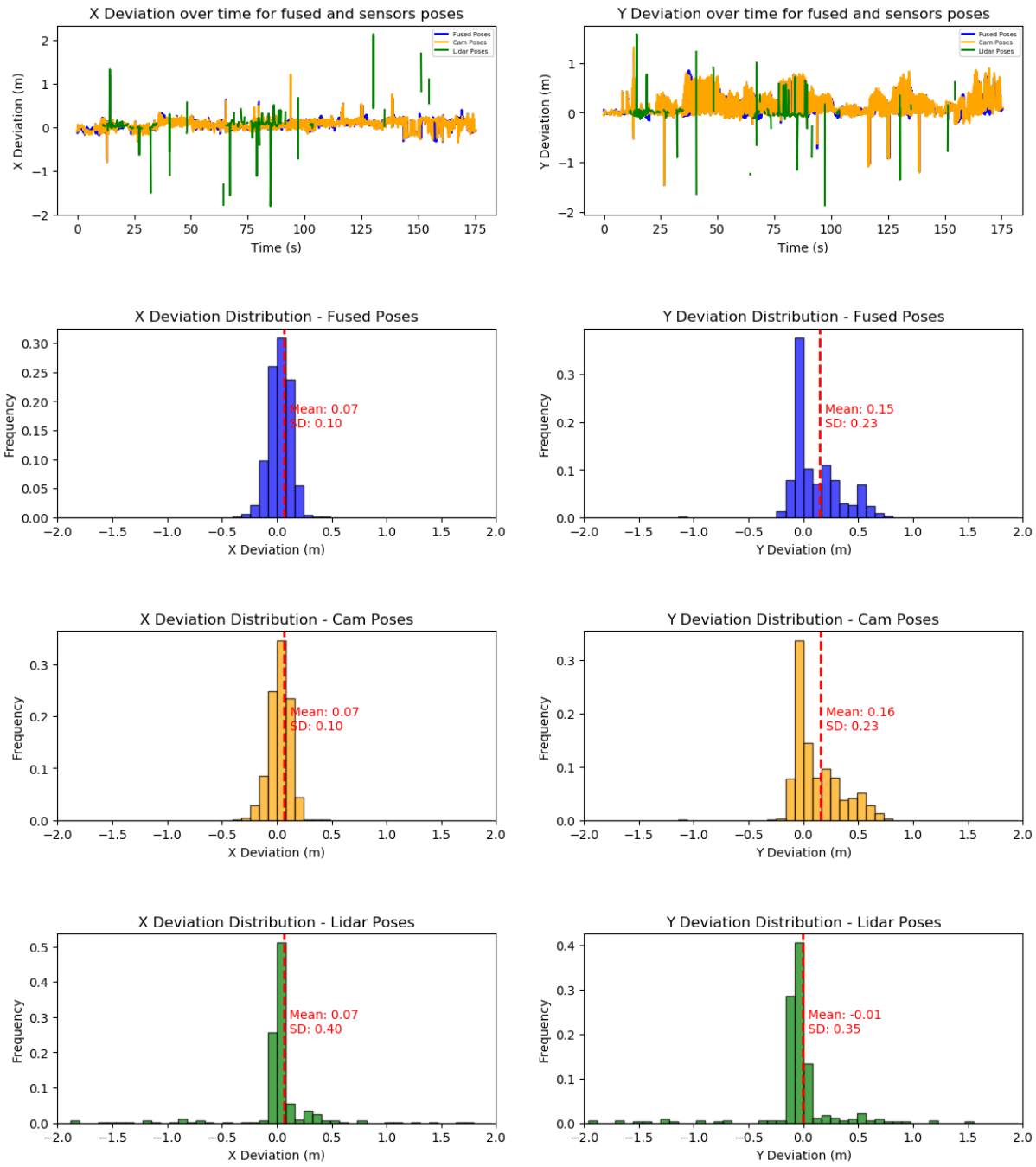


Figure C.8: Second experiment, High-level fusion with standalone LiDAR detection: X and Y deviations metrics.

C.2.2 High-level fusion with partially assisted LiDAR detection

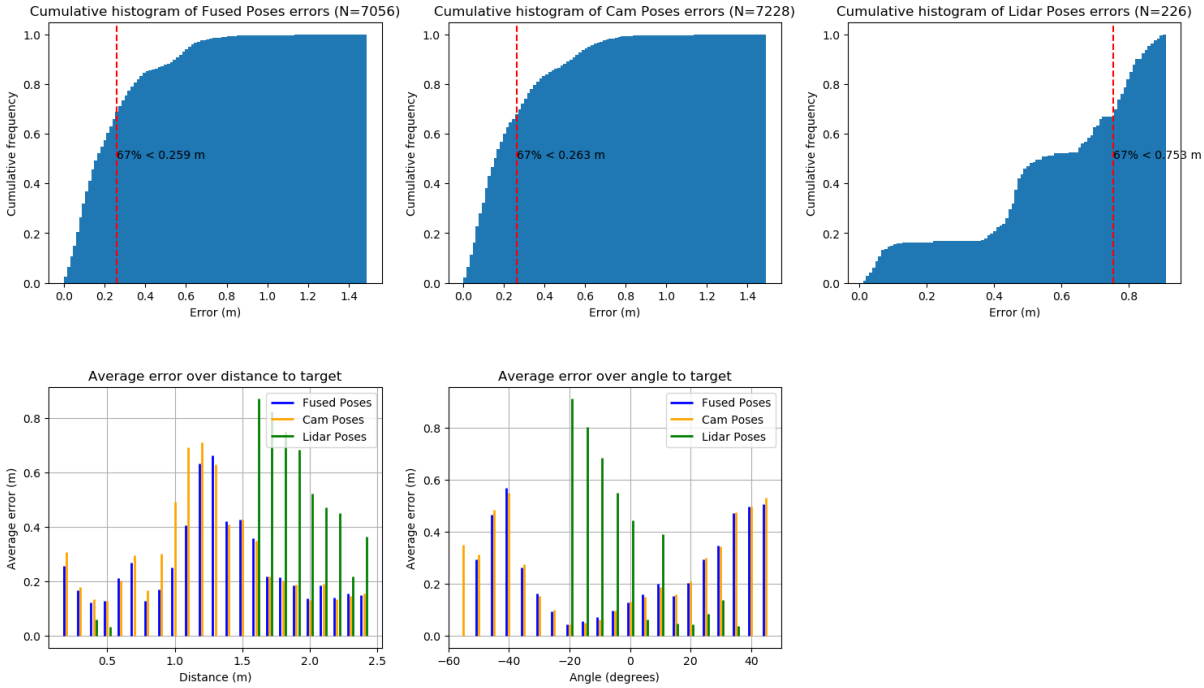


Figure C.9: Second experiment, High-level fusion with partially assisted LiDAR detection: Errors Evaluation metrics.

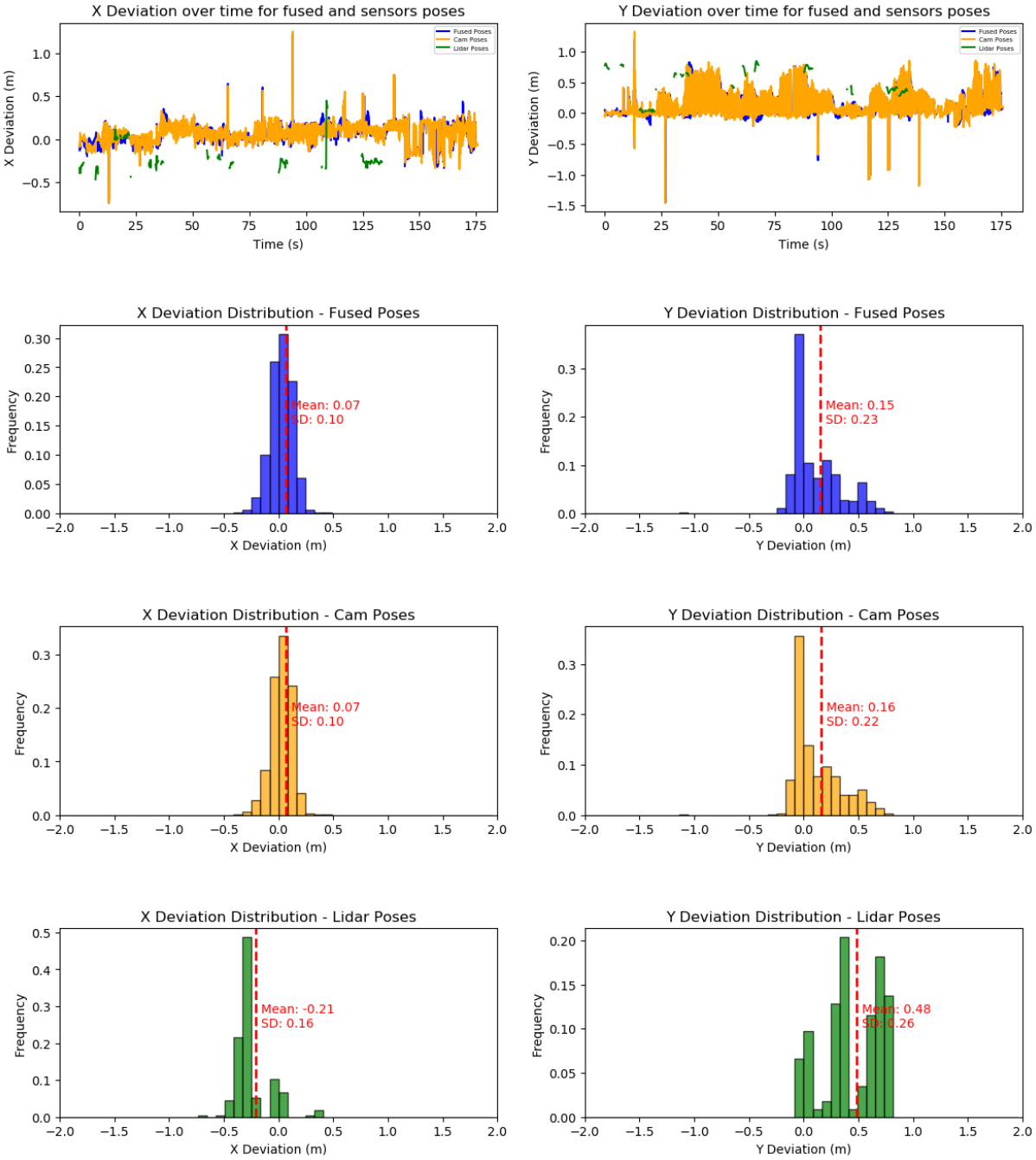


Figure C.10: Second experiment, High-level fusion with partially assisted LiDAR detection: X and Y deviations metrics.

C.2.3 High-level fusion with fully assisted LiDAR detection

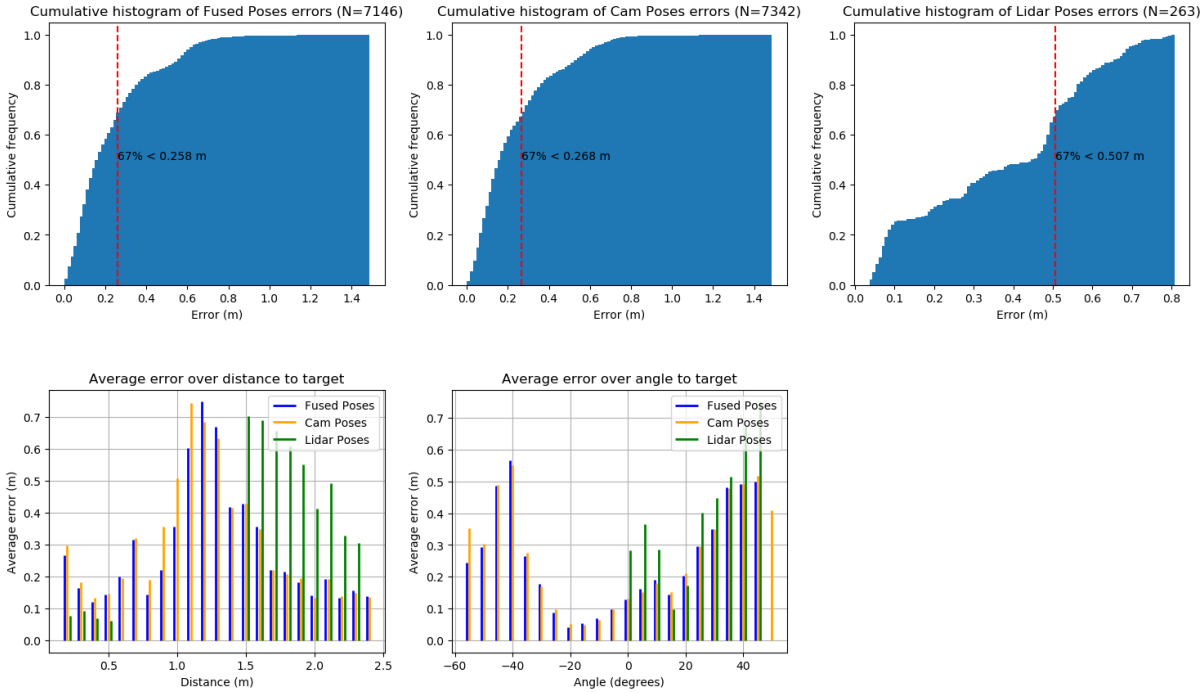


Figure C.11: Second experiment, High-level fusion with fully assisted LiDAR detection: Errors Evaluation metrics.

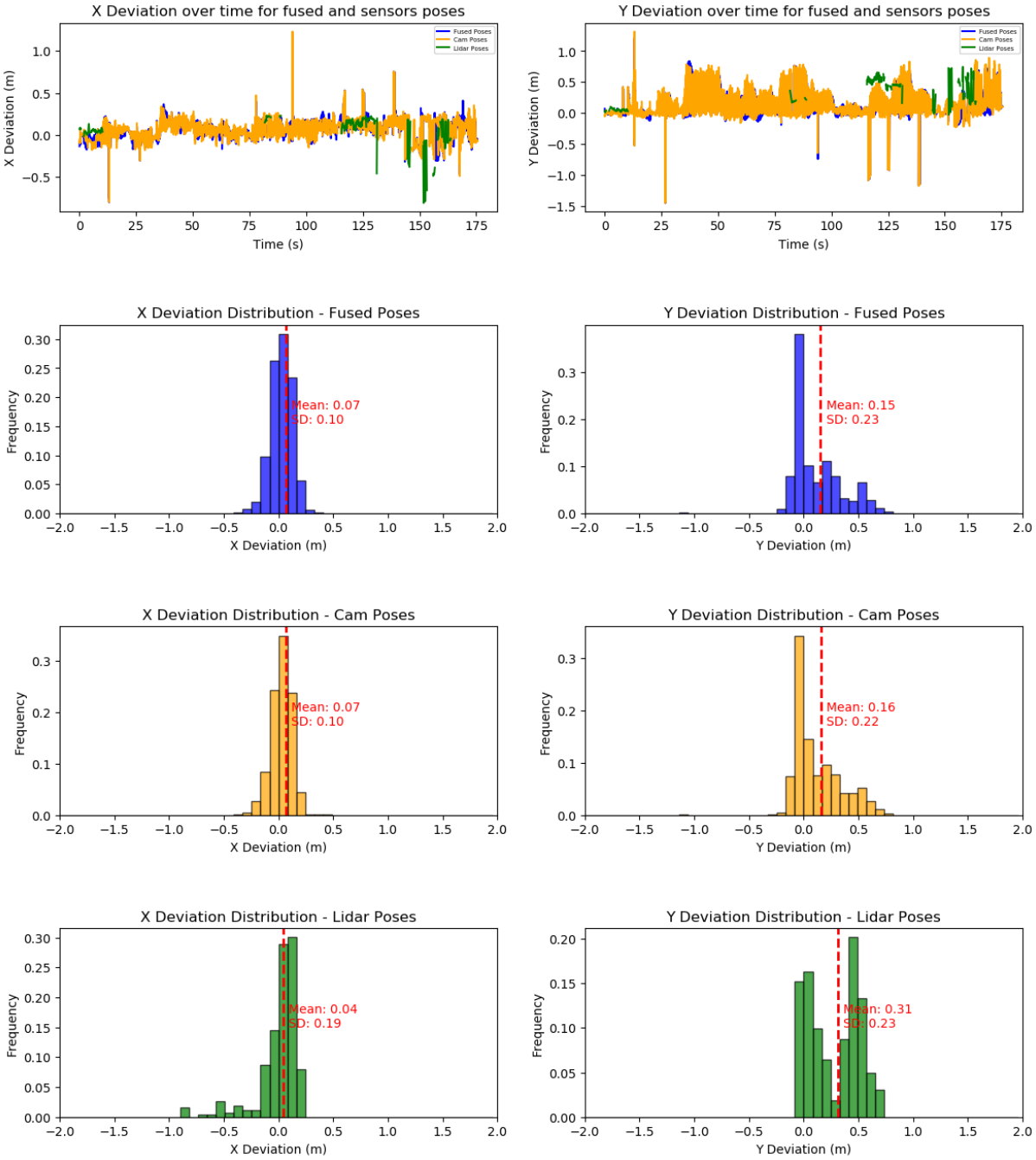


Figure C.12: Second experiment, High-level fusion with fully assisted LiDAR detection: X and Y deviations metrics.

C.3 Third experiment

C.3.1 High-level fusion with standalone LiDAR detection

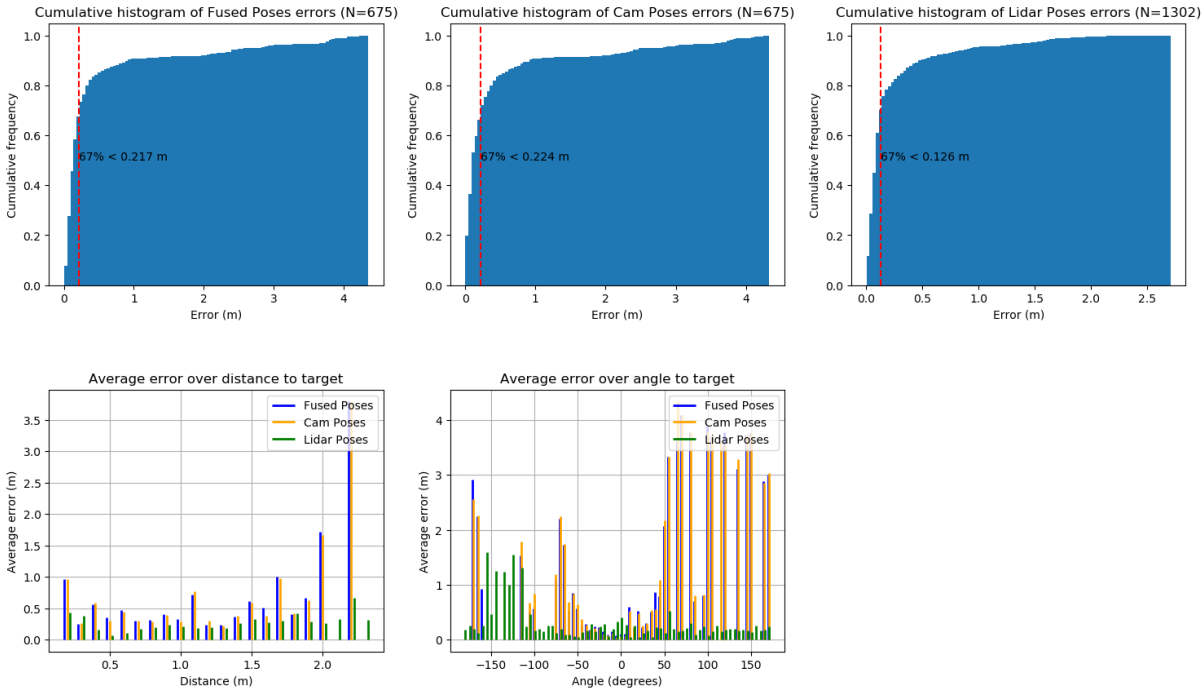


Figure C.13: Third experiment, High-level fusion with standalone LiDAR detection: Errors Evaluation metrics.

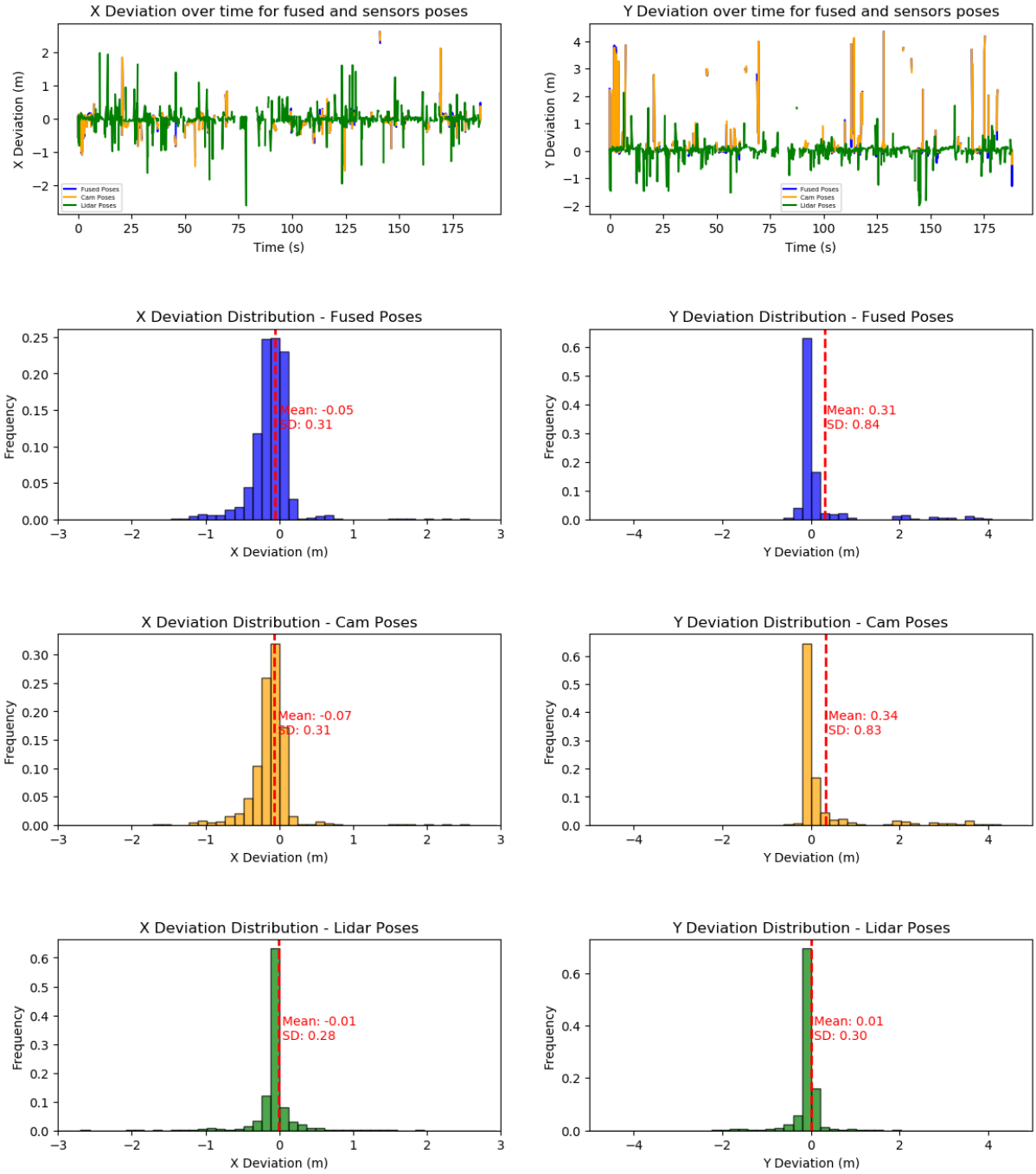


Figure C.14: Third experiment, High-level fusion with standalone LiDAR detection: X and Y deviations metrics.

C.3.2 High-level fusion with partially assisted LiDAR detection

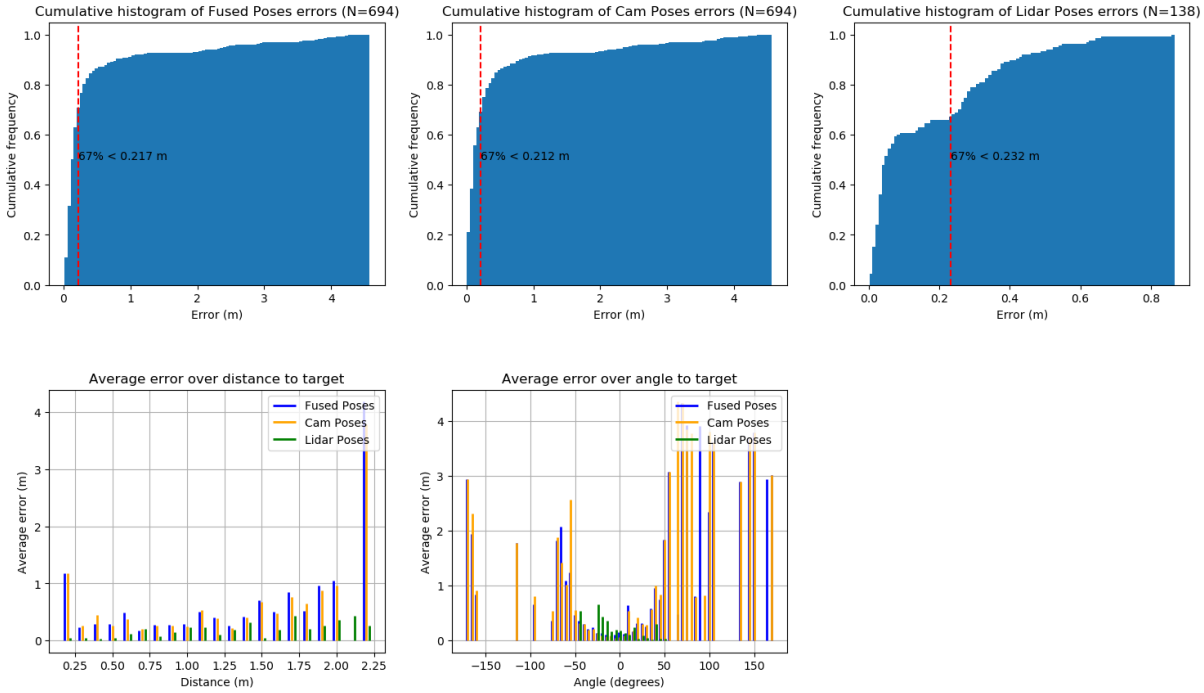


Figure C.15: Third experiment, High-level fusion with partially assisted LiDAR detection: Errors Evaluation metrics.

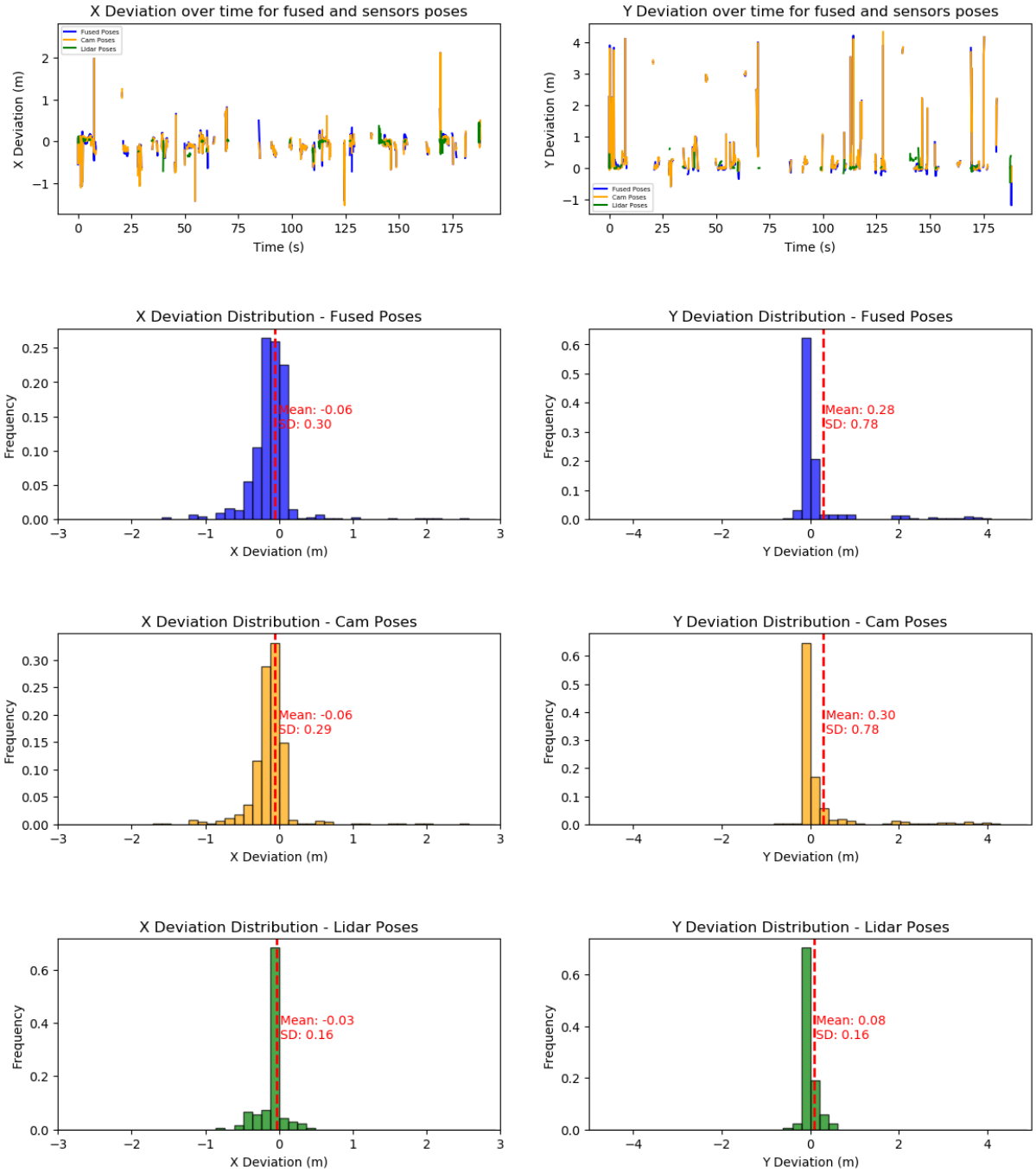


Figure C.16: Third experiment, High-level fusion with partially assisted LiDAR detection: X and Y deviations metrics.

C.3.3 High-level fusion with fully assisted LiDAR detection

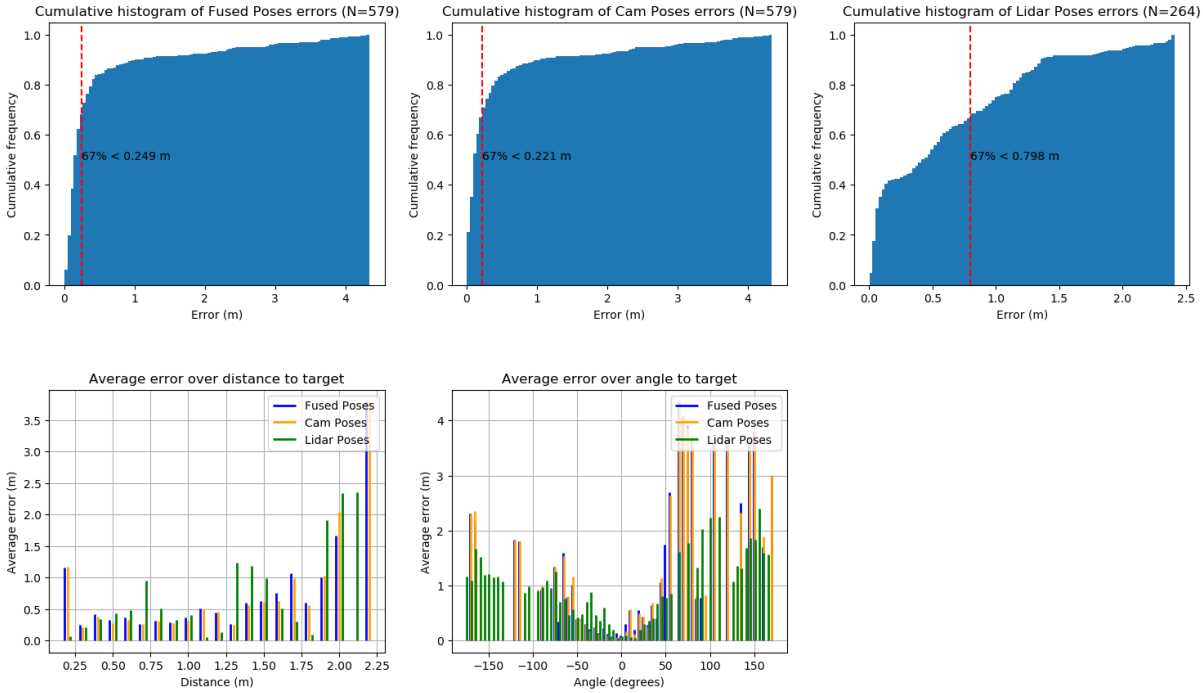


Figure C.17: Third experiment, High-level fusion with fully assisted LiDAR detection: Errors Evaluation metrics.

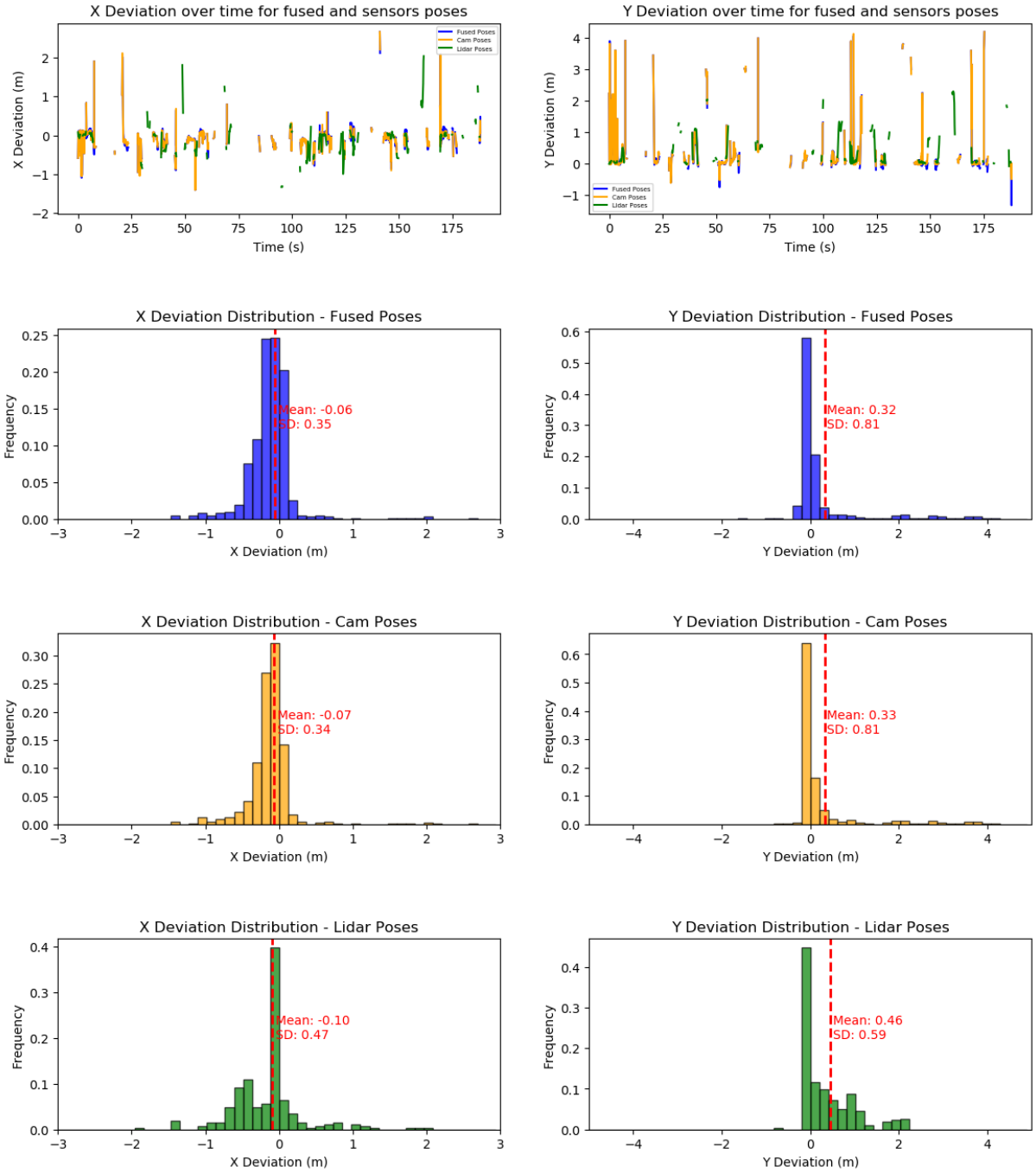


Figure C.18: Third experiment, High-level fusion with fully assisted LiDAR detection: X and Y deviations metrics.

Bibliography

[1] M. Kegeleirs, G. Grisetti, and M. Birattari. "Swarm SLAM: challenges and perspectives". In: *Frontiers in Robotics and AI* 8 (2021), p. 23. DOI: 10.3389/frobt.2021.618268. URL: https://dx.doi.org/10.3389/frobt.2021.618268.

[2] D.L. Hall and J. Llinas. "An Introduction to Multisensor Data Fusion". In: *Proceedings of the IEEE* 85.1 (Jan. 1997), pp. 6–23. ISSN: 1558-2256. DOI: 10.1109/5.554205. URL: https://ieeexplore.ieee.org/abstract/document/554205 (visited on 04/29/2024).

[3] Prasanna Kolar, Patrick Benavidez, and Mo Jamshidi. "Survey of Datafusion Techniques for Laser and Vision Based Sensor Integration for Autonomous Navigation". In: 20.8 (2020), p. 2180. DOI: 10.3390/s20082180. URL: https://www.proquest.com/docview/2390442172/abstract/AFCDBA3C6C8B44DEPQ/1 (visited on 04/29/2024).

[4] R. Omar Chavez-Garcia. "Multiple Sensor Fusion for Detection, Classification and Tracking of Moving Objects in Driving Environments". PhD thesis. Université de Grenoble, Sept. 2014. URL: https://hal.science/tel-01082021 (visited on 04/29/2024).

[5] Tresna Dewi, Naoki Uchiyama, Shigenori Sano, and Hiroki Takahashi. "Swarm Robot Control for Human Services and Moving Rehabilitation by Sensor Fusion". In: 2014 (2014). ISSN: 16879600. DOI: 10.1155/2014/278659. URL: https://www.proquest.com/docview/1503488410/abstract/827EA259FA9A41B5PQ/1 (visited on 04/29/2024).

[6] Andres J. Barreto-Cubero, Alfonso Gómez-Espinosa, Jesús Arturo Escobedo Cabello, Enrique Cuan-Urquizo, and Sergio R. Cruz-Ramírez. "Sensor Data Fusion for a Mobile Robot Using Neural Networks". In: *Sensors (Basel, Switzerland)* 22.1 (2021), pp. 305–. ISSN: 1424-8220. DOI: 10.3390/s22010305.

[7] Manuele Brambilla, Eliseo Ferrante, Mauro Birattari, and Marco Dorigo. "Swarm Robotics: A Review from the Swarm Engineering Perspective". In: *Swarm Intelligence* 7.1 (Mar. 2013), pp. 1–41. ISSN: 1935-3820. DOI: 10.1007/s11721-012-0075-2. URL: https://doi.org/10.1007/s11721-012-0075-2 (visited on 04/29/2024).

[8] Mauro Birattari, Antoine Ligot, Darko Bozhinoski, Manuele Brambilla, Gianpiero Francesca, Lorenzo Garattoni, David Garzón Ramos, Ken Hasselmann, Miquel Kegeleirs, Jonas Kuckling, Federico Pagnozzi, Andrea Roli, Muhammad Salman, and Thomas Stützle. "Automatic Off-Line Design of Robot Swarms: A Manifesto". In: *Frontiers in Robotics and AI* 6 (July 2019). ISSN: 2296-9144. DOI: 10.3389/frobt.2019.00059. URL: https://www.frontiersin.org/journals/robotics-and-ai/articles/10.3389/frobt.2019.00059/full (visited on 07/07/2024).

[9] Muhammad Salman, David Garzón Ramos, and Mauro Birattari. "Automatic Design of Stigmergy-Based Behaviours for Robot Swarms". In: *Communications Engineering* 3.1 (Feb. 2024), pp. 1–13. ISSN: 2731-3395. DOI: 10.1038/s44172-024-00175-7. URL: https://www.nature.com/articles/s44172-024-00175-7 (visited on 07/07/2024).

[10] Ken Hasselmann. "Advances in the automatic modular design of control software for robot swarms: Using neuroevolution to generate modules". PhD thesis. PhD thesis. Brussels, Belgium: Université libre de Bruxelles, 2023.

[11] Jonas Kuckling. "Optimization in the automatic modular design of control software for robot swarms". In: (2023).

[12] Antoine Ligot. "Assessing and forecasting the performance of optimization-based design methods for robot swarms: Experimental protocol and pseudo-reality predictors". In: (2023).

[13] M. Kegeleirs, D. Garzón Ramos, K. Hasselmann, L. Garattoni, G. Francesca, and M. Birattari. "Transferability in the automatic off-line design of robot swarms: from sim-to-real to embodiment and design-method transfer across different platforms". In: *IEEE Robotics and Automation Letters* 9.3 (2024), 2758–2765. DOI: 10.1109/LRA.2024.3360013. URL: https://dx.doi.org/10.1109/LRA.2024.3360013.

[14] Mauro Birattari, Antoine Ligot, and Gianpiero Francesca. "AutoMoDe: A Modular Approach to the Automatic Off-Line Design and Fine-Tuning of Control Software for Robot Swarms". In: *Automated Design of Machine Learning and Search Algorithms*. Ed. by Nelishia Pillay and Rong Qu. Cham: Springer International Publishing, 2021, pp. 73–90. ISBN: 978-3-030-72069-8. DOI: 10.1007/978-3-030-72069-8_5. URL: https://doi.org/10.1007/978-3-030-72069-8_5 (visited on 07/07/2024).

[15] D. Garzón Ramos, D. Bozhinoski, G. Francesca, L. Garattoni, K. Hasselmann, M. Kegeleirs, J. Kuckling, A. Ligot, J. Mendiburu Fernando, F. Pagnozzi, M. Salman, T. Stutzle, and M. Birattari. "The automatic off-line design of robot swarms:recent advances and perspectives". In: *R2T2: Robotics Research for Tomorrow's Technology*. Ed. by Giulia De Masi, Eliseo Ferrante, and Paolo Dario. Abu Dhabi, United Arab Emirates: Technology Innovation Institute, 2021. URL: https://dx.doi.org/.

[16] G. Spaey, M. Kegeleirs, D. Garzón Ramos, and M. Birattari. "Evaluation of alternative exploration schemes in the automatic modular design of robot swarms". In: *Artificial Intelligence and Machine Learning: BNAIC 2019, BENELEARN 2019*. Ed. by Bart Bogaerts, Gianluca Bontempi, Pierre Geurts, Nick Harley, Bertrand Lebichot, Tom Lenaerts, and Gilles Louppe. Vol. 1196. CCIS. Cham, Switzerland: Springer, 2020, 18–33. DOI: 10.1007/978-3-030-65154-1_2. URL: https://dx.doi.org/10.1007/978-3-030-65154-1_2.

[17] Numan Senel, Klaus Kefferpütz, Kristina Doycheva, and Gordon Elger. "Multi-Sensor Data Fusion for Real-Time Multi-Object Tracking". In: *Processes* 11.2 (2023), pp. 501–. ISSN: 2227-9717. DOI: 10.3390/pr11020501.

[18] Luigi D'Alfonso, Walter Lucia, Pietro Muraca, and Paolo Pugliese. "Mobile Robot Localization via EKF and UKF: A Comparison Based on Real Data". In: *Robotics and Autonomous Systems* 74 (Dec. 2015), pp. 122–127. ISSN: 0921-8890. DOI: 10.1016/j.robot.2015.07.007. URL: https://www.sciencedirect.com/science/article/pii/S0921889015001517 (visited on 04/29/2024).

[19] Chao Zhang, Quanzhong Zhan, Qi Wang, Haichao Wu, Ting He, and Yi An. "Autonomous Dam Surveillance Robot System Based on Multi-Sensor Fusion". In: *Sensors (Basel, Switzerland)* 20.4 (2020), pp. 1097–. ISSN: 1424-8220. DOI: 10.3390/s20041097.

[20] Vladimír Kubelka, Michal Reinstein, and Tomáš Svoboda. "Improving Multimodal Data Fusion for Mobile Robots by Trajectory Smoothing". In: *Robotics and Autonomous Systems* 84 (Oct. 2016), pp. 88–96. ISSN: 0921-8890. DOI: 10.1016/j.robot.2016.07.006. URL: https://www.sciencedirect.com/science/article/pii/S0921889015300191 (visited on 04/29/2024).

[21] Yassen Dobrev, Peter Gulden, and Martin Vossiek. "An Indoor Positioning System Based on Wireless Range and Angle Measurements Assisted by Multi-Modal Sensor Fusion for Service Robot Applications". In: *IEEE Access* 6 (2018), pp. 69036–69052. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2879029. URL: https://ieeexplore.ieee.org/document/8519725 (visited on 04/29/2024).

[22] Nader Abdelaziz and Ahmed El-Rabbany. "INS/LIDAR/Stereo SLAM Integration for Precision Navigation in GNSS-Denied Environments". In: *Sensors* 23.17 (Jan. 2023), p. 7424. ISSN: 1424-8220. DOI: 10.3390/s23177424. URL: https://www.mdpi.com/1424-8220/23/17/7424 (visited on 04/29/2024).

[23] G. Ajay Kumar, Jin Hee Lee, Jongrak Hwang, Jaehyeong Park, Sung Hoon Youn, and Soon Kwon. "LiDAR and Camera Fusion Approach for Object Distance Estimation in Self-Driving Vehicles". In: *Symmetry* 12.2 (Feb. 2020), p. 324. ISSN: 2073-8994. DOI: 10.3390/sym12020324. URL: https://www.mdpi.com/2073-8994/12/2/324 (visited on 04/29/2024).

[24] Pranay Mathur, Ravish Kumar, and Rahul Jain. "Multi-Sensor Fusion-Based Object Detection Implemented on ROS". In: *Machine Learning and Autonomous Systems*. Ed. by Joy Iong-Zong Chen, Haoxiang Wang, Ke-Lin Du, and V. Suma. Singapore: Springer Nature, 2022, pp. 551–563. ISBN: 9789811679964. DOI: 10.1007/978-981-16-7996-4_40.

[25] Haibin Liu, Chao Wu, and Huanjie Wang. "Real Time Object Detection Using LiDAR and Camera Fusion for Autonomous Driving". In: 13.1 (2023), p. 8056. DOI: 10.1038/s41598-023-35170-z. URL: https://www.proquest.com/docview/2814645971/abstract/2D8A85799DC74D28PQ/1 (visited on 04/29/2024).

[26] Martin Dimitrievski, Peter Veelaert, and Wilfried Philips. "Behavioral Pedestrian Tracking Using a Camera and LiDAR Sensors on a Moving Vehicle". In: *Sensors* 19.2 (Jan. 2019), p. 391. ISSN: 1424-8220. DOI: 10.3390/s19020391. URL: https://www.mdpi.com/1424-8220/19/2/391 (visited on 04/29/2024).

[27] Shenglin Li and Hwan-Sik Yoon. "Sensor Fusion-Based Vehicle Detection and Tracking Using a Single Camera and Radar at a Traffic Intersection". In: *Sensors* 23.10 (May 2023). ISSN: 1424-8220. DOI: 10.3390/s23104888. URL: https://www.osti.gov/pages/biblio/1974483 (visited on 04/29/2024).

[28] Oscar Javier Montañez, Marco Javier Suarez, and Eduardo Avendano Fernandez. "Application of Data Sensor Fusion Using Extended Kalman Filter Algorithm for Identification and Tracking of Moving Targets from LiDAR–Radar Data". In: *Remote Sensing* 15.13 (Jan. 2023), p. 3396. ISSN: 2072-4292. DOI: 10.3390/rs15133396. URL: https://www.mdpi.com/2072-4292/15/13/3396 (visited on 04/29/2024).

[29] Yaqin Wang, Dongfang Liu, and Eric Matson. "Accurate Perception for Autonomous Driving: Application of Kalman Filter for Sensor Fusion". In: *2020 IEEE Sensors Applications Symposium (SAS)*. Mar. 2020, pp. 1–6. DOI: 10.1109/SAS48726.2020.9220083. URL: https://ieeexplore.ieee.org/abstract/document/9220083 (visited on 04/29/2024).

[30] Taeklim Kim and Tae-Hyoung Park. "Extended Kalman Filter (EKF) Design for Vehicle Position Tracking Using Reliability Function of Radar and Lidar". In: 20.15 (2020), p. 4126. DOI: 10.3390/s20154126. URL: https://www.proquest.com/docview/2427966772/abstract/BCB10F0374BE4330PQ/1 (visited on 04/29/2024).

[31] Qi Kong, Liangliang Zhang, and Xin Xu. "Outdoor Real-Time RGBD Sensor Fusion of Stereo Camera and Sparse Lidar". In: 2234.1 (2022), p. 012010. ISSN: 17426588. DOI: 10.1088/1742-6596/2234/1/012010. URL: https://www.proquest.com/docview/2649750021/abstract/90979C6DB9D04A1APQ/1 (visited on 04/29/2024).

[32] Will Maddern and Paul Newman. "Real-Time Probabilistic Fusion of Sparse 3D LIDAR and Dense Stereo". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2016, pp. 2181–2188. DOI: 10.1109/IROS.2016.7759342. URL: https://ieeexplore.ieee.org/abstract/document/7759342 (visited on 04/29/2024).

[33] Kihong Park, Seungryong Kim, and Kwanghoon Sohn. "High-Precision Depth Estimation with the 3D LiDAR and Stereo Fusion". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. May 2018, pp. 2156–2163. DOI: 10.1109/ICRA.2018.8461048. URL: https://ieeexplore.ieee.org/abstract/document/8461048 (visited on 04/29/2024).

[34] A Geiger, P Lenz, C Stiller, and R Urtasun. "Vision Meets Robotics: The KITTI Dataset". In: *The International Journal of Robotics Research* 32.11 (Sept. 2013), pp. 1231–1237. ISSN: 0278-3649. DOI: 10.1177/0278364913491297. URL: https://doi.org/10.1177/0278364913491297 (visited on 04/29/2024).

[35] M. Kegeleirs, R. Todesco, D. Garzón Ramos, G. Legarda Herranz, and M. Birattari. *Mercator: hardware and software architecture for experiments in swarm SLAM*. Tech. rep. TR/IRIDIA/2022-012. Brussels, Belgium: IRIDIA, Université libre de Bruxelles, 2022.

[36] *OAK-D W — DepthAI Hardware Documentation 1.0.0 Documentation*. URL: https://docs.luxonis.com/projects/hardware/en/latest/pages/DM9098w/ (visited on 04/30/2024).

[37] *SpatialLocationCalculator — DepthAI Documentation | Luxonis*. URL: https://docs.luxonis.com/projects/api/en/latest/components/nodes/spatial_location_calculator/#spatiallocationcalculator (visited on 04/30/2024).

[38] Shuncong Shen, Mai Saito, Yuka Uzawa, and Toshio Ito. "Optimal Clustering of Point Cloud by 2D-LiDAR Using Kalman Filter". In: *Journal of Robotics and Mechatronics* 35.2 (Apr. 2023), pp. 424–434. DOI: 10.20965/jrm.2023.p0424. URL: https://www.fujipress.jp/jrm/rb/robot003500020424/ (visited on 04/30/2024).

[39] *Sensing Kit | 8 Sensor Modules | Custom Configurations*. URL: https://www.terabee.com/shop/lidar-tof-range-finders/teraranger-multiflex/ (visited on 04/30/2024).

[40] Jeffrey Humpherys, Preston Redd, and Jeremy West. "A Fresh Look at the Kalman Filter". In: *SIAM Review* 54.4 (2012), pp. 801–823. ISSN: 0036-1445. JSTOR: 24248361. URL: https://www.jstor.org/stable/24248361 (visited on 04/30/2024).

[41] R. E. Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *Journal of Basic Engineering* 82.1 (Mar. 1960), pp. 35–45. ISSN: 0021-9223. DOI: 10.1115/1.3662552. URL: https://doi.org/10.1115/1.3662552 (visited on 04/12/2024).

[42] Greg Welch and Gary Bishop. "An Introduction to the Kalman Filter". In: (1995). URL: https://perso.crans.org/club-krobot/doc/kalman.pdf (visited on 04/30/2024).

[43] Roger Labbe. "Kalman and Bayesian Filters in Python". In: *Chap* 7.246 (2014), p. 4. URL: https://www.researchgate.net/profile/Gopinath-Balu-2/project/Multiple-Object-Tracking-2/attachment/5c1220563843b006754b0f7a/AS:703302526320644@1544691797777/download/Kalman_and_Bayesian_Filters_in_Python.pdf?context=ProjectUpdatesLog (visited on 04/22/2024).

[44] *OAK-D — DepthAI Hardware Documentation 1.0.0 Documentation*. URL: https://docs.luxonis.com/projects/hardware/en/latest/pages/BW1098OAK/#stereo-depth-perception (visited on 04/24/2024).

[45] *Downloads & Support_YDLIDAR|Focus on Lidar Sensor Solutions*. URL: https://www.ydlidar.com/service_support/download.html?gid=5 (visited on 04/24/2024).

[46] Shahrokh Akhlaghi, Ning Zhou, and Zhenyu Huang. "Adaptive Adjustment of Noise Covariance in Kalman Filter for Dynamic State Estimation". In: *2017 IEEE Power & Energy Society General Meeting*. July 2017, pp. 1–5. DOI: 10.1109/PESGM.2017.8273755. URL: https://ieeexplore.ieee.org/abstract/document/8273755 (visited on 04/30/2024).

[47] H. W. Kuhn. "The Hungarian Method for the Assignment Problem". In: *Naval research logistics* 52.1 (2005), pp. 7–21. ISSN: 0894-069X. DOI: 10.1002/nav.20053.

[48] *Scipy.Optimize.Linear_sum_assignment — SciPy v1.13.0 Manual*. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.linear_sum_assignment.html#scipy.optimize.linear_sum_assignment (visited on 04/30/2024).

[49] David F. Crouse. "On Implementing 2D Rectangular Assignment Algorithms". In: *IEEE Transactions on Aerospace and Electronic Systems* 52.4 (Aug. 2016), pp. 1679–1696. ISSN: 1557-9603. DOI: 10.1109/TAES.2016.140952. URL: https://ieeexplore.ieee.org/abstract/document/7738348 (visited on 04/30/2024).

[50] Martin Engelcke, Dushyant Rao, Dominic Zeng Wang, Chi Hay Tong, and Ingmar Posner. "Vote3Deep: Fast Object Detection in 3D Point Clouds Using Efficient Convolutional Neural Networks". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. May 2017, pp. 1355–1361. DOI: 10.1109/ICRA.2017.7989161. URL: https://ieeexplore.ieee.org/abstract/document/7989161 (visited on 05/01/2024).

[51] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. "Multi-View 3D Object Detection Network for Autonomous Driving". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 1907–1915. URL: https://openaccess.thecvf.com/content_cvpr_2017/html/Chen_Multi-View_3D_Object_CVPR_2017_paper.html (visited on 05/01/2024).

[52] Yin Zhou and Oncel Tuzel. "VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4490–4499. URL: https://openaccess.thecvf.com/content_cvpr_2018/html/Zhou_VoxelNet_End-to-End_Learning_CVPR_2018_paper.html (visited on 05/01/2024).

[53] K. S. Arun, T. S. Huang, and S. D. Blostein. "Least-Squares Fitting of Two 3-D Point Sets". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-9.5 (Sept. 1987), pp. 698–700. ISSN: 1939-3539. DOI: 10.1109/TPAMI.1987.4767965. URL: https://ieeexplore.ieee.org/abstract/document/4767965 (visited on 07/15/2024).

[54] Vitor Santos, Daniela Rato, Paulo Dias, and Miguel Oliveira. "Multi-sensor extrinsic calibration using an extended set of pairwise geometric transformations". In: *Sensors* 20.23 (2020), p. 6717.

[55] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and R. Medina-Carnicer. "Generation of Fiducial Marker Dictionaries Using Mixed Integer Linear Programming". In: *Pattern Recognition* 51 (Mar. 2016), pp. 481–491. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2015.09.023. URL: https://www.sciencedirect.com/science/article/pii/S0031320315003544 (visited on 07/15/2024).

[56] G. Legarda Herranz, D. Garzón Ramos, J. Kuckling, M. Kegeleirs, and M. Birattari. *Tycho: a robust, ROS-based tracking system for robot swarms*. Tech. rep. TR/IRIDIA/2022-009. Brussels, Belgium: IRIDIA, Université libre de Bruxelles, 2022.

[57] M. Kegeleirs, D. Garzón Ramos, and M. Birattari. "Random walk exploration for swarm mapping". In: *Towards Autonomous Robotic Systems, TAROS*. Ed. by Kaspar Althoefer, Jelizaveta Konstantinova, and Ketao Zhang. Vol. 11650. LNCS. Cham, Switzerland: Springer, 2019, 211–222. DOI: 10.1007/978-3-030-25332-5_19. URL: https://dx.doi.org/10.1007/978-3-030-25332-5_19.