CSE200 Software Project

Tracking Objects Across Frames

Final Report

DELFT, JUNE 19TH 2022

Group 5B:
Aleksandra Andrasz,
Filip Biliński,
Natalia Bryła,
Paul Misterka,
Natalia Pietnoczko

TA:
Aleksandra Wolska

TU Coach:
AP Qing Wang

Client:
S4FE

S4FE
check | walk | track | report

TU Delft

Preface

The authors of this report are five second-year Computer Science and Engineering students at Delft University of Technology. For the purposes of the Software Project course, we were working on a project for S4FE, an Italian startup. The task was to develop a system for detecting and tracking people in monitored spaces.

The reader of this report should have basic technical academic knowledge. The topics mainly include software engineering. For clarity, when a more advanced term is used, we add a footnote with an explanation.

Readers who are only interested in the implementation can focus on Chapter 5. Those particularly interested in the achieved results can jump to Chapter 6. Alternatively, if one wants to learn more about the ethical concerns that come with our project, they should look into Section 4.1.

Our thanks go to our TU Coach, AP Qing Wang, for supervising and supporting our group in this project. As well as our TA, Aleksandra Wolska, for her advice and feedback on assignments and teamwork. Most importantly, we thank S4FE for this amazing opportunity to work in Barcelona on such an innovative and ambitious project and for trusting us with their hardware. Special thanks go to Pietro Rustici for his mentorship and involvement in the project.

Delft, 19 June 2022
Aleksandra Andrasz
Filip Biliński
Natalia Bryła
Paul Misterka
Natalia Pietnoczko

Summary

Comfort nowadays is highly valued by society. Focusing on feeling well in public spaces, it is possible to arrange, for example, restaurants, museums or parks in many different ways. With the help of technology, we can design certain areas to have a minimum number of overcrowded spots. The result is increased well-being and a feeling of safety for the visitors. To achieve this goal, we implemented a multiple object tracking system that detects every person in a camera view and tracks them. Within this software, it is easy to analyze the movement of crowds and quickly identify the most critical spots with the highest object density.

This report aims to present the project design, architecture and implementation of the introduced software. All those parts were created and reviewed based on the knowledge of team members, guidelines from the client (company S4FE) and the current state-of-the-art in object detection and object tracking algorithms.

Project design describes the first ideas for the future implementation and analyses the limitations that were identified since the beginning of the project. Tracking people in public spaces involves gathering video information about certain individuals. That is why we firstly reasoned about ethical concerns and ways of achieving this goal without violating the General Data Protection Regulation. The idea is to perform all the computations on the cameras and later analyze only the raw data with anonymous, randomly generated UUIDs, timestamps and locations. After researching and implementing object detection and multiple object tracking (MOT) algorithms, there was also a plan to add additional functionalities to the system. The design of the bird-view transformation feature was made to be able to analyze the data from a more user-friendly perspective. Additionally, we came up with an idea to test the performance of different algorithms by the benchmarking system. With it, it is possible to decide which algorithm performs better.

We wanted our system to be easily scalable and customizable, so designing and implementing the architecture was crucial. The idea is to make the pipeline architecture with a chain of responsibilities design pattern. This approach allows developers to include and exclude certain parts of the pipeline. Moreover, it also allows attaching a newly created part of the pipeline to the existing chain. Therefore, by implementing the architecture in a described way, we leave room for future improvements.

The implementation of the system was a major part of the project. That is why we thoroughly describe the approaches taken while implementing this system. Starting from camera setup and preprocessing of images which helps in obtaining better results, we follow with the description of a less basic feature: filtering false positive detections in the background. This feature allows the system to distinguish the false detections that are not moving from the correct tracked objects. Nonetheless, the most important functionalities are object detection and multiple object tracking (MOT), so the main focus is on them. We present different algorithms for both object detection and MOT that we managed or did not manage to implement. Additionally, in order to compare these various algorithms, the benchmarking system was created and used for comparison of algorithm's results. What is more, the implementation of the bird-view transformation along with mapping was done and described. This feature allows the user to see the visualized density of objects in a certain space from the top view.

One of the most challenging part of the implementation was tracking across different camera views. This includes finding the area spanned by every camera and then performing tracking. This part is done by reconstructing one map for all the cameras and tracking objects on this map. Additionally, we describe the implementation of the control panel, which is the easiest way to configure and run the system.

Another necessary functionality of our software is communication, which we divided into two parts: database setup and intercamera communication. Setting up the intercamera communication required additional research and discussions with the client. After that, we decided to have a server that will gather data from all the cameras with the possibility to have the server locally and run the network without an internet connection.

The project was challenging from the beginning and required us to be well-organized and do a lot of deep research. In the end, we completed all of the must-have requirements and more than half of the should-haves. We have implemented a working prototype that detects and tracks people in a camera view. This system later provides users with useful information about density of crowd in every type of space where the deployment of cameras is possible.

Contents

1 Introduction

In the post-covid times monitoring the density of crowds, especially in enclosed environments became an important challenge for the safety and well-being of society. It was proven that density of people has an influence of morbidity of COVID-19 [1]. Excluding the importance of preventing virus spread monitoring the movement of crowds bring several other benefits to the management of crowded spaces. This includes: being able to plan event organization based on past data to avoid overcrowding spaces, being able to plan restaurant table layouts to account for the movement of people in the space etc. It is crucial that a system performing the task is reliable and the devices are not visible in the space. This is important for people in the spaces to feel comfortable while still being able to retrieve data and plan accordingly.

The objective of this report is to present the design process, architecture and implementation of the software developed by our team in the past 10 weeks capable of monitoring the movement of crowds in variable environments. The core of our development process was a rapid iteration of the software with benchmarking. Rapid iteration enabled us to try out multiple detection and tracking algorithms while benchmarking was used to verify the performance of various iterations. The software was developed to be able to run stably and be optimized for small devices with low computational resources as this was a requirement from our client. The cameras on which the system is supposed to run can be seen on Figure 1. Another limiting factor of this project was working with privacy-sensitive data. Privacy is fundamental human right [2] which is very relevant in this project. Keeping privacy in mind the software should be developed with the GDPR guidelines in mind.

Figure 1: *Prototypes of cameras provided by S4FE company.*

The report is presented in the following structure. Chapter 2 provides context for the problem at hand as well as some ideas for dealing with the problem. Next Chapter 3 is a walk-through of the requirements elicitation for our product as well as a final list of requirements. A high-level overview of the software architecture is presented in Chapter 4 and the description of actual implementation techniques is in Chapter 5. Chapter 6 summarizes the present software state and discusses possible improvements. Finally, Chapter 7 presents the conclusion of the report.

2 Problem Analysis

Having a good understanding of the complex challenges S4FE is trying to solve is essential. This chapter gives a comprehensive overview of the posed task in section 2.1, while section 2.2 provides descriptions of its real-world use cases. In section 2.3 we talk about competing solutions, describe differences between the problems they try to solve and analyze their approaches. Section 2.4 emphasizes the importance of creating a law-compliant solution.

2.1 Problem Statement

While tech giants like Google and Amazon have been mining the internet for decades, having access to real-world data has been largely out of their reach. The untapped potential of understanding everyday human behaviour is elusive and realizing it is the principal objective of this project.

The goal is to create a cost-efficient and privacy-protecting system that can reliably track crowds in changing environments. The system would provide a visual feed from a distributed network of embedded devices equipped with RGB cameras. We are designing software that can extract the positions of people on the input images and tag individuals throughout an entire session. These statistics should be anonymized to preserve privacy of tracked people. Later the information would be saved and analyzed to provide insightful data to the users.

The produced software ideally would be extendable, maintainable, and easy to use. The client would like to integrate our solution into their product. Therefore the codebase must conform to style guidelines and be adequately documented. Finally, we need to provide automated testing that verifies the cohesion and performance of the system.

2.2 Overview of Use Cases

S4FE provided us with multiple target use cases of the system. This section goes into the details on exhibition layout optimization in museums and shows how understanding congestion increases restaurants' service quality. It also demonstrates how tracking people's movement between specific locations can provide value to outdoors event organizers.

Museums
The system could be used for better planning of museums' exhibition layout. It should be possible to retrieve data about how much time, on average, a visitor spends in a particular place. Later this information could help reduce crowdedness on the most popular sites, improving the visitor experience.

Based on the client's past deployments, it is critical to handle situations where objects similar to people (e.g. posters with images of people, statues) are in the view. If the software recognizes these objects as persons, the data could be skewed, heavily degrading performance.

Restaurants
The system can help restaurant owners in attracting customers with swift, high-quality customer service. The owners want to minimize congestion, increase throughput and decrease waiting times. These clients of S4FE are interested in knowing where congestion happens the most frequently and why. Having that information, the restaurant can quickly improve the venue layout.

Outdoor Events
Having identified issues, the staff can swiftly modify the space, allowing more people to move without problems during big outdoor events. One of the biggest challenges the organizers face is handling how the crowd moves around - people move in different directions at different speeds. They also often walk in groups. The system should still be able to accurately detect popular routes and identify critical points where the crowd slows down.

2.3 Research on Existing Solutions

Although the project can be considered innovative, many solutions to similar problems already exist. This section will summarize what problems our competition solves and how they implement their solution.

CrowdVision
CrowdVision[1] offers systems of cameras that track people and their movements. The goal is to provide data that can later be used for improving the experience of the visitors. Among the clients of CrowdVision, we can find airports, retail shops, convention centers, etc., so the visitors include travellers and shoppers. The cameras monitor the area and process the image to count the number of people in the view, their speed, and their direction of movement [3].

The company focuses on gathering accurate data from dense environments, but they do not consider changing lighting conditions. One of the reasons for that can be that in places such as retail stores, it is always expected to get high-quality images [4] because the lighting conditions do not change often. In the use cases given by S4FE, in museums or restaurants, the lights can play a huge role in a marketing strategy or an artistic vision. For that reason, we have to consider using different models for detecting objects depending on the current lighting conditions in the room.

In our project, we also account for spaces where it is not possible to place the camera on the ceiling or on the pillars. CrowdVision cameras are supposed to mount them that way, where the cameras are looking down to guarantee the anonymity of the pedestrians [4]. CrowdVision also does not have to worry about privacy regulations, while we have to consider extensive privacy protection laws.

Numina
Numina is another solution for a problem similar to ours, as it uses data "to help urban planners and municipal governments design better streets and public places" [5]. However, Numina is a network of sensors that tracks the behaviours of road users.

Since Numina's sensors are placed outside, they must work in a dynamic environment, specifically in varying lighting conditions. The data they collect includes the timestamp, position, and type of the tracked object. Then it is used for deriving the objects' path visualizations. Next, those paths are transformed into a heat map [5].

Numina is meant to be used outside in open spaces. Given the use cases given by our client (museums, restaurants, sports events), our system has to work in both large open spaces and small narrow rooms such as hallways. That can introduce additional challenges in mapping the camera image to the bird view. However, We do not look into increasing the durability of our cameras because they are meant to be used in controlled conditions.

2.4 Privacy-Compliant Data Storage

An essential requirement for the system is GDPR[2] compliance, which imposes certain constraints on data stored and sent through the system. Therefore, throughout the project, we make sure that our decisions concerning data storage align with this requirement. For example, images are not stored or sent to different devices, preventing the use of algorithms that make use of feature descriptors [6].

Considering that the system needs to store large amounts of data whose structure can change, the design of the persistence layer has to be robust. We believe that a non-relational database, such as MongoDB[3], fits the use cases best. The database should enable quick data extraction for efficient analysis. Ideally, the architecture would enable seamless integration with cloud services and 3rd party tools.

[1] CrowdVision: pedestrian analytics and insights company `https://www.crowdvision.com/`
[2] General Data Protection Regulation: https://gdpr-info.eu/
[3] MongoDB, a cross-platform document-oriented NoSQL database: https://www.mongodb.com/

3 Requirements

This chapter discusses the product requirements for the final product. Understanding stakeholders' values is crucial to building a good product, and defining the list of requirements is the centerpiece of that process. Section 3.1 describes the methods used for formulating the requirements and lists what stakeholders were taken into account. Next, in section 3.2 a full list of requirements is presented.

3.1 Requirements Elicitation

As the project is rather research-focused, defining a complete list of requirements is difficult. This is further exacerbated by the fact that we can only meet our client online and have few opportunities to interview potential stakeholders.

To derive a complete, comprehensive list of requirements, we devoted much time to interviewing our clients, researching alternatives, and discussing possible approaches. Before starting the implementation, we met four times with the company, read 30 different research papers on object detection and MOT, and held three separate discussions where we brainstormed our list of requirements. The list was reviewed by the client multiple times, and in the end, was accepted by both client and the TU Delft coach.

The key to deriving a complete list of requirements is to identify the stakeholders and reason about their expectations of the system. While talking to the client we identified the following stakeholders for our project:

- **The client - S4FE**
  The core stakeholder, the main source of the requirements and the reviewer. Provides the specifications of the cameras and suggests possible solutions.
- **Future users of the system**
  The users will be interested in analyzing the gathered data. The system has to save information that is useful for the user and allows deriving the desired statistics.
- **People tracked by the system**
  The cameras will usually be placed in public spaces, so anyone can enter the monitored area. The system has to conform to privacy regulations so that it does not save any sensitive data about tracked people in the database.
- **TU Delft and the course staff**
  Before the product development can start, the TU Delft coach has to accept the requirements. They make sure that the project meets TU Delft standards as we represent the university.
- **Developers**
  Developers formulate the requirements based on information received from stakeholders and our personal experience. As a software engineering team, we have to elicit and formulate a feasible list of requirements.

3.2 Prioritized List of Requirements

We used MoSCoW method to derive comprehensive list of requirements presented in Appendix A The list will be reviewed and adapted throughout the project to meet our goals.

The minimum viable product is achieved by completing all of the **must-haves**. That system at that stage should be fully functional and respond to the basic stakeholders' expectations. However, it might still be a bit slow and limited.

With the **should-haves** performance problems should be less significant and some modules are extended with additional functionalities. For instance, must-haves contain a simple UI that just displays processed data whereas in the should-haves it is possible to display different views and more detailed information about the image. Object detection is upgraded by filtering the false-positive detections and tracking between different camera views is added.

The **could-have** describe requirements that pertain to automatization, security measures and useful features. By implementing them we make sure that the system is more maintainable in the long run and is essentially a full product ready for release.

We also came up with the **won't haves** requirements, these are the requirements that will not be implemented in the system. We decided that certain features cannot be attached because they are not ethically correct or there will not be enough time to work on them.

4 Product Design

This section gives an high-level overview of system architecture and used tools as well as the results of the research conducted for parts of the project. The design is divided into five logical parts. The first one is ethical concerns and their relevance in context of this project which is analyzed in section 4.1. Section 4.2 presents the approach for database setup and shows the chosen solution. Continuing in section 4.3 image processing pipeline, which is the software run by every embedded device, is described. Section 4.4 gives an overview of the benchmarking system, which enables fast, effortless, and comprehensive pipeline testing. The final section 4.5 focuses on describing the need of user interface and the way of displaying gathered data.

4.1 Ethical Concerns

This section analyzes ethical concerns that are important in the context of the project. As the project contains tracking people and analyzing their behaviour, it is necessary to ask appropriate ethical questions regarding working with human beings. Those issues are explored in detail in section 4.1.1. Where personal data concerns are described in section 4.1.2 which analyzes those issues and explains how we planned to mitigate them.

4.1.1 Human Participation

The system gathers video information about people and later analyzes it, hence the solution might directly influence individuals. For example, when deployed in a restaurant it could provide data about optimal table layout. Therefore we should recognize the responsibility in delivering a system that preserves important ethical values such as equity and privacy.

It is crucial, at every step of system development, to preserve a "consciously ethical mindset" [7]. When analyzing each part of the system, we found that there are parts that require extra attention. For example, object detection systems are shown to have varying error rates for different demographic groups [8]. For that reason equity has to be taken into account when designing, implementing and testing the system.

Another aspect would be analyzing the impact the system could have on future users. It is very useful to have inside into the optimal space organization but leaving all decisions to the machine might have a negative effect. Examining the use case of restaurants one can imagine someone who would like to rely solely on suggestions made by the algorithm. This could decrease the creativity of people planning the space and could result in the restaurant being designed similarly to other places which use the same system. In fact, optimization of the space only based on the movement of people would overrule other values of such space like its authenticity or artistic expression. This issue could be mitigated by deliberately choosing not to display certain suggestions to the customer. Allowing the user to prioritize the output rather than receiving a completed solution.

4.1.2 Personal Data

During requirements elicitation, we realized that data processed by the system may be considered sensitive or violate privacy laws. The system should benefit not only possible clients but also people in general. Therefore, it is important not to affect the privacy of intermediate users. We also do not see any justification for violating people's privacy by misuse of personal data in our system.

Regarding personal data, there are several aspects of the system that need examining. Firstly, there are concerns about storing the data ethically, which requires an examination of ethical issues connected to analyzing and distributing personal data. A significant value for those concerns is privacy. This not only means protecting sensitive data from malicious users but also being transparent about what data is stored [9].

Types of data used by the dystem
It was necessary to identify what data the software would use and analyze, how will it be done and what parts of that data are sensitive. The only privacy-sensitive data in the system is the input; a live feed from a camera, the images are not stored though. This is because the system has to be GDPR[4] compliant. Most of

[4]General Data Protection Regulation: https://gdpr-info.eu/

the ethical issues can be mitigated by following the GDPR guidelines as the regulation is assuring the privacy of the data subjects. With that, we decided on certain types of data would not be stored in our system, for example, as mentioned above live feed from the camera. Also storing features that could make identifying individual people possible, might come in conflict with GDPR.

Other data that the system handles is not privacy sensitive as it does not allow tracing back to an individual person. GDPR allows data subjects a "right to be forgotten", not storing images will make our software compliant with this rule. In conclusion, we should adhere to GDPR rules and especially keep in mind how the images and other personal data are handled in the system.

Security of stored data
Another aspect of ethical concerns regarding personal data is security. As stated by the European Commission report "Ethics and data protection" [2] data protection is a fundamental human right and is intimately linked to autonomy and human dignity. The same report, later on, explains how all systems should adhere to a principle of 'Data protection by design' and that whenever it is possible to enhance data privacy it should be done without discussion.

The system does store data about previously mapped objects from the system in a database. Therefore ethical aspects regarding security of data are relevant to our software. Furthermore, the stored information could be exploited by malicious users, for example, crowd movement data gathered at an event could give terrorists useful information about the best timing for an attack. Therefore it is crucial that we secure the data that we store inside the database. Firstly the database is not accessible on the network, instead, an API is used for communication to create a level of indirection. Secondly, all the API endpoints are secured with authorization. A malicious user will not be able to retrieve the data without the authorization keys.

Ethical benefits
Lastly, there are certain aspects that make the software beneficial in terms of ethical values. There are a lot of tracking systems which take little consideration of ethical concerns. Therefore many might think that the only way to analyze the movement of people in public spaces is by giving up the privacy of data subjects. We can show the opposite by creating a product that performs well and also adheres to important ethical values. The system could be used in place of a surveillance system that does not comply with GDPR. This would set a new standard for people tracking in public spaces.

4.2 Database Setup

One of the first steps of designing the software was to research different database options and choose the one that suits the needs of the system in the most efficient way. Data types are the first thing that should be taken into account while deciding on a database approach. The system stores large amount of data in a structure that can be modified. To make sure it is done effectively the system uses a non-relational database which allows to save data that might have different attributes and can change. Given the requirements we decided that MongoDB will fit our use case best. Not only does it allow to save records as flexible JSON-like documents, but also to perform single queries very fast. With this approach it is possible to easily extract the data after gathering it. That will come in handy when analyzing the data and interpreting it to make useful diagrams and heat maps.

One of the most important requirements is GDPR compliance, which enforces specific constrains on stored and sent data. Given this and after an insightful analysis of the requirements for the system we identified what data has to be stored in a database. A crucial aspect is that storing the images or sending them to other devices would violate GDPR, so the system should avoid that.

4.3 Image Processing Pipeline Design

The image processing pipeline is the foundation of the project (schema shown in Figure 2). The pipeline extracts accurate, high-fidelity data about the movement of the crowds on the edge. Initially, it is divided into four parts - prepossessing in section 4.3.1, object detection in 4.3.2, multiple object tracking in 4.3.3 and mapping in 4.3.4.

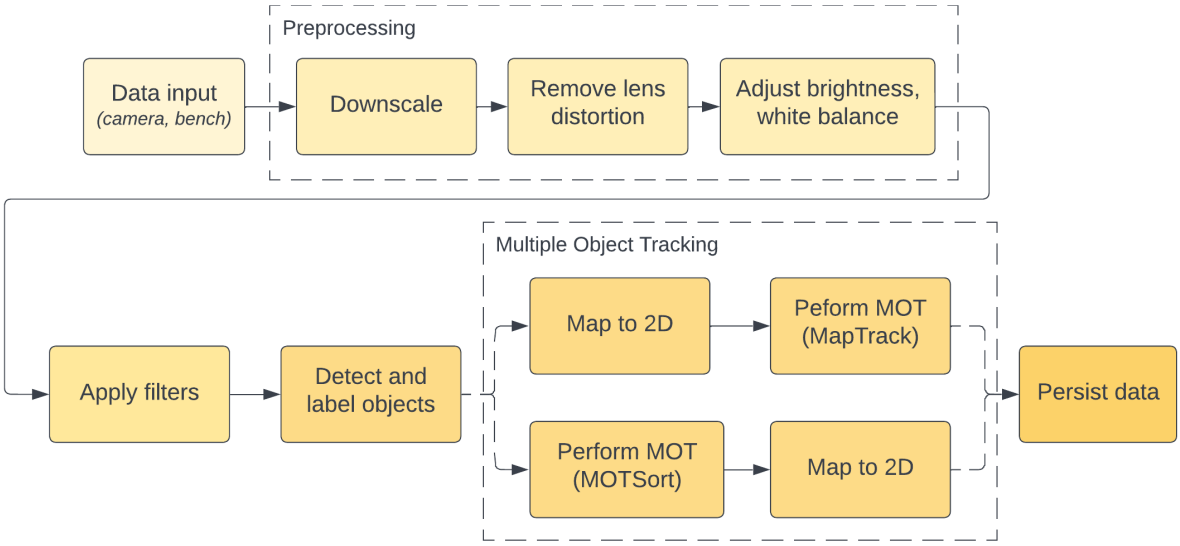

Figure 2: *The image processing pipeline.*

4.3.1 Preprocessing

The first stage of the image processing pipeline is preprocessing, which prepares images for the extraction of tracking data. This step enhances the image quality and increases the fidelity of the tracked objects. Preprocessing also downscales and compresses the visual feed, making subsequent pipeline stages more efficient.

To obtain non-distorted images from the camera it needs to be calibrated first. This is done by estimating camera parameters and adjusting them. The process is simple: it only requires taking few pictures of chessboard. Since detecting the pattern is quite easy the parameters can be estimated based on the images.

Object detection algorithm heavily relies on quality of the image, for example too dark image might make it impossible for the model to find the objects. Therefore the system adjusts the image by performing Histogram Equalization and applying other appropriate techniques.

4.3.2 Object Detection

Object detection is a well known problem, therefore there already are many existing solutions. We believe the best approach is to choose one of them as opposed to implementing object detection from scratch. Complexity of the problem is high, so developing efficient algorithm for embedded devices could be considered a large project itself while variety of available solutions allows to choose framework, that meets our requirements, in reasonable time.

One of the requirements for the system is that to work in a variety of environments. This means it has to take into account factors like: number of moving objects present at the same time, speed of those objects and lighting level. Therefore, we experiment with different models and choose the best ones depending on scenarios.

Before starting the implementation, it was necessary to research if deployment in a way that our client wants is actually viable. We were requested to deploy the solution on the edge, the prototype of the camera, which would run the software, using the same processor as Raspberry Pi 4 that has no AI hardware accelerators. Previous iteration of the project by the client was able to process a frame every 0.5s so we know that at least that performance is achievable. However with newer algorithms developed since then we hope to achieve better performance.

4.3.3 Multiple Object Tracking

Once objects are identified and labeled they can be tracked between subsequent frames. This problem has been heavily researched in the past decade and is well-known as Multiple Object Tracking (MOT). Algorithms that solve MOT are generally referred to as trackers. Most of the trackers implement a tracking-by-detection approach where the problem is split in to object detection and data association parts. However, we researched various approaches, and this section gives a brief overview of the best tracker candidates for final implementation.

There are three main ways to tackle this problem in which algorithms do data association: tracking by associating objects appearance using feature extraction (described for example in Multiple People Tracking by Lifted Multicut and Person Re-identification [10]) , tracking by modeling objects movement (used in SORT [11]) or mixing both (used in deepSORT [12]).

As the software has to be efficient for embedded devices we decided to use one of the motion modeling algorithms as they are proven to be less computationally demanding. Here are options that we consider the best as of now:

SORT
SORT[5] works based on calculating an assignment cost matrix and solving assignment problem optimally by using the Hungarian algorithm [13]. Most of the work in SORT is done with regard to calculating the cost matrix for the assignment problem. As stated in the SORT paper "The assignment cost matrix is then computed as the intersection-over-union (IOU) distance between each detection and all predicted bounding boxes from the existing targets" [11].

Furthermore the algorithm has additional parameters IOU_{min} and T_{lost}. IOU_{min} is a threshold below which assignment is rejected (since then it is highly unlikely) and T_{lost} represents number of frames after which a lost object identifier is terminated (it won't be tracked anymore, this prevents issue of growing number of trackers during runtime of the system). Even though the algorithm is from 2016 it still is very performant with MOT17 benchmarks claiming update rate of 143.3 Hz on a 4 core 2.5 GHz processor [14].

MAATrack
Modelling Ambiguous Assignments for Multi-Person Tracking (MAATrack) in Crowds is an algorithm that has improved its precision for crowded environments compared to other MOT algorithms. It is achieved by singling out so called ambiguous assignments (where the assignment has high probability of being wrong) and dealing with those cases separately and using the standard Hungarian method for the rest of the assignments [15]. This comes at some performance trade off but improves the precision significantly in crowded environments. Therefore depending on type of environment that our system will be deployed in it may be beneficial to use this as our tracking algorithm.

ByteTrack
Compared to the other two algorithms ByteTrack[6] stands out by considering all object detections in the scene (while most algorithms delete all objects with low detection scores). This will in general produce higher MOTA[7] scores compared to other algorithms. It is done in ByteTrack by first matching the high detection score boxes to tracks and then attempting to match rest of the objects to the remaining tracks. However, the downside of the algorithm is its performance with authors claiming 30FPS on MOT17 data set while using single V100 GPU [16]. This may then not be the best choice of algorithm but the approach of keeping all object detections and spliting data association into two stages could prove useful when trying to optimize our solution.

4.3.4 Mapping

Mapping the camera view to a bird view 2d plane is an essential part of the software. This feature allows the clients to see and easily analyze crowd movements.

[5]Object tracking algorithm, SORT implementation: https://github.com/abewley/sort
[6]Object tracking algorithm, ByteTrack implementation: https://github.com/ifzhang/ByteTrack
[7]Multiple Object Tracking Accuracy

The system should take the image from the camera and generate a map that represents the monitored space. Tracked objects would be visualized as moving points on the map. Each point should be labeled with its object id generated by the tracking.

Usually the cameras monitor the space from the corner of the room. That introduces an additional challenge to perform a perspective transformation to create a map. This can be done by sketching out the plane on the image that we would like to be mapped and then using OpenCV framework transforming it into the that we want.

4.4 Benchmarking System

In order to provide the guarantee of incremental performance improvements, the client requested to complete comprehensive benchmark-based reports of the system's performance in the weekly releases. Due to the aforementioned hardware constraints, the product development is overall likely to be heavily benchmark-driven. Therefore, we have decided to implement CI[8] jobs that will measure the performance of the image processing pipeline automatically, decreasing the development lead time.

Upon implementing a new algorithm, any developer will be able to either use CI runners locally or run the job on the VCS[9] remote. The job will then request the Raspberry Pi to run the test benchmark and return its result in the job command line. From this, the performance of the new component/pipeline can be easily evaluated.

Due to the expeditious iteration speed and the non-deterministic nature of the problems encountered, traditional software engineering testing approaches are often not suitable. Furthermore, as one of the main goals of the projects is achieving high performance, performing extensive algorithm benchmarks is crucial. That is why we decided to heavily rely on popular benchmark suites that quickly and effectively estimate the accuracy and performance of our E2E system [17].

4.5 Data Visualization

Another feature that the system should have is the user interface. It is mainly used to control the system's state and to visualize processed data. This UI displays the processed data from the 2d view taken from the entire set of collaborative devices and a histogram of object density in a certain spot.

Moreover, in purpose of testing we have decided to display the visual feed from any camera that will be in use. Additionally user will see the bounding boxes calculated by one of the object detection algorithms. This feature helps to gain deeper understanding of the software and its output.

[8] Continuous Integration
[9] Version Control System

5 Implementation

This chapter thoroughly describes the exact implementation of the software. First, the methodology of the development is briefly described in Section 5.1. Section 5.2 presents how the camera setup is accomplished before executing the pipeline. After that, section 5.3 provides the overview of the implemented pipeline and explicit descriptions of every separate part of this pipeline. The communication between cameras and database is described in section 5.4, while implementation of tracking across views is included in section 5.5. Section 5.6 explains workings of the control panel. The implementation of the benchmarking system is characterized in section 5.7. Lastly, section 5.8 the analysis and testing tools that were used to ensure the code quality.

5.1 Development Methodology

A development methodology that fits the project and its developers well can have an incredible impact on the success of that project. Unfortunately, it is not a simple task - the question of how to best organize work has been asked a countless number of times.

Due to the high complexity of the project, we decided to implement an exceptionally structured process. We employ full-fledged agile planning with well-specified responsibilities, granular tasks, and comprehensive descriptions. Research on unfamiliar topics is dedicated to specific people. The scrum master is fully responsible for creating and enacting a weekly sprint plan, during which they devote their focus to management. Their main task is to oversee every part of the product development process, enabling other team members' full potential. Accounting for various possibilities, working with every colleague on every part of the project, and proactively adapting the plan are their spotlight.

Even though everyone had different strengths, every team member has a similar contribution to the project. To illustrate that the final work distribution for the project is visualized in Appendix B.

5.2 Camera Setup

Before executing the pipeline, the user has to perform a setup which depends on specifics of each camera and the monitored environment. The setup can be divided into steps that allow to obtain parameters for different parts of the pipeline. Those steps are independent, so it is not necessary to maintain strict order of their execution. Moreover, the setup has to be performed only once per camera as all parameters are saved within the system.

Lens parameters
To obtain lens parameters which include distortion coefficients and camera matrix, the system uses a standard method available as OpenCV[10] tutorial [18]. To compute the parameters, we first take a few pictures of a chessboard from different angles. Then a python script computes the necessary parameters. Firstly the script finds the corners of the pattern. To make it easier, the images can be edited in an image editing software. The change can be seen on Figure 3. After the editing, the chessboard corners are sharper. Lastly, the script approximates distortion coefficients and camera matrix and saves it to a file for later use. Those parameters are primarily used to undistort the frames. The details of this step of the prepossessing are described in section 5.3.2.

[10] OpenCV: python framework for image processing https://opencv.org/

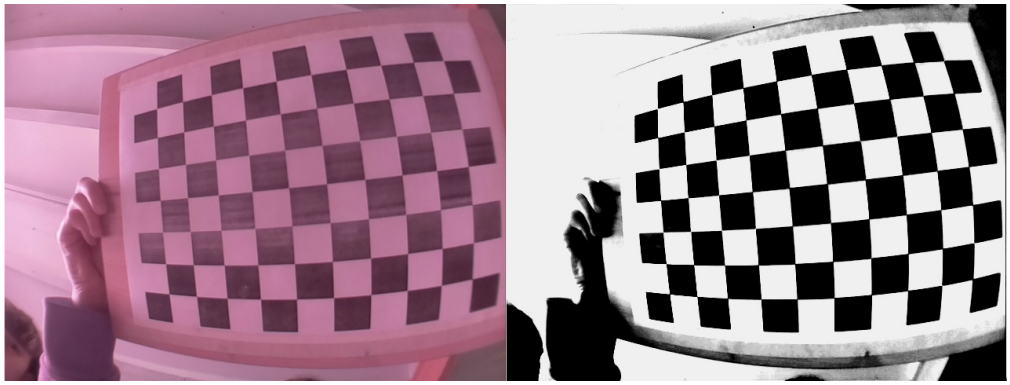

Figure 3: *Images before and after altering in image processing software. The chessboard is more pronounced and sharp.*

Environment map
Tracking across different camera views requires a precise map of the monitored environment. This means the system has to know the exact position of each camera and the plan of the space. Firstly origin point has to be picked together with x and y axis. Then each camera position is measured: x and y coordinate. We also had to measure the rotation of the camera with respect to the x and z axes. Lastly, the system uses the information about the horizontal angle of FoV[11] and the aspect ratio of the lens.

Background extraction
To execute filtering, further described in section 5.3.3, it is necessary to have an image of the background. It can be obtained by taking a picture of the empty environment, but as it is not always possible to get such image, the system contains a script that automatically extracts the background. The script receives video taken in the monitored environment and builds the background by removing dynamic elements and averaging the non-active tiles (considered background). The output is later saved to be used in the pipeline.

5.3 Image Processing Pipeline Implementation

This section describes the implementation of a core part of the software, the image processing pipeline. The first section 5.3.1 describes the general architecture of the pipeline. Next sections go into detail about the implementation of each of the pipeline components. Section 5.3.2 describes preprocessing, filtering is analyzed in section 5.3.3. Later section 5.3.4 describes object detection and section 5.3.5 multiple object tracking. Lastly, section 5.3.6 talks about the mapping.

5.3.1 Pipeline Architecture

The pipeline is specifically designed to enable full, real-time customization of the entire system. Using the chain of responsibility design pattern[12] it is easy to include and exclude parts of the system according to one's preferences. A class diagram that represents the implementation of the design pattern can be seen in Figure 4.

[11] Field of View
[12] About chain of responsibility: https://www.geeksforgeeks.org/chain-responsibility-design-pattern/

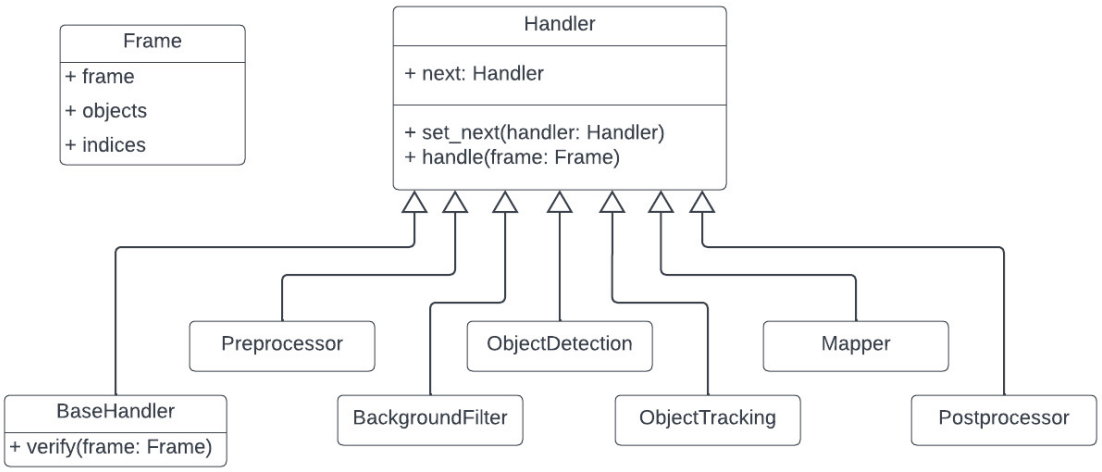

Figure 4: *Class diagram for the Chain Of Responsibility implementation in the system.*

The pipeline accepts and processes a single camera frame wrapped in a `Frame` object. A `Frame` contains a matrix of the currently processed image as well as a list of detected objects that change depending on the current stage of the chain of responsibility. Additionally, `Frames` can have more fields added by a module; for instance, the object detection module also adds the confidence of its detections.

A `Handler` class is an abstraction for a pipeline module. It has a `next` field, which specifies which module should be executed next. That field can be set by `set_next(handler)` method. The `handle(frame)` method contains a frame processing implementation appropriate to the module. When finished with its processing, this method will call the next module's `handle` method.

All concrete handlers, so all classes that extend from the `Handler`, represent one of the pipeline steps. Each can be further extended to add new implementations of the module. For instance, `ObjectTracking` can have multiple implementations of different tracking algorithms. A special case of a concrete handler is a `BaseHandler`, which is always the initial node in the chain. It has a `verify` method, which is called after building a complete pipeline to run it on an example frame and check that no exceptions are thrown. This way, the system can ensure that the pipeline is correctly setup.

The user can specify which modules to include in the pipeline from the UI or by manually editing the config file. When starting the system, a customized pipeline is created and verified. Then the camera feed is used as an input to the pipeline.

5.3.2 Preprocessing

To achieve the best performance in object detection and multiple object tracking, it is important to first prepare the image from the camera for later processing. Considering the functionality and implementation of the system, the preprocessing has to be split into two main parts. The first part focuses on undistorting images with the parameters obtained from camera calibration. After that, the software performs the image's contrast adjustment using the histogram equalization method.

Image undistortion
The image taken from most of the cameras has a distortion that is dependent on the camera lens and may negatively influence the performance of the system. As stated in [18], "Two major kinds of distortion are radial distortion and tangential distortion.". Both of these distortions make the image less similar to a real-world view, as the straight lines are curved. To avoid the negative consequences of using a distorted image, the system undistorts every image with the use of parameters gained during the camera calibration setup described in section 5.2. After that, the returned image is cropped, so that the number of pixels without meaningful content is minimized and the number of meaningful ones maximized. Figure 5 presents the images

before and after applying this method.

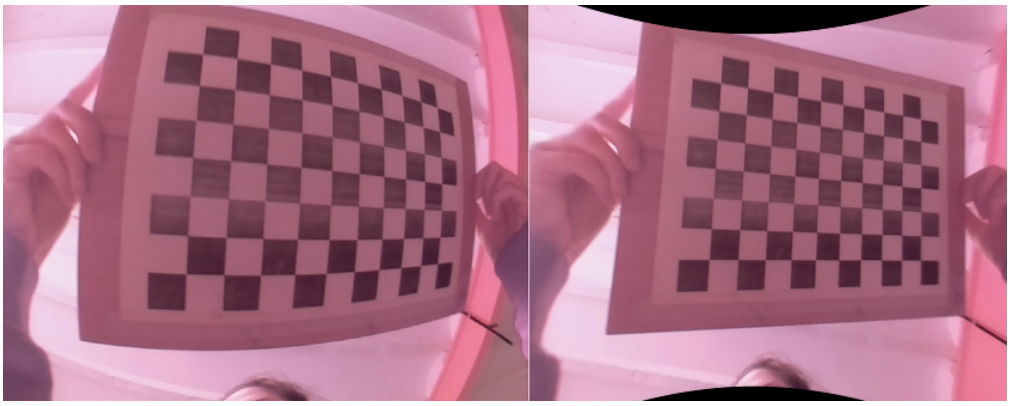

Figure 5: *Images before and after lens calibration.*

Histogram equalization
Another part of preprocessing that improves the performance of the pipeline is histogram equalization. According to [19], preprocessing based on adjusting the contrast is not supposed to make the image more appealing for the users or developers, but to make the image the most accurate for the system so that it can obtain the best result. Histogram equalization is a method that uses RGB image histograms for every color. These histograms present the number of pixels that each level of intensity has. The image that we want to obtain and that would give the best results for the system should have an almost equal number of pixels at all intensities.

To get the needed image, it is first clipped to delete the lowest and highest values of intensity. The reason behind that is color overlapping that has to be avoided. After that the representation of an image is changed from RGB to HSV[13] to be able to use OpenCV[14] function for histogram equalization, then the system performs it. Lastly, it converts the image back to RGB for later computations. This operation also increases the confidence value for detection done by object detection algorithm (4.3.2) and the achieved effect is presented in the Figure 6.

Figure 6: *Images before and after histogram equalization with corresponding bounding boxes and confidence values.*

[13] https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html
[14] https://opencv.org/

5.3.3 Filtering

Certain environments contain an abundance of images of people in the form of posters or screens that display static images of people. Because object detection algorithms are trained on carefully crafted datasets that avoid these types of places, they cannot discern people from depictions of people. The filtering module allows users to specify areas that should be dynamically blurred out, called filters.

Using the K-nearest neighbours (KNN) background substraction algorithm[15] the camera can understand apparent movement. Whenever a person walks over a filter, the area is not blurred out so that the moving person can be detected. Otherwise, the area is perceived as static, and the blur is applied, preventing false detection while detecting moving objects.

Figure 7 illustrates the filtering process in detail. The top-left image shows the original input, while the one on the top-right shows perceived movement. This output is then tiled - the image is divided into square tiles, which create a visual mask. The bottom-left image shows that mask. If the average intensity in a tile is strong enough, the tile and its neighbouring tiles will be replaced by the corresponding tiles from the original image. Otherwise, the black areas are replaced with a static background image with filters. The result of the filtering can be seen in the bottom-right image.

Figure 7: *The motion estimation process. Image source: [17]*

5.3.4 Object Detection

Object detection is performed on images in order to extract previously defined objects. The algorithms usually are based on neural networks which predict bounding boxes that define the size and position of detected objects. Those bounding boxes are outputted and can be shown on the image. Example output of object detection algorithm is shown on Figure 8.

[15]KNN algorithm: https://docs.opencv.org/3.4/db/d88/classcv_1_1BackgroundSubtractorKNN.html

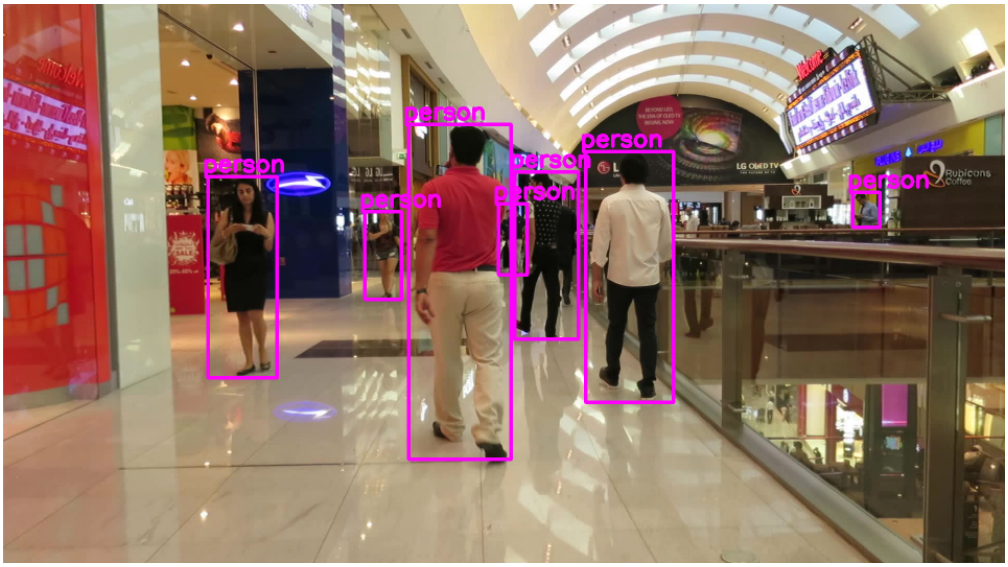

Figure 8: *Example output from object detection. The bounding boxes define the positions of detected people. Image source: [17]*

Object detection is an important part of the system and a significant bottleneck due to the high computational load. Therefore this pipeline module required extra care during development. As mentioned in section 4.3.2 the implementation of object detection is based on existing state-of-art frameworks. The choice of an appropriate framework was based on accuracy in AP[16] and latency measured in FPS. New objects tracking models appear frequently, so we used websites with benchmarks comparisons to quickly go through available options and compare them regarding use case scenarios. One of such websites was Papers With Code[17] which provides a complete overview of the accuracy and speed of different frameworks on various benchmarks.

One of the goals for the system was to allow different detection models to be integrated into the software. Even though benchmarks give a good overview of the performance, they are executed on devices different to the client's cameras. Therefore each chosen model had to be run on the target device. To aid the integration, we implemented an inheritance structure that is visualized in Figure 9.

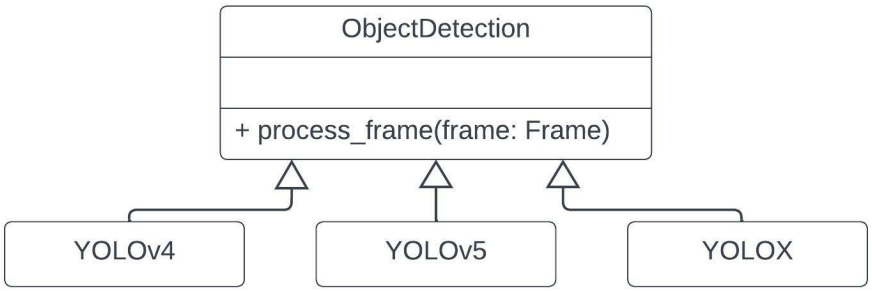

Figure 9: *Class diagram for the Object Detection classes.*

The structure includes `ObjectDetection` abstract class which is extended by each class that performs object detection. The abstraction ensures consistency between different implementations and acts as a guideline for future additions. Each class implements `process_frame` that receives a frame from the camera and returns

[16] Average Precision
[17] Papers With Code: Object Detection benchmark comparisons https://paperswithcode.com/task/object-detection

a list of predicted bounding boxes. The implemented classes have distinct parameters and possibly different methods depending on model type.

Since the object tracking is executed in real-time, we decided to explore models based on YOLO[18] method. The models implement one-stage detection; they predict bounding boxes for each object in the frame during one pass through a single neural-network [20]. Therefore YOLO-based models have low inference time, which is crucial for real-time tracking on embedded devices. Currently, the system can be run on 3 different open-source models:

1. **YOLOv4**[19]: The implementation of YOLOv4 is open-source and easily accessible due to its popularity. The base version had an acceptable performance on a standard laptop, but on target devices, the efficiency was under 0.05 FPS. Which makes it impossible to perform object tracking. We deploy YOLOv4-tiny, which is a reduced version of YOLOv4 with lower accuracy but higher inference speed. The client's cameras achieve around 0.8 FPS with YOLOv4-tiny.

2. **YOLOv5**[20] Second model included in the system is YOLOv5. Although the performance of YOLOv4-tiny was acceptable, it did not achieve specifiable accuracy. YOLOv5-nano is not only more accurate but also performs better on target devices with the efficiency of around 1 FPS. YOLOv5 also allows easy integration with TensorFlow Lite[21] which was not possible with YOLOv4. In the future, TensorFlow implementation would allow the client to connect an AI accelerator to the camera and achieve much higher performance. That is why we decided to already set up the implementation of YOLOv5 for TensorFlow Lite.

3. **YOLOX**[22] Lastly, YOLOX was added due to its high performance on different data sets. It had the best performance on Argoverse-HD in 2021 [21]. YOLOX-nano outperformed YOLOv4-tiny in terms of accuracy, achieving above 25% AP on COCO dataset [22]. It was also more efficient on embedded devices with around 0.7 FPS. Similarly to YOLOv5, it allows for easy integration with TensorFlow Lite, but at that point, the system does not support it.

5.3.5 Object Tracking

Object tracking is a module that follows the object detection. There exist various state-of-art algorithms for solving the multiple object tracking problem. To make it possible to use them in the pipeline the system has two wrappers: one for tracking on an image and the other for tracking on a map. A class diagram of this module can be seen on Figure 10.

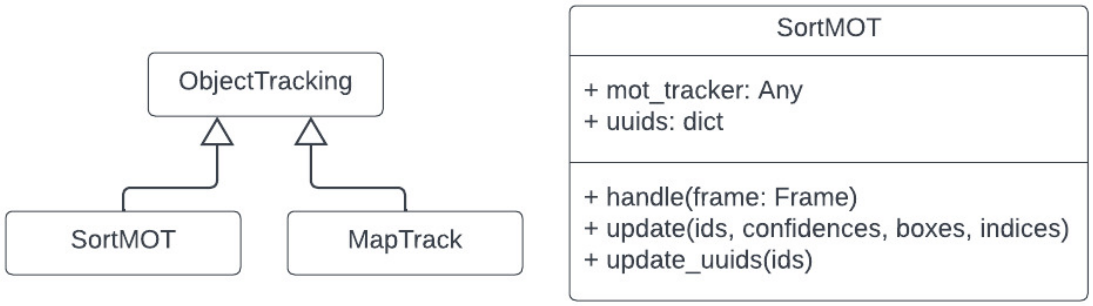

Figure 10: *Overview of Object Tracking classes. Note: MapTrack has an equivalent UML diagram to SortMOT.*

[18] You Only Look Once first - object detection model first introduced by Joseph Redmon
[19] YOLOv4 source code: https://github.com/AlexeyAB/darknet
[20] YOLOv5 source code: https://github.com/ultralytics/yolov5
[21] TensorFlow Lite: python framework for GPU optimization https://www.tensorflow.org/
[22] YOLOX source code: https://github.com/Megvii-BaseDetection/YOLOX

SortMOT

`SortMOT` class is a wrapper for SORT (described in section 4.3.3) and other MOT algorithms that extend from SORT. This class prepares the data from object detection for object tracking and performs object tracking. The `mot_tracker` field is a MOT algorithm object that contains the actual implementation of object tracking. This allows us to preprocess the data before calling the algorithm's methods. SortMOT also has a `uuids` dictionary, which contains a translation from the object ids yielded by the MOT algorithm to generated UUIDs.

When initializing a SortMOT object, it is necessary to provide which algorithm should be used as a `mot_tracker`. In the current version of the system, there are two options: pure SORT and OC_SORT [23]. The algorithm is specified in the pipeline configuration file.

The `handle` implementation of SortMOT calls the `update` method that takes care of making proper calls to MOT code to perform object tracking. SORT-based algorithms accept bounding boxes in a different format than the detection module outputs. Therefore, the first step in object tracking is formatting the object detection output. The bounding boxes from any YOLO algorithm are represented as a combination of the centre point coordinates of the box, its height and width. It is converted to coordinates of two points: top-left and bottom-right. Additionally, it is concatenated with the confidences of object detections which will be used by SORT to assign these detections to trackers most optimally. It is also important to filter the detections that should not be taken into consideration. The indices of the chosen detections are passed in the `indices` array. At the end, objects with a label different than "person" are also removed. The labels can be found on the `ids` list. After preparing the data, `update` can call `mot_tracker.update`. The output is a list of bounding boxes with a tracking id. Then, `update_uuids` is called to make sure that every tracked object has a UUID in the dictionary. Example output from the tracking on the image is visualized in Figure 11.

(a) *Frame i* (b) *Frame i+1*

Figure 11: *A visualization of an example output of object tracking across frames. Subfigure (a) represents a frame from a video and (b) the next frame from the same video. As can be seen the ids of objects remained the same. Image source: [17]*

MapTrack

`MapTrack` is another wrapper for SORT-like MOTs, but with an adjustment so that it can be used on points instead of bounding boxes. The class structure of `MapTrack` is exactly the same as for `SortMOT`. It is also necessary to specify whether SORT or OC_SORT should be used. However, the implementation of the `update` function is slightly different.

`MapTrack`'s `update` function has a different input and output than in `SortMot`. Instead of bounding boxes, it accepts points that represent an object on a map. To make it possible to use the tracking algorithm on points, each point gets an artificial bounding box around it which can be accepted by the `mot_tracker`. Similarly to `SortMot`, filtering for valid detections is necessary. Once the tracking is done, the artificial bounding boxes have to be transformed back to points. `update` also draws these bounding boxes on the provided as a parameter map. Lastly, the UUIDs of the tracked objects are updated. An example output can be seen on Figure 12.

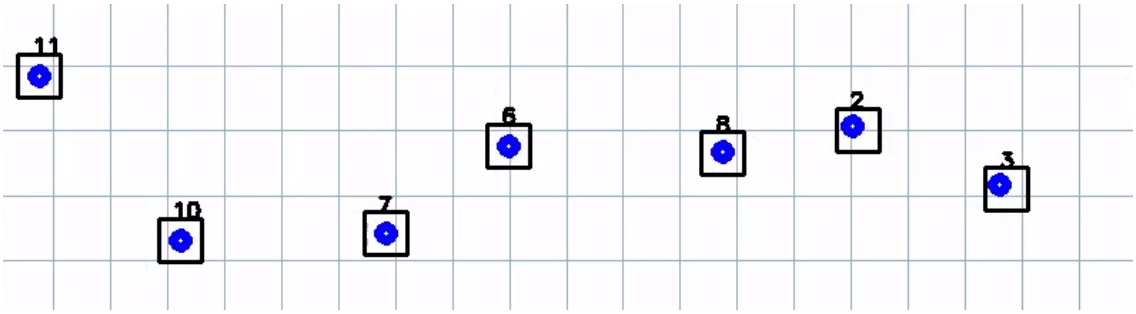

Figure 12: *Example output from MapTrack.*

This approach introduces an additional factor for the tracking performance - the size of the bounding box. If it is too small, the chance of the MOT algorithm losing the track of the object increases as the IoU is very low. If the bounding boxes are too large, they overlap too much, and the results become inaccurate.

5.3.6 Mapping

Mapping transformation allows the users of the system to see the camera view from the top, a "bird's", perspective. This feature is crucial for the software as it enables to analyze the density in spaces on a clear 2d plane. After choosing the calibration points, the image is transformed into a rectangular plane, and every detected object is mapped onto this plane as a dot. These dots are signed with an id and are mapped in a spot respectively to their spot on an original camera feed.

To achieve an accurately transformed image, the system first takes seven points from an input image which will later correspond to four vertices on a 2d plane. These points are chosen in the UI by the user before performing the calibration. With those points, the system performs the transformation, wraps the image, and calculates the transformation matrix and transformed image as the output.

Another operation that has to be done after birds-view transformation is mapping the detected objects onto a 2d plane. We first create a white background and offset for the rectangle. This rectangle is drawn on a plane and represents the area from the input image, limited by previously chosen points. Next, with the use of a transformation matrix, the system translates the coordinates of each detected point from the original image to the 2d mapping image and adds the offset. Finally, every point is put on the background image with different color so it is possible to observe the tracking of every object from the top. An example visualization of the mapping output can be seen in Figure 13.

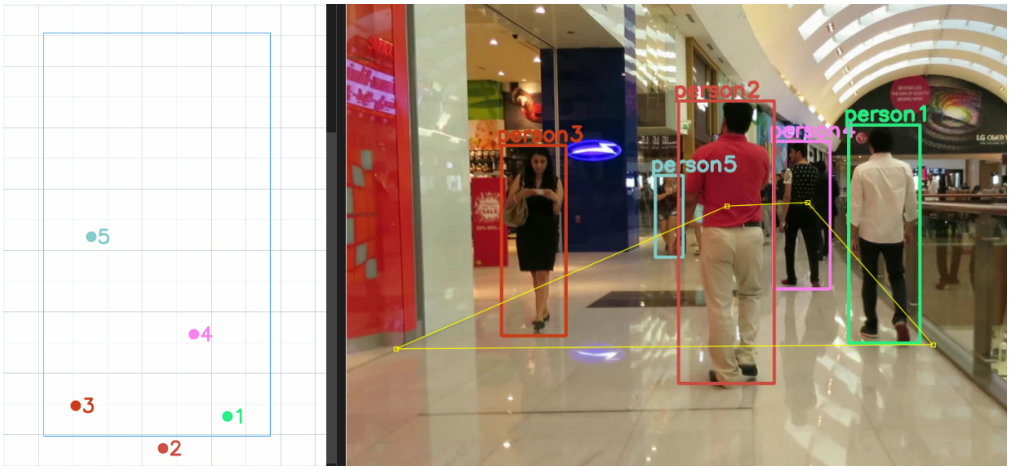

Figure 13: *Example output from mapping (with tracking performed on frames). On the right there is a frame from the camera feed and on the left the corresponding map generated by the system. Image source: [17]*

5.4 Communication Setup

Communication within the system is crucial as it allows tracking between multiple camera views. This section describes how the communication in the system is set up. First, section 5.4.1 explains how the database is set up, and section 5.4.2 describes the details of the intercamera communication setup.

5.4.1 Database

The database setup in the system is divided into two parts. First of all, each of the cameras runs its internal database. Next to that, there is a central server which runs a collective database.

The internal database run by each of the cameras is a no-SQL database MongoDB[23]. In this setting, MongoDB is very convenient as it stores data in JSON format, which is easy to handle in python using JSON library and dictionaries. The internal database consists of only one table. Therefore a relational database would add complexity that is not needed as, in this case, there are no relations to the model. For ease of deployment, we provide a docker-compose.yaml[24] file in the repository so that the database can be easily run. Though, if preferred, one could compile the MongoDB code for ARM[25] platform and run it without any abstractions added by docker, which could improve the overall performance slightly.

On the other hand, when it comes to the central database, the system uses a SQL database. We believe that this approach is better, as it allows indexing and retrieving items fast with regard to the timestamp. The central database is used to perform tracking across camera views as it collects and stores data from all cameras. Therefore the software will constantly retrieve items from it based on the growing timestamp. That is why in this setting, indexing will drastically increase the performance of the system, and for this reason, a database that supports indexing was necessary.

5.4.2 Intercamera Communication Setup

One of the requirements of the software was to implement tracking across multiple camera views. To achieve this, the cameras have to be able to send their data. After a few different ideas on how the communication should be set up together with the client, we concluded that having some sort of central server that will gather data from all the cameras makes the most sense. As an example, this makes it possible to move the server to the cloud and therefore will allow for bigger deployments with more cameras. However, it will still

[23]MongoDB https://www.mongodb.com/
[24]Docker and Docker-compose are tools for abstracting system specifications when building apps https://www.docker.com/
[25]Advanced RISC Machine - architecture of computer processor

be possible to have the server locally and run the network without a connection to the internet.

The communication of the cameras with the central server is setup with MQTT[26] and Node-RED[27]. MQTT allows for easy organization of data sent to the server. We use the MQTT pub-sub model to achieve the organization by making each of the cameras publish messages on their topic. The central server subscribes to all the topics and therefore is aware of where the data is coming from. Node-RED is a front-end for MQTT that allows for easier setup of the network that a user wants to deploy. Figure 14 shows an example of Node-RED set up with MQTT of two cameras with their topics. The top part is used for debugging, it allows to see if the injected messages are properly forwarded. In real deployment "msg.payload" node would be replaced with a node connected to the central database.

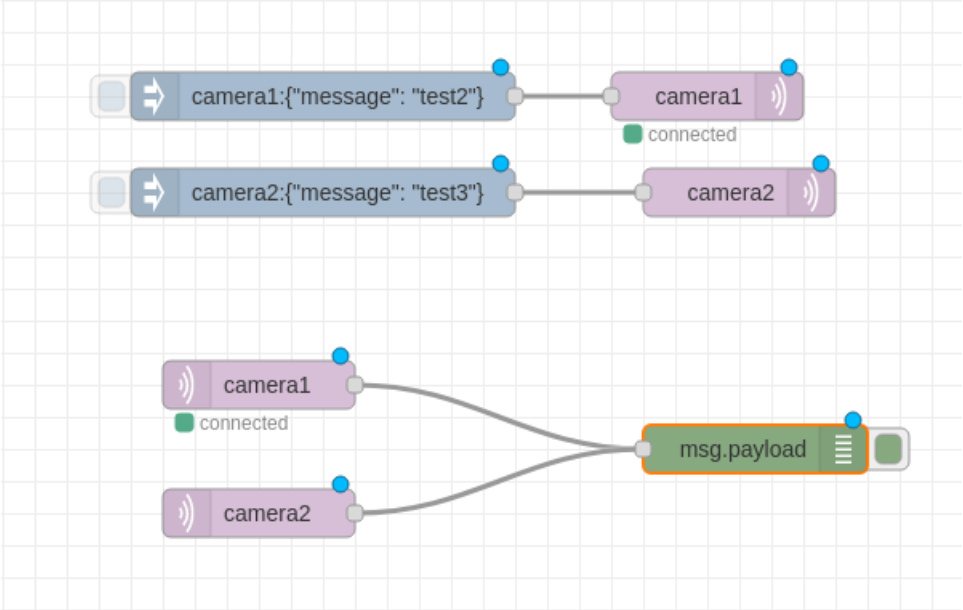

Figure 14: *Example setup of Node-RED with MQTT.*

It is also important to mention that the communication part runs asynchronously from the core of the system, which reduces the overhead on the core and therefore improves the performance. The communication is enclosed into a service. Our repository contains an example publisher.service file that can be run as a systemd[28] service. This is achieved through an additional attribute in the internal database that indicates whether a piece of data was published or not. The service queries the database periodically and publishes all unpublished data to the correct topic. Afterwards, it updates the attribute so that the data is not published multiple times. Figure 15 shows how the communication service interacts with other parts of the system.

[26] MQTT - standard for IoT messaging https://mqtt.org/
[27] Node-RED https://nodered.org/
[28] Systemd is a suite of basic building blocks for a Linux system. Most of GNU/Linux systems use systemd and therefore can run mentioned services

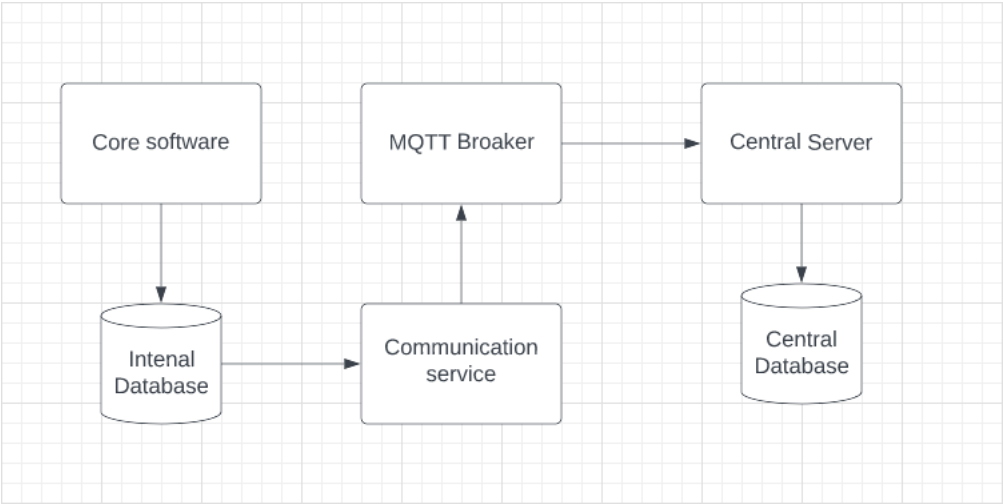

Figure 15: *Diagram of communication within the system*

5.5 Tracking Across Camera Views

This section describes how tracking across views was implemented in the system. First, subsection 5.5.1 describes how the area spanned by the camera is determined given the absolute camera position. The next subsection 5.5.2 describes the reconstruction of a map containing all the camera views and tracking objects on that map. Finally, subsection 5.5.3 shows how the combined data can be turned into a heatmap.

5.5.1 Camera View Extraction

Given the absolute positions of each camera and its rotation, the system is able to determine the coordinates of the area monitored by the device. Firstly, all the measurements have to be taken as described in section 5.2. Then it is possible to compute orthonormal vectors u, v, w that define camera orientation. Four rays that define FoV[29] of the camera are calculated using the following formula:

$$ray = v \pm u \times \tan\frac{\alpha}{2} \pm w \times \tan\frac{\beta}{2} \tag{1}$$

where α is the vertical angle and β horizontal angle of FoV. Intersection point with rays and ground plane defines the coordinates of the edges of the image. Since the top edges of the images are not necessary on the ground, we only define the coordinates of the bottom edges. As those coordinates are known in the relative coordinate system of the camera, it is possible to transform the points from the relative coordinate system to the absolute coordinate system of the environment.

5.5.2 Map Reconstruction and Tracking

After the absolute position of all camera views is known the map reconstruction is performed. To make sure that all cameras are visible on the map, it is created based on the maximal x and y coordinates of all cameras. Additionally, we draw camera positions and orientation on the map.

The tracking is done as a post-session simulation. This means that first, all data from the cameras is gathered and sent to a central database and only then analyzed. The simulation is performed as follows. All data is retrieved from the database and put into a matrix. Then the data is sorted ascending with regard to the timestamp. Afterwards, the data is processed frame by frame. As the data only consists of a timestamp, camera position and id of the camera from which the data comes from we have to artificially recreate frames. To do that, the system goes through the list in order of timestamp and retrieves all data within a time period

[29] Field of View

of a given timestamp. Then all those points are put onto a map.

Positioning points on the map is done with regard to the camera position and orientation. The data gathered in the database contains the position of the points relative to the camera the point was recorded on. The position is also scaled based on mapping configuration so that the coordinate scale is consistent across cameras. When mapping onto a common plane is performed, the points have to be translated to match a common coordinate system. This is done with regard to any point in the space. The only requirement is to measure x and y coordinates of each of the cameras from the chosen point as well as an angle to a chosen axis parallel to the floor plane.

After all points within a time frame are drawn on the map, MapTrack is executed on the frame. As described in section 5.3.5, MapTrack is a SORT-based object tracking algorithm that works on a map. We repeat the process of frame reconstruction and running MapTrack until all points are processed, which results in a simulation of the tracking on the whole session. Figure 16 represents an example output of tracking across views on one processed frame.

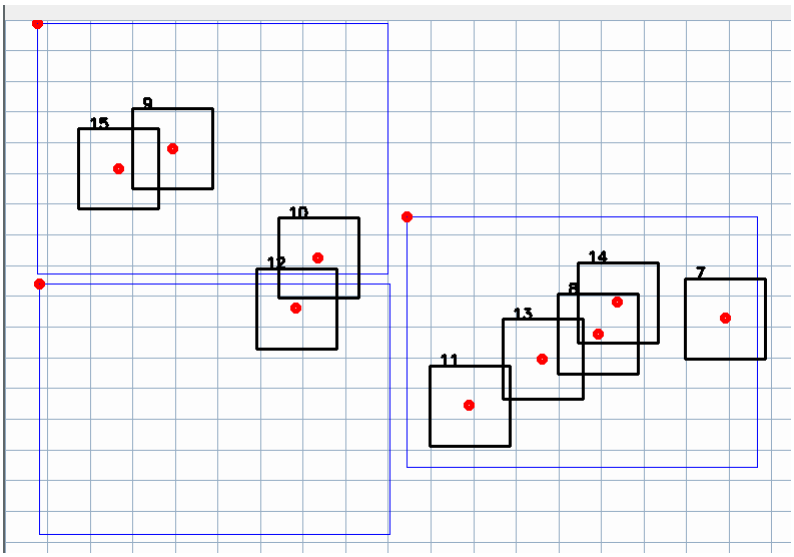

Figure 16: *Example output of tracking across multiple views.*

5.5.3 Heatmap Generation

To provide meaningful insights, we give the end-users an option to generate a density heatmap. Given the deployment data, the script measures how often people are present in an area. If there are a lot of people, this spot on the output is very bright, and conversely, the spot is black if there were no people present. Figure 17 shows an example.

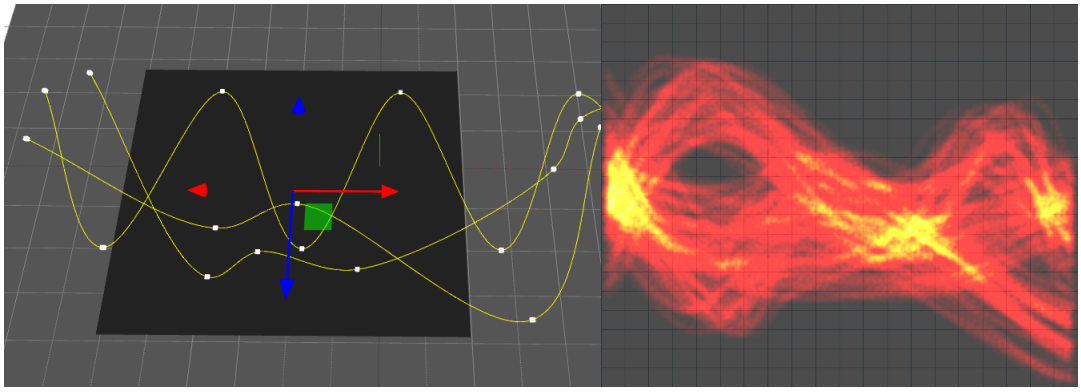

Figure 17: *Density heatmap generated from the simulation data.*

The image on the left shows the simulation tool. The yellow tracks are followed by virtual objects, while the black rectangle constrains the range of the saved tracking data. After running the tool, the simulation data is fed into the heatmap script, resulting in the right image.

5.6 Control Panel

When deploying cameras, the control panel is the easiest and fastest way to configure and verify the health of a fleet. Each embedded device hosts its own Flask[30] server that connects it to the local network and enables remote control. The published website is made up of two major pages - the calibration page and the tracking page.

Calibration page
When a user sets up a camera in a new environment, the pipeline has to be calibrated through the calibration page shown in Figure 18. As per section 5.2, this involves bird's view transform and scale calibration. To begin, the user sets the source of the input, either an example video or the device's camera. To start calibration, the user clicks the `Calibrate mapping` button. The transform trapezoid and scale reference can be set by clicking on the loaded image. The first four chosen points represent the trapezoid. The following three points need to be chosen in such a way that distances on the y and x axis between the points are equal to one meter. Having finished the procedure, the user can then click on the `save` button to persist the settings on the device.

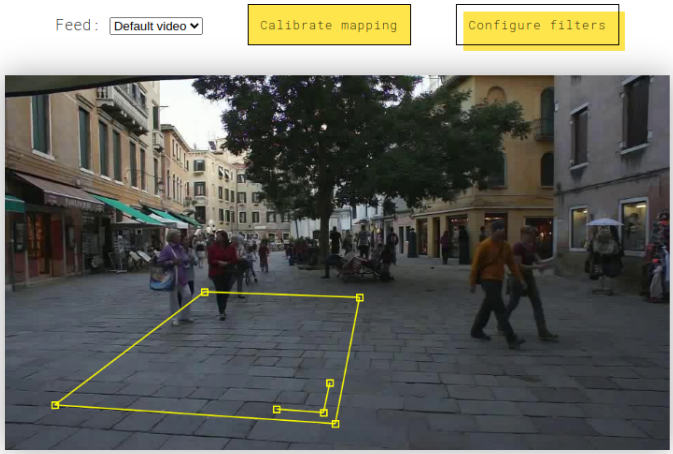

Figure 18: *Part of control panel's calibration page. Image source: [17]*

[30]Python's Flask: https://palletsprojects.com/p/flask/

To configure filters from 5.3.3, the user can follow a similar procedure. By clicking on the `Configure filters` button the mapping calibration is cleared, allowing the user to select the top left and bottom right corners of each of the filters. There is no limit on the number of filters that can be picked. As the control panel remembers input data, the user can freely switch between the different configuration modes. Identically as with the transform calibration, the configuration can be easily persisted by clicking the `save` button.

Tracking page
Having set up the camera, users can access the tracking page to pick the pipeline modules and run the system. Figure 19 shows this page. The menu on the left allows users to choose the source of the feed, either a default video or the device's camera. The checkboxes below correspond to individual pipeline modules to be turned on or off. Unfortunately, complex module configuration is impossible through the UI, and the user must set it manually using the pipeline config.

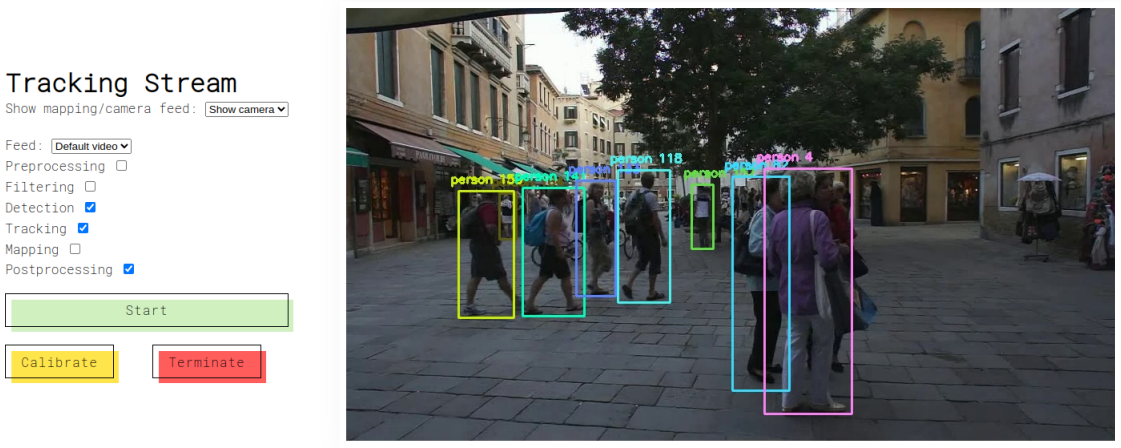

Figure 19: *The UI allows you to choose device's pipeline modules. Image source: [17]*

Once everything is ready, the user can click the `start` button to enable the pipeline. The processed images will appear on the right in real-time. If mapping is enabled, the user can switch to the mapping feed by using the dropdown under the `Tracking Stream` caption. The mapping feed will display a 2D space projection with tracked objects represented as points.

5.7 Benchmarking System

The benchmarking system is based on the MOT17[31] dataset, which provides 14 video sequences with assigned tracking bounding boxes. Using the robust TrackEval [24] dataset evaluator the benchmarking system can easily extract various tracking metrics, such as HOTA [25] and IDF1 [26], when provided with input from our pipeline. Moreover, we install the benchmarking suite on the target embedded device, the Raspberry Pi 4 Compute Module[32]. This way, it is possible to quickly compare the different object detection and MOT implementations for accuracy and real-world performance.

To make running the benchmark simple, we introduce auxiliary bash scripts that automatically install required packages, download data and convert the output from our pipeline to the format the evaluator accepts. To best simulate real-world conditions, the installation scripts downscale video sequences to 640x480px resolution, identical to the resolution of our devices' cameras. Once the benchmark is installed, the user can specify data sources for the object detection algorithm. After that, they can plug in their pre-configured pipeline and run the benchmarking script to get the results.

[31] MOT17 website: https://motchallenge.net/data/MOT17/
[32] Raspberry Pi 4 Compute Module datasheet: https://datasheets.raspberrypi.com/cm4/cm4-datasheet.pdf

Furthermore, the benchmarking system is integrated into our CI. The last optional step is the `bench` stage that runs on the Raspberry Pi with the help of the GitLab runner[33]. Whenever new code is uploaded, but especially when a developer makes a new merge request, they should trigger the `bench` stage. The spawned CI job automatically runs the pushed code against the benchmarking system, evaluating the speed and precision of the tracking system. Note that the most important benchmarks in our repository are tagged for easy access.

5.8 Code Quality

High-quality code is reliable, easy to understand and concise. A uniform style structures and standardizes the code and maintains the consistency between files. We use different quality checking tools and strategies to ensure that the code meets our client's standards. Section 5.8.1 describes our approach to writing consistently legible code, and section 5.8.2 elaborates on the different techniques that verify the correctness of the code.

5.8.1 Style Conventions and Maintainability

The software follows strong stylistic conventions that prevent bugs and decreases feature lead times. Because the code is designed with maintainability in mind, the repository contains large amounts of technical documentation and documents describing our development process. This section briefly describes our stylistic principles, how they are enforced, and the types of documentation provided.

Stylistic conventions
When developing software in a team, one must set guidelines that prevent mutually incompatible code from being created. In our case, the code follows the Google Python code style guide[34]. This guide contains rules that define commenting conventions, naming schemes, proper indentation, orderly documentation, and much more. Additionally, we made slight adjustments to default rules to fit our preferences. Overall, this guide ensures that a consistent code style is maintained across all files.

Static analytics
To verify the style we use the `pylint`[35] static analysis tool for Python. `Pylint` automatically evaluates the codebase's style score on a scale from -10 to 10. If there are any warnings or errors, `pylint` produces output that helps developers easily identify and fix existing problems. Each file containing python code must conform to predefined requirements. Otherwise, we cannot integrate it into the software.

Documentation
Because documentation is paramount to maintainability, we prioritize creating extensive, easy-to-read resources. Each central directory contains a thorough description of its contents in a `README` file. It explains the general idea behind the individual part of the system, provides technical instructions, and credits external sources. Because GitLab automatically displays `README` files while navigating the repository, it is trivial to find relevant information and code. Furthermore, the `docs` directory contains comprehensive documentation of our process, including agendas, meeting notes, and planning documents. Together with the code revision system and agile tools, our documentation makes it trivial to see and analyze our progress from the start of the project.

5.8.2 Testing

While the benchmarking system (5.7) tests the performance of the system, it is also necessary to add testing that allows checking smaller parts of the code. In that way, it is possible to locate the problem when an unexpected behaviour of the system occurs. This section describes the testing techniques used to ascertain that the codebase works well.

[33] GitLab runner: https://docs.gitlab.com/runner/
[34] Google Python code style guide: https://google.github.io/styleguide/pyguide.html
[35] Static analysis tool for python https://pylint.pycqa.org/en/latest/index.html

Unit testing
Since the project often relies on existing open-source implementations, standard testing methods are not always suitable. The unit tests mainly focus on mocking the external libraries, verifying that they are called correctly and their output is correctly reformatted (if necessary), and returned. The mocking is done using unittest framework[36]. If a method raises an exception somewhere, there is also a unit test that calls the method with input that should trigger the exception and checks if that was the case.

Integration testing
Integration testing was done to ensure that the system is working when multiple modules of the software are running. We specifically check if the pipeline gives an exception when there are some missing or wrong attributes. The tests also verify if the pipeline is built correctly and the system executes operations without errors or exceptions.

Branch coverage
With branch coverage, it is possible to check the percentage of code branches executed during testing, and it can clearly show which parts of the code were verified while running tests. That is why it was important to use branch coverage and get a sufficient score while testing the codebase. At the end of the project, we have achieved 83% of branch coverage for all the modules combined, and the full percentage table can be found in table Appendix C. We have omitted some modules in our repository because part of this code is an open-source code, and other files are testing modules.

System testing
During the development, it was also important to test the code on the cameras themselves. They are significantly different from a normal PC, and we could not afford to abstract out the hardware with tools like Docker[37] since it would add too much overhead for a slow processor that the cameras are equipped with. Therefore with every release, we made sure to deploy the system to the cameras and thoroughly test every feature we added. Sometimes, developing new features was done with multiple deployments as it was the only way to test and debug specific parts of the software, for example, the tracking across camera views.

Runtime verification of image processing pipeline
As described in section 5.3.1 before creating the pipeline is verified. The pipeline processes an example frame to catch any potential exceptions before running the system on the actual camera feed. If the verification was not successful, the system does not allow the continuation of the process.

CI pipeline
CI pipeline provides an easy overview of code quality. The pipeline has three stages. The last one performs the benchmarks and has to be triggered manually. However, the first two stages of the CI pipeline: the static analysis and testing stage, both run on GitLab branches automatically every time the code is updated. Static analysis is performed using pylint. In the testing stage, all unit tests are run but not integration tests. Those tests are computationally intensive and require additional data which is not uploaded to GitLab. The pipeline fails in case the stages yield an error. Additionally, after a successful run, GitLab displays branch coverage computed during test execution. All aforementioned features allow the teammates to quickly check if the code on GitLab conforms to quality guidelines.

[36] Unittest documentation: https://docs.python.org/3/library/unittest.html
[37] Docker https://www.docker.com/

6 Product Discussion

In this section, we discuss the resulting product after 10 weeks of development. First in section 6.1 we layout an overview of all completed requirements. Next in section 6.2 we describe the final achieved benchmarks of our system. Finally, section 6.3 contains a general discussion on the project and the state of the product.

6.1 Overview of Completed Requirements

At the end of the implementation of our system, we have managed to complete a major part of all requirements and the full list with them can be found in Appendix D. We completed all of the **must-have** requirements. The system detects an object in a camera view, tracks them, and performs a bird-view transformation along with mapping every object to a 2d plane. Since SORT takes care of tracking entirely, storing the velocity and direction is not necessary. Therefore we store only data of identified objects (UUID, position and timestamp). What is more, a basic user interface was implemented, so that it shows the camera feed together with the postprocessing - bounding boxes and object id.

Additionally, we also implemented more than half of the **should-haves**. The false-positive detection risk was minimized by filtering. The UI was improved with the possibility to configure the pipeline and to set the calibration values (mapping calibration points, etc.). Cameras can communicate with each other to send the detection information they gathered and session data can be retrieved securely by making an API call. The system was also optimized for embedded devices. During the discussion with the client, we decided that the computations will be performed on an external server so there is no need for collecting all data to one camera anymore. Other requirements were also not appropriate or turned out to be too time-consuming for an only 10-week project.

Moreover, we managed to complete one of the **could-haves** in full, and one in part. In the current implementation of the system, the camera network is not affected if one camera loses connection to the network. The system is also capable of generating density heatmaps from the gathered tracking data, but it cannot visualize the flow of people.

6.2 System Benchmarks

At the beginning of the project, we set a series of embedded device performance and accuracy targets that the system should meet. Unfortunately, as we could not use specialized hardware, these targets could not be met. Given the circumstances, we are confident about the tracking performance. Based on the benchmarking system introduced in section 4.4, Appendix E presents a number of benchmark runs for different pipeline configurations that quantize how fast and well the system works. Before analyzing the full results in detail, we draw three trivial conclusions about the pipeline from the appendix data:

MOT algorithms do not impact performance
Due to the GDPR limitations, used MOT algorithms cannot utilize computationally expensive feature extraction techniques. Instead, they are based on Kalman filters that are extremely fast. This means that regardless of the variation of `SORT` used the inference speed is unchanged.

The number of detected objects does not change inference time
Provided sequences have very varying average object density (between 8.3 and 45.3). In case a computationally expensive MOT techniques were used, we would expect the inference times to vary significantly. However, with `SORT` the inference time for each of the scenes is virtually the same.

Changing the SORT impelentation has negligible impact on accuracy
Contrary to our expectations, `OCSORT` does not provide significant accuracy gains. Despite being a more advanced, newer version of `SORT` the acquired results are comparable. We speculate that if the quality of object detection was higher, the difference would be sizeable. Tweaking the different hyperparameters could also prove to provide an improvement.

Using these conclusions, we can compress the results to show relevant metrics only. Table 1 presents a short version of the obtained results. Please note that `YOLOv5` is not included as it is inherently incompatible with benchmark's sequence resolution.

Configuration	HOTA	MOTA	Interference FPS
YOLOv4 tiny + OCSort	27.2	24.9	**1.91**
YOLOv4 small + OCSort	**35.2**	34.7	0.26
YOLOX nano + OCSort	34.0	**34.9**	1.27

Table 1: Compressed pipeline benchmark results.

Unsurprisingly, the smallest model is also the fastest. `YOLOv4 tiny` provides interference at nearly 2 FPS on the embedded device, beating `YOLOX` by 50%. `YOLOv4 small` and `YOLOX nano` have extremely similar performance, but the latter is nearly 5 times faster. Therefore, we deem `YOLOv4 small` unfit for CPU inference, and instead recommend using `YOLOX nano`. The faster alternative provides little to no potential for improvement, and is not intended to run on the edge. However, the `YOLOX` family is specifically designed to accommodate embedded device inference[38], providing a better ecosystem for both general-purpose CPUs and specialized GPUs and ASICs[39].

6.3 Discussion on the Process and Final Product

In this section, we would like to present some of our thoughts after developing the software for 10 weeks. The project proposed by S4FE was very interesting and thought us a lot. On the other hand, 10 weeks was not nearly enough to develop high-quality software and our version should be treated as a prototype.

The project proposed by the client was broad in many ways and we had to choose in which direction we wanted to develop our software. Of course, all the decisions were consulted with the client so that the final product aligns with the client's expectations as much as possible. In general, we leave the project developed more in some areas than others since perfecting it in every way would require at least twice the time.

First of all the project required a lot of research. Object detection and object tracking are fields of computer science that are constantly developed these days and the state-of-art changes with new developments. Therefore before even starting any coding we spent over a week researching in the field and getting to know what challenges we would face and how to solve them. Even after that week of constant research and we still had to go back and research more before coding different parts of the software.

Another observation about the project is the fact that we failed in a few areas. For example, during the design phase, we planned out trying a few object tracking algorithms. To be exact we wanted to try MAATrack [15] and ByteTrack [16]. As it turned out the code for MAATrack is not open-source or not yet published. We tried to implement it using the pseudo-code provided in the white paper but after a long time of trying with no effects, we had to abandon the issue to work on other requirements for the software. With ByteTrack we found the code but were unable to install it due to it not being compatible with newer python versions. While ByteTrack would most likely turn out to be slow for embedded deployment we think that MAATrack if implemented correctly would improve the performance of the system, especially in crowded environments.

The most lacking feature of our software is the tracking across camera views. While tracking on simulations works well tracking on real data is problematic. This is due to inaccurate mapping and camera view determination. Next to that the system does not handle overlapping camera views. This would require implementing an edge case where duplicate detections from area of overlap are merged together. Therefore to make the feature reliable it should be developed further and the mapping should be improved.

One of the most important things we got right was the principle of modularity. Separation of concerns proved to decrease the system's complexity, increase testability, and set clear boundaries between the different log-

[38] YOLOX light models: https://github.com/Megvii-BaseDetection/YOLOX#light-models
[39] Application specific integrated circuits

ical contexts of our software. Thanks to implementing the chain of responsibility, we could leverage OOP principles and avoid a massive amount of code duplication. Some pipeline modules were implemented as interfaces, further increasing the ease of use.

Since our project utilizes a wide array of state-of-the-art, modern resources like object detection algorithms, we rely on many external dependencies. Even though Python comes with a package manager, we ran into problems compiling dependencies over different branches. After a couple of weeks, we had to coordinate standardizing dependency versions as some of the functionalities did not work, e.g. due to different Python versions. Instead of resolving such issues when they have already grown large, we advocate that developers agree on the tech stack used from the very start. If new dependencies are added during the development, the team should ensure that their versions are not conflicting and enable the required functionalities.

Lastly, we would like to discuss the performance. Our goal was to improve the FPS of the software compared to the client's previous iterations. Unfortunately we were unable to do so as our system will at most show similar performance. Our biggest hope for performance improvement was to use TensorFlow Lite[40] in combinations with AI accelerators such as CoralAI [41]. Our client ordered some of those accelerators and there were supposed to be shipped during week 3. Unfortunately, the shipping got delayed and we were unable to try running our software with those accelerators. However, the software is mostly ready to be run on those devices. Of course, we had no chance to test that so it may need some minor adjustments.

[40] TensorFlow Lite https://www.tensorflow.org/lite
[41] Coral AI https://coral.ai/

7 Conclusion and Recommendations

In this section, we provide the closure of the report which we derived after completing the product. In section 7.1 there is detailed conclusions to the whole report and in subsequent section 7.2 lists recommendations for future development of the system.

7.1 Conclusion

The aim of this report was to describe the process we went through the past 10 weeks while developing a tracking system at the request of the S4FE startup. The problem of tracking crowds in everyday settings is a difficult one to solve. There are many potential privacy concerns that need to be considered when designing a solution. Additionally, the solution must be practical and affordable in order to be adopted by businesses and organizations. In the end we were able to create well functioning prototype that meets all must-have and more than half should-have requirements.

From the start, we wanted to make sure our goals are clear and achievable. To this end, we conducted a thorough problem analysis through meeting with the client and a literature review. We found that many existing solutions use various technologies, but the specific use cases for clients of S4FE are yet to be implemented. The goal was to provide a low-cost and privacy-preserving system that would work well on embedded devices. The solution could be incorporated into the client's system so it had to maintain high code standards.

Conclusions from problem analysis were taken into account during requirements elicitation and product design. We analyzed the possible stakeholders of the project to derive a comprehensive and detailed list of requirements. The product design is directly based on the requirements and further research. From the start, we paid attention to the ethical concerns that our project creates. Those were exhaustively analyzed and taken into account during the design phase, for example, decisions concerning data storage were directly influenced. Being aware of the importance of a good plan for such a complex project, we created the architecture of the system with an overview of the image processing pipeline and connected to it benchmarking system and data visualization software.

When looking back at the design phase we are satisfied with the initial plan. Composed architecture is well-structured and functional. At the same time, we underestimated the complexity of certain tasks and the time needed to complete them. There were many obstacles along the way which we did not foresee. Lack of experience in the computer vision and multimedia analysis field made planning even harder. This resulted in adjusting certain goals to fit within the time limit we had. For example, we were not able to achieve better performance compared to S4FE's existing system. It is worth noting that those are natural struggles in projects with a high level of complexity. So even though there is room for improvement we consider the planning carried out well.

With product design in place, it was possible to start the implementation of the whole system. Even though the planning phase was over we still had to conduct research on state-of-art frameworks that would fit our use case. The research was done for components such as object detection, multiple object tracking, and the setup of communication between devices. Also during the implementation phase, the architecture of the pipeline and all its components were created, together with benchmarking system, UI, and setup of communication between cameras. The system was tested on target devices after arriving in Barcelona and carrying out an appropriate setup. We used 3 cameras so the deployment was quite simple, this could be improved in the next version of the system. In the end, we were able to create a system that can serve as a well-functioning prototype.

We spent considerable amount of time reflecting on what we achieved in terms of implementation. The implementation was the largest part of the project so we deem it the most important section in the report, with the most conclusions. We analyze completed requirements the performance of the system measured on the popular benchmarks and lastly, discuss what could have been done better. After summing up the met requirements we were able to analyze possible extensions to the system.

We would like to elaborate on recommendations in the next section. With all work we have done we recognize the project is quite complex and would benefit a lot from further development, we want to explore all the

different possibilities for that. One of the ideas for the system was to make it easily extensible so many of those suggestions would be easy to develop. Along with improving performance and extending the system to more cameras, we could provide more insightful data to the possible users.

7.2 Recommendations

Unfortunately, despite the huge amount of work done our system is far from finished. Thanks to the vast amount of experience gathered throughout the project, we feel strongly about what can be improved and the correct approaches to solving a similar problem. This final section aims to provide an overview of features we'd like to see implemented, as well as general guidance on how to pursue them. We hope that our deep understanding of the topic proves to be a good guide for further development.

Massive deployments
A primary end goal of our software is to enable massive camera deployments with dozens of cameras working together. Ideally, the area of large buildings like convention centers could be fully covered by a single camera network. Such a vast amount of rich data would inevitably drive insane value for customers at a low cost. However, to make such deployments a reality, we need to increase the system's reliability and decentralize parts of the system. The former is done by fixing minor architectural problems and bugs that plague a quickly developed system. If we devoted more time to fixing existing problems, the system failure rate would go down significantly, allowing large amounts of cameras to work continuously for hours. Furthermore, the database has to be decentralized. In order to perform well in a network with tens of devices, it has to run in a distributed manner. Finally, parts of the pipeline such as tracking and mapping could be moved to a central processing device, decreasing the load on the edge devices.

Using edge AI accelerators
Currently, the tracking performance is satisfactory but far from perfect. Due to the nature of object detection algorithms, specialized hardware is needed to enable highly parallel, efficient computing with low energy consumption. As our current devices run the software on general compute CPUs, it is nearly impossible to reach real-time performance, resulting in lower fidelity data. To fix this issue, we would like to work with highly specialized edge AI accelerators like Google's Edge TPU[42] or Nvidia's Jetson TX2[43] instead. These devices provide between 2 to 3 magnitudes higher theoretical performance, providing interference at hundreds of frames per second[44]. To enable their use, we would have to modify the existing hardware, adapt the software to use the accelerator's framework, and test the system in real-world scenarios. However, despite the effort required, we are optimistic that the specialized hardware would be a substantial overall improvement.

Building analytics tools
The raw information we collect is not meaningful on its own. Supplementing the tracking system is a set of visualization tools and general processing techniques that helps people understand the data. However, aside from `MapTrack` or heatmap generation, we recommend extending the analytics suite much further. Similar to how wind flow is visualized, we could visualize the flow of people. We want to provide web tools that analyze data and find insights between the tracking data and, e.g. time of the day and year or weather conditions. Finally, we firmly believe that creating a tool that automatically combines the tracking data with the floor plan of the area would greatly aid end-users in deriving essential insights. Together, these extensions would make the tracking system much more valuable and relevant to S4FE's clients.

Embracing the ease of use
As a team of experienced software developers, we cannot forget that non-technical users might struggle using command-line interface (CLI) tools widespread in our project. Moreover, some techniques like the calibration procedure can seem complicated. Therefore, a part of the development efforts must be directed towards embracing the ease of use. Making sure that anybody can operate the tracking system starts with expanding the control panel and creating simple but comprehensive running instructions. Help pages and explanatory videos could aid the technical project documentation. Moreover, we recommend replacing the current calibration procedure with intelligent algorithms that can set up a device with little to no human input. The mapping and scale derivation can be easily replaced by QR code scanning, while the need for

[42] Google's Edge TPU: https://coral.ai/docs/m2/datasheet/
[43] Nvidia's Jetson: TX2 https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/
[44] Nvidia Jetson benchmarks: https://developer.nvidia.com/embedded/jetson-benchmarks

false-positive filters can be automatically detected. Combined, these changes would considerably decrease the expertise required to run our system.

References

[1] H. Yin, T. Sun, L. Yao, *et al.*, "Association between population density and infection rate suggests the importance of social distancing and travel restriction in reducing the COVID-19 pandemic," en, *Environmental Science and Pollution Research*, vol. 28, no. 30, pp. 40 424–40 430, Aug. 2021, ISSN: 1614-7499. DOI: 10.1007/s11356-021-12364-4. [Online]. Available: https://doi.org/10.1007/s11356-021-12364-4 (visited on 06/19/2022).

[2] European Commission, *Ethics and data protection*, 2021. [Online]. Available: https://ec.europa.eu/info/funding-tenders/opportunities/docs/2021-2027/horizon/guidance/ethics-and-data-protection_he_en.pdf.

[3] *CrowdVision - Automated passenger tracking using video analytics*, en-US. [Online]. Available: https://www.crowdvision.com/solutions-airports/ (visited on 06/18/2022).

[4] *CrowdVision - Automated customer analytics for retail stores and malls*, en-US. [Online]. Available: https://www.crowdvision.com/solutions-retail/ (visited on 06/18/2022).

[5] J. Ding, *Defining Accuracy for Street-Level Mobility Data*, en-US, Feb. 2020. [Online]. Available: https://numina.co/defining-accuracy-for-street-level-mobility-data/ (visited on 06/18/2022).

[6] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," en, *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004, ISSN: 0920-5691. DOI: 10.1023/B:VISI.0000029664.99615.94. [Online]. Available: http://link.springer.com/10.1023/B:VISI.0000029664.99615.94 (visited on 06/19/2022).

[7] D. Garcia-Gasulla, A. Cortés, S. Alvarez-Napagao, and U. Cortés, *Signs for Ethical AI: A Route Towards Transparency*, Number: arXiv:2009.13871 arXiv:2009.13871 [cs], May 2022. [Online]. Available: http://arxiv.org/abs/2009.13871 (visited on 06/18/2022).

[8] B. Wilson, J. Hoffman, and J. Morgenstern, *Predictive Inequity in Object Detection*, Number: arXiv:1902.11097 [cs, stat], Feb. 2019. DOI: 10.48550/arXiv.1902.11097. [Online]. Available: http://arxiv.org/abs/1902.11097 (visited on 06/18/2022).

[9] J. A. Johnson, "Constructing Data Ethically," en, *New Directions for Institutional Research*, vol. 2019, no. 182, pp. 35–50, 2019, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/ir.20306, ISSN: 1536-075X. DOI: 10.1002/ir.20306. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/ir.20306 (visited on 06/18/2022).

[10] S. Tang, M. Andriluka, B. Andres, and B. Schiele, "Multiple People Tracking by Lifted Multicut and Person Re-identification," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, ISSN: 1063-6919, Jul. 2017. DOI: 10.1109/CVPR.2017.394.

[11] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple Online and Realtime Tracking," in *2016 IEEE International Conference on Image Processing (ICIP)*, arXiv:1602.00763 [cs], Sep. 2016, pp. 3464–3468. DOI: 10.1109/ICIP.2016.7533003. [Online]. Available: http://arxiv.org/abs/1602.00763 (visited on 06/18/2022).

[12] N. Wojke, A. Bewley, and D. Paulus, *Simple Online and Realtime Tracking with a Deep Association Metric*, Number: arXiv:1703.07402 arXiv:1703.07402 [cs], Mar. 2017. [Online]. Available: http://arxiv.org/abs/1703.07402 (visited on 06/18/2022).

[13] H. W. Kuhn, "The Hungarian method for the assignment problem," en, *Naval Research Logistics Quarterly*, vol. 2, no. 1-2, pp. 83–97, Mar. 1955, ISSN: 00281441, 19319193. DOI: 10.1002/nav.3800020109. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/nav.3800020109.

[14] A Bewley, *Sort17: Simple online and realtime tracking, mot challenge.* Accessed: 30. May 2022. [Online]. Available: https://motchallenge.net/method/MOT=2508&chl=10&vidSeq=MOT17-08-DPM.

[15] D. Stadler and J. Beyerer, "Modelling Ambiguous Assignments for Multi-Person Tracking in Crowds," in *2022 IEEE/CVF Winter Conference on Applications of Computer Vision Workshops (WACVW)*, ISSN: 2690-621X, Jan. 2022, pp. 133–142. DOI: 10.1109/WACVW54805.2022.00019.

[16] Y. Zhang, P. Sun, Y. Jiang, *et al.*, *ByteTrack: Multi-Object Tracking by Associating Every Detection Box*, Number: arXiv:2110.06864 arXiv:2110.06864 [cs], Apr. 2022. DOI: 10.48550/arXiv.2110.06864. [Online]. Available: http://arxiv.org/abs/2110.06864 (visited on 06/18/2022).

[17] A. Milan, L. Leal-Taixe, I. Reid, S. Roth, and K. Schindler, *MOT16: A Benchmark for Multi-Object Tracking*, Number: arXiv:1603.00831 arXiv:1603.00831 [cs], May 2016. [Online]. Available: http://arxiv.org/abs/1603.00831.

[18] *OpenCV: Camera Calibration*. [Online]. Available: https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html (visited on 06/07/2022).

[19] B. Kumar, *Image Preprocessing - Why is it necessary?* en, Feb. 2021. [Online]. Available: https://medium.com/spidernitt/image-preprocessing-why-is-it-necessary-8895b8b08c1d (visited on 06/07/2022).

[20] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," arXiv, Tech. Rep. arXiv:1506.02640, May 2016, arXiv:1506.02640 [cs] type: article. [Online]. Available: http://arxiv.org/abs/1506.02640 (visited on 06/07/2022).

[21] *Papers with Code - Argoverse-HD (Full-Stack, Test) Benchmark (Real-Time Object Detection)*, en. [Online]. Available: https://paperswithcode.com/sota/real-time-object-detection-on-argoverse-hd-5 (visited on 06/07/2022).

[22] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "YOLOX: Exceeding YOLO Series in 2021," arXiv, Tech. Rep. arXiv:2107.08430, Aug. 2021, arXiv:2107.08430 [cs] type: article. [Online]. Available: http://arxiv.org/abs/2107.08430 (visited on 06/07/2022).

[23] J. Cao, X. Weng, R. Khirodkar, J. Pang, and K. Kitani, "Observation-centric sort: Rethinking sort for robust multi-object tracking," *arXiv preprint arXiv:2203.14360*, 2022.

[24] JonathonLuiten, *TrackEval*, original-date: 2020-09-16T14:27:29Z, Jun. 2022. [Online]. Available: https://github.com/JonathonLuiten/TrackEval (visited on 06/19/2022).

[25] J. Luiten, A. Osep, P. Dendorfer, *et al.*, "Hota: A higher order metric for evaluating multi-object tracking," *International Journal of Computer Vision*, pp. 1–31, 2020.

[26] E. Ristani, F. Solera, R. S. Zou, R. Cucchiara, and C. Tomasi, *Performance Measures and a Data Set for Multi-Target, Multi-Camera Tracking*, Number: arXiv:1609.01775 arXiv:1609.01775 [cs], Sep. 2016. DOI: 10.48550/arXiv.1609.01775. [Online]. Available: http://arxiv.org/abs/1609.01775 (visited on 06/19/2022).

Appendix A: List of MoSCoW Requirements

Requirements

Must have:

- The system must calibrate the camera lens using the camera's calibration matrix.
- The system must preprocess the images to prepare them for the core workload.
- The system must store data about camera position and direction.
- The system must uniquely identify multiple objects throughout subsequent frames (MOT) using UUID.
- The system must map objects onto a 2d (bird's view) projection.
- The system must predict the velocity and direction of the movement of objects using available data.
- The system must remember the processed data (UUIDs, timestamps, location) within a session (whole camera operation time).
- The system must boast a local UI that allows users to control a system's state.
- The UI must display the bounding boxes and past processed data from the 2d view.
- The CI must easily perform automated benchmarks on the target device.

Should have:

- The end-user should be able to easily and securely retrieve session data for further evaluation.
- The UI should display the visual feed from any camera in the network (testing purposes only).
- The UI should display the processed data from the entire network.
- The UI should display a histogram of object density.
- The system should be optimized for the target embedded device.
- The system should use view masks (e.g. distance) to filter the input/output.
- The system should be adjustable based on object density and movement speed, and ambience conditions.
- The network should uniquely identify objects across different cameras with overlapping FoV (Field of View).
- The network should be able to communicate information about object position between cameras.

Could have:

- The UI could display data about the network health.
- The system could uniquely identify objects throughout the different cameras regardless of their relative position.
- The system could automatically provide metrics and visualize traffic density and flow.
- The system could automatically calibrate the bird's view transform projection (using e.g. depth estimation, edge detection heuristics).
- The system could automatically detect and adjust the image processing pipeline to environmental conditions.
- The network could implement an authentication system so that non-authorized users are not able to access the data.
- The network should be resilient to loss of one or more of the devices.

Won't have:

- The system won't have bounding boxes that limit the region of recognition.
- The system won't send camera frames to other devices.
- The automated benchmarks won't support multiple target devices.

Non-functional requirements

- The system's target embedded device is the Raspberry Pi 4 Compute Module.
- The system is written in Python 3 using TensorFlow.
- The system runs on Raspbian.
- The system saves processed data in a local MongoDB database.
- The system is GDPR-compliant.
- The benchmarking system runs the MOT17 benchmark on the target embedded device.
- The system should consume up to 9W.
- The network consists of uniform embedded devices only.
- Target performance depending on phase:

Metric	Must	Should	Could
FPS	2	5	10
Cameras	1	3	5

Bench (MOTA)	Must	Should	Could
MOT17 (15ppl)	45%	55%	65%

Appendix B: Work Distribution Table

Module	Task	Aleksandra	Filip	Natalia B	Paul	Natalia P
Setup	pipeline architecture		x	x		
	system deployment on the camera	x	x			
Preprocessing	calibration	x				
	histogram equalization			x		
Filtering	filtering				x	
Object Detection	YOLOv4	x				
	YOLOv5		x			
	YOLOX	x				
Object Tracking	SortMot					x
	MapTrack		x			x
Mapping	map generation		x	x		x
	2d transformation				x	
Communication	database		x	x		
	intercamera communication		x			x
Tracking Between Views	map reconstruction from multiple cameras	x	x			
	server + server database		x			
Benchmarking	benchmarking				x	
UI	implementation	x				x
	design				x	
Code Quality	code cleanup				x	
	integration testing			x		
	testing	x	x	x	x	x
Other	simulation tools		x			
	heatmap generation				x	
Research	database			x		
	object tracking algorithms		x			x
	object detection algorithms	x				
	intercamera communication					x
	tracking between views	x				x
	embedded optimization		x			

Figure 20: *Teble that represents work distribution among the team members.*

Appendix C: Branch Coverage Report

Coverage report: 83%

coverage.py v6.4, created at 2022-06-17 13:38 +0200

Module	*statements*	*missing*	*excluded*	*branches*	*partial*	*coverage*
src/communication/db_connectivity.py	44	3	0	24	0	93%
src/communication/publisher.py	33	8	0	2	1	74%
src/communication/subscriber.py	22	7	0	2	1	67%
src/detection/YOLOv4/yolov4.py	40	6	0	12	1	87%
src/detection/YOLOv5/yolov5.py	61	0	0	6	0	100%
src/detection/yolo_interface.py	8	0	0	2	0	100%
src/main.py	57	8	0	20	1	88%
src/mot/map_track.py	51	5	0	16	2	90%
src/mot/mot_interface.py	8	0	0	2	0	100%
src/mot/sort_mot.py	39	2	0	14	2	92%
src/pipeline/bg_filtering.py	84	22	0	30	4	75%
src/pipeline/handler.py	43	0	0	12	0	100%
src/pipeline/mapping.py	103	1	0	26	2	98%
src/pipeline/postprocess.py	33	1	0	14	1	96%
src/pipeline/preprocess.py	10	0	0	2	0	100%
src/pipeline/save_handler.py	26	7	0	8	1	71%
src/pipeline/transform.py	12	1	0	2	1	86%
src/setup/env/camera_setup.py	26	6	0	2	0	79%
src/setup/pipeline/lens_calibration.py	58	34	0	10	0	35%
src/setup/pipeline/pipeline_setup.py	82	16	0	44	7	75%
src/utils.py	23	15	0	4	0	37%
Total	**863**	**142**	**0**	**254**	**24**	**83%**

coverage.py v6.4, created at 2022-06-17 13:38 +0200

Figure 21: *Branch coverage table.*

Appendix D: List of Completed Requirements

Completed Requirements

Must have:

- The system must calibrate the camera lens using the camera's calibration matrix.
- The system must preprocess the images to prepare them for the core workload.
- The system must store data about camera position and direction.
- The system must uniquely identify multiple objects throughout subsequent frames (MOT) using UUID.
- The system must map objects onto a 2d (bird's view) projection.
- The system must predict the velocity and direction of the movement of objects using available data.
- The system must remember the processed data (UUIDs, timestamps, location) within a session (whole camera operation time).
- The system must boast a local UI that allows users to control a system's state.
- The UI must display the bounding boxes and past processed data from the 2d view.
- The CI must easily perform automated benchmarks on the target device.

Should have:

- The end-user should be able to easily and securely retrieve session data for further evaluation.
- The UI should display the visual feed from any camera in the network (testing purposes only).
- The system should be optimized for the target embedded device.
- The system should use view masks (e.g. distance) to filter the input/output.
- The network should be able to communicate information about object position between cameras.

Could have:

- The network should be resilient to lose of one or more of the devices.

Non-functional requirements

- The system's target embedded device is the Raspberry Pi 4 Compute Module.
- The system is written in Python 3 using TensorFlow.
- The system runs on Raspbian.
- The system saves processed data in a local MongoDB database.
- The system is GDPR-compliant.
- The benchmarking system runs the MOT17 benchmark on the target embedded device.
- The system should consume up to 9W.

Appendix E: Benchmarking Results

Sequence	No. of objects	Average object density	Frames count	Width	Height	FPS
2	62	31	600	640	360	30
4	83	45.3	1050	640	360	30
5	133	8.3	837	640	480	14
9	26	10.1	525	640	360	30
10	57	19.6	654	640	360	30
11	75	10.5	900	640	360	30
13	110	15.5	750	640	360	25

Table 2: Description of the MOT17 benchmark video sequences. Source: [17]

Sequence	Real time [s]	System time [s]	FPS	HOTA	MOTA
2	314	1025	1.91	16.29	9.39
4	555	1801	1.89	25.27	25.06
5	438	1429	1.91	37.52	45.38
9	275	895	1.91	35.44	48.45
10	342	1116	1.91	19.51	20.16
11	471	1541	1.91	37.31	48.25
13	388	1265	1.93	18.60	11.20
Combined	2783	9072	1.91	25.72	24.78

Table 3: Benchmark for YOLOv4-tiny + SORT

Sequence	Real time [s]	System time [s]	FPS	HOTA	MOTA
2	317	1025	1.89	17.44	9.28
4	553	1799	1.90	26.65	25.32
5	436	1430	1.92	39.53	44.12
9	276	896	1.90	35.33	48.17
10	344	1119	1.90	20.49	20.79
11	468	1533	1.92	40.36	47.71
13	390	1271	1.92	19.40	11.91
Combined	2784	9073	1.91	27.15	24.88

Table 4: Benchmark for YOLOv4-tiny + OCSORT

Sequence	Real time [s]	System time [s]	FPS	HOTA	MOTA
2	2311	8207	0.26	24.30	20.40
4	4047	14382	0.26	36.25	34.13
5	3232	11467	0.26	45.28	51.12
9	2016	7155	0.26	44.82	59.14
10	2508	8919	0.26	28.74	36.21
11	3456	12293	0.26	46.88	51.81
13	2874	10216	0.26	29.44	27.24
Combined	20444	72639	0.26	35.38	35.10

Table 5: Benchmark for YOLOv4-small + SORT

Sequence	Real time [s]	System time [s]	FPS	HOTA	MOTA
2	2348	8340	0.26	24.24	20.08
4	4060	14415	0.26	35.99	33.96
5	3231	11490	0.26	44.63	49.68
9	2023	7194	0.26	44.00	58.44
10	2534	8999	0.26	28.94	35.96
11	3474	12347	0.26	47.00	51.27
13	2888	10261	0.26	29.07	26.27
Combined	20558	73046	0.26	35.16	34.68

Table 6: Benchmark for YOLOv4-small + OCSORT

Sequence	Real time [s]	System time [s]	FPS	HOTA	MOTA
2	479	1764	1.25	23.60	18.06
4	832	2922	1.26	35.34	39.18
5	654	2333	1.28	39.77	41.72
9	420	1456	1.25	40.96	56.09
10	513	1829	1.27	26.66	31.30
11	704	2516	1.28	47.58	51.15
13	592	2087	1.27	28.04	21.63
Combined	4194	14907	1.27	34.03	34.93

Table 7: Benchmark for YOLOX-nano + SORT

Sequence	Real time [s]	System time [s]	FPS	HOTA	MOTA
2	480	1691	1.25	23.52	17.73
4	837	2976	1.25	35.51	38.99
5	663	2358	1.26	39.51	39.60
9	420	1464	1.25	40.69	55.57
10	518	1838	1.26	26.32	30.57
11	716	2548	1.26	47.56	50.28
13	595	2119	1.26	27.69	20.98
Combined	4229	14994	1.26	34.01	34.42

Table 8: Benchmark for YOLOX-nano + OCSORT