NTNU
Norwegian University of Science and Technology

Crowdstream

Displaying Video across the Phones of Large Crowds at Concerts

Lars-Kristian Dahl

Master of Science in Informatics
Submission date: June 2018
Supervisor: Mads Nygård, IDI
Co-supervisor: Ole-Ivar Holthe, Zedge

Norwegian University of Science and Technology
Department of Computer Science

Abstract

Light-emitting wristbands and other wearable devices have become a popular gimmick at concerts and large events in recent years. Usually, these devices can be remotely controlled through radio signals, allowing them to light up in response to stage events, at specific intervals, or in beat with music. However, few of these are spatially aware, which means that the emitted colors are often uniform or randomly distributed across the venue. Therefore, these devices are great for simple effects, but they can not be utilized for more advanced effects that require positional awareness and coordination. The few systems that are spatially aware either require elaborate set ups and provide very coarse positioning, or they require that the end-user inputs their exact location in terms of section, row and seat number. This significantly limits the use and applicability of these systems.

Zedge, a company specializing in mobile applications, wanted to investigate whether it was possible to utilize mobile phones to add positional awareness to such systems. This could in effect substitute the specialized bands and devices with existing mobile devices, and would allow for far more sophisticated light shows to be coordinated in large venues and crowds. They also wanted the solution to be cheap and easy to use, without requiring elaborate setup and specialized hardware.

This report proposes a new, novel approach for indoor positioning that is potentially better suited than existing approaches at providing relative positions in indoor venues. The project set out to investigate if it is possible to create a system that can be used to orchestrate a position-aware light show across the devices of an audience in a large indoor arena, using only an inexpensive, off-the-shelf camera and the devices themselves to determine their positions.

The resulting approach is cheaper, easier to set up, provides better relative accuracy and is less susceptible to signal interference than any other existing solution when used under specific conditions and assumptions. It is accompanied by a fully functional prototype of a system that can serve as a platform for further research and improvement. The prototype includes a server with a simple but extendable implementation of a new approach to Indoor Positioning Systems based on mobile devices and a camera, an Android client, and an administrator interface for managing the server.

Preliminary results indicate that the system should be capable of detection, positioning and streaming with at least 500 devices in an indoor venue. The report also provides ideas and suggestions for further research and improvements.

Abstract

Lysende armbånd og andre lignende enheter har blitt en populær gimmick på konserter og andre store eventer de siste årene. Slike enheter kan vanligvis kontrolleres ved hjelp av radiosignaler, som tillater dem å lyse i sammenheng med hendelser på scenen eller i takt med musikken. Det er imidlertid svært få enheter som benytter seg av posisjonering, noe som betyr at fargesammensetningen ofte er helt tilfeldig eller unfiform. Disse systemene fungerer derfor greit til enkle lysshow, men kan ikke brukes for mer avanserte effekter som krever koordinasjon og informasjon om enhetenes posisjon. De få systemene som benytter seg av slik informasjon krever enten avanserte og kostbare oppsett og tilbyr bare grov posisjonering, eller de krever at hver enkelt deltager manuelt fyller inn data som for eksempel seksjon, rekke og setenummer. Dette legger en kraftig begrensning for hvor og når systemet kan brukes.

Zedge, et firma som spesialiserer seg på mobile enheter, ønsket å undersøke om det var mulig å benytte deltagerenes mobiltelefoner til å posisjonere enhetene i slike systemer. Ett slikt produkt kan i mange tilfeller erstatte spesialiserte armbånd og andre enheter, og kan tillate mer sofistikerte lysshow som kan koordineres mellom enhetene basert på deres posisjon. De ønsket også at en eventuell løsning skulle være billig og lett å bruke, uten avanserte oppsett og spesialisert hardware.

Denne rapporten beskriver en ny fremgangsmåte for inndendørs posisjonering, som i noen tilfeller kan tilby bedre relativ posisjonering enn eksisterende løsninger. Målet for prosjektet var å undersøke hvorvidt det er mulig å lage ett system som kan brukes til å koordinere ett lysshow over flere enheter, ved å bruke ett relativt enkelt og kommersielt tilgjengelig kamera til å posisjonere enhetene.

Resultatet er en fremgangsmåte som er billigere, enklere å sette opp, og som er mindre påvirket signalstøy enn eksisterende løsninger, når den brukes under de rette forutsetninger. Som en del av resultatet foreligger det også en funksjonell prototype som kan brukes som en platform for videre utvikling. Systemet inkluderer implementasjon av en server som enkelt kan utvides og forbedres, samt en Android klient og en administrator klient for administrasjon av serveren.

Foreløpige resultater indikerer at systemet burde klare å oppdage, posisjonere og strømme video til 500 enheter i en innendørs arena, og rapporten bidrar også med spesifikke forslag og ideer til videre utvikling.

Preface

This thesis culminates my studies at the Norwegian University of Science and Technology. It has been written to fulfill the graduation requirements for the Master of Science in Informatics; Software programme, which was undertaken between August 2017 and June 2018.

The project was embarked upon at the request of Zedge, who put forth a very interesting concept and idea for a thesis. Their ideas and project has kept me motivated through 10 months of intense diligence and labor, which is now coming to an end.

I would like take this opportunity to utter my sincere appreciation and thanks to the staff of the Norwegian University of Science and Technology, and in particular to the Department of Computer Science (IDI), for their help through all of this, and especially this final year.

Further, I would like to express my earnest gratitude towards my supervisors; *Mads Nygård* (IDI), who always held me accountable for the progress and on the right track, and *Ole-Ivar Holthe* (Zedge), who provided invaluable input and constructive criticism. The project would not be possible were it not for their excellent guidance and support throughout.

I would also like to thank everyone that has assisted me during tests, experiments, and those who have provided invaluable comments on the thesis. They are, in no particular order; Mej-Jain Fung, Sondre Kvam and Julian Lam.

I also want to express my deep and sincere gratitude to my parents, for providing me with unfailing support and continuous encouragement throughout my years of study.

Last, but not least, my most profound and heartfelt thanks goes to my partner whom I hold most dear, *Anita Kildemo Flor*, for her never-ending belief, support and encouragement. This accomplishment would not have been possible without her.

Thank you.

Lars-Kristian Dahl   Trondheim, June 2018

Contents

List of Tables

List of Figures

1. Introduction

1.1. Background

Over the recent years, light-emitting wristbands and devices, such as Glow Motion and Xylobands, have become a favorite gimmick at concerts and other large events. Xylobands were first used during a Coldplay concert in 2012, [1], and has since been used by many artists and at several different events and venues with great success.

These devices can usually be controlled to emit light remotely through radio signals, either manually in response to stage events, at specific intervals, or in tune with the music. However, they are not spatially aware, resulting in the emitted colors being uniform or randomly distributed across the venue. As such, these devices are great for simple effects, but they can not be utilized for more advanced effects that require positional awareness and coordination.

Zedge, a company specializing in mobile applications, hereafter referred to as the client, wanted to research whether it was possible to utilize mobile phones to add positional awareness to such systems. This could in effect substitute the specialized bands and devices with existing mobile devices, and would allow for far more sophisticated light shows to be coordinated in large venues and crowds. The client also wanted to investigate if it was possible to develop a system that is cheap and easy to use, without requiring elaborate setup and specialized hardware. Displaying and coordinating colors across mobile devices is not a difficult task; however, Indoor Positioning System (IPS) is a very complex and challenging problem.

Positioning systems are becoming increasingly important, and research in this area is being fueled by increasing demand and interest from academia, industry and government agencies. Outdoor positioning systems have seen much progress, and the Global Positioning System (GPS) have had particularly great success. IPS however, are still in the early stages of development.

A large variety of IPS technologies exist, such as Infrared, Ultrasound, WiFi, Radio Frequency Identification (RFID), Bluetooth (BT), Ultra Wide Band (UWB), and others

[1]https://www.geek.com/gadgets/xylobands-turn-coldplay-concerts-into-giant-light-show-1496831/

[1],[77], but there is no single solution that works well under all circumstances. Throughout the last decade, several new technologies and methods have also emerged, such as RADAR by Microsoft Research Asia [45], HORUS by University of Maryland [79], LiFS as proposed by Tsinghua University [69] and the ZigBee specification [49].

Nevertheless, many challenges remain for IPS, such as transmitter-receiver synchronization errors, multi-path interference effects and a need for high sampling rates [1]. Therefore, the best general systems often combine several different technologies and methods, making the solutions expensive and complicated.

Fingerprint-positioning based on WiFi is among the most popular and widespread technologies for 2D modeling, it is relatively cheap, and can provide excellent accuracy and performance under the right circumstances. It is likely the most suitable existing option for indoor positioning in large crowds, and it is a well-researched technology. However, it has several significant drawbacks as described below, many of which are shared with most of the other existing IPS solutions.

WiFi fingerprint positioning requires a broad set of algorithms to support it, and so computation and algorithm complexity is relatively high. The method also requires a tremendous amount of work up front for data support, so the preliminary work has a high-cost factor [70]. However, most significantly, this method, along with most other existing IPS solutions, is highly susceptible to interference from the human body [21]. Other significant issues include the multi-path effect [62] and variations in the software and hardware of the devices to be positioned [35]. These factors significantly reduce the applicability of these technologies in crowded areas such as concert arenas, which is precisely where the system being developed is to be used.

Zedge proposed a solution based on a stage-mounted camera, that would detect devices in the crowd and find their relative position. They did not specify exactly how the solution would detect the devices and their positions but wanted to know if such a solution was at all feasible and worth investigating.

This research aims to develop a new method for indoor positioning that is better suited than existing approaches for providing relative positions in indoor venues. The client has a hypothesis that a system such as the one described above is viable, but any outcome from this research, positive or negative, is valuable to them. It is evident that a positive result provides more value for the client than a negative result, but either outcome provides some value. If successful, the approach could provide a method that is cheaper, easier to set up, provides better relative accuracy and is less susceptible to interference than any other existing solution when used under specific conditions and assumptions.

1.2. Objectives

The objective is to create a system that can be used to project a light-show across the devices of large crowds at indoor events, using a stage-mounted camera for indoor positioning. The specific scenario is defined as follows:

Indoor concert hall or arena The system is only intended for indoor use, where GPS is not available.

Low-light environment The system is intended for low-light or no light conditions, where the light can be controlled by the event organizer.

100 to 2000 participants It will be used for small to medium-sized events.

Stage-mounted camera The organizer must be able to mount a camera near the top of the stage, such that it can capture the whole audience.

Pre-installed apps The participants are required to have the client application installed on their devices.

One-time positioning One-time positioning is adequate, and the ability to continuously track device positions is not required.

Active participants The audience is assumed to be active and willing to participate in a setup procedure over a short duration, announced over speakers.

The general idea was initially pitched by Zedge and has since been more precisely defined and refined in cooperation with them. The research problem then led to a hypothesis, that such a system was feasible, and several related Research Questions (RQs).

- RQ-1: Is it possible to locate devices at an indoor arena using a stage-mounted camera, for the purpose of displaying light-shows across the devices?
- RQ-1.1: Can a camera and a unique discrete-time light signal sent from each device be utilized to identify the position of the devices under these conditions?

- RQ-1.2: How does the camera hardware specifications affect the performance of the system, and what are the minimum requirements for the camera?
- RQ-2: Which methods and techniques are most appropriate for object detection and object tracking under these conditions?

RQ-1 is a result of said hypothesis, and it has also led to some closely related sub-questions. RQ-1.1 and RQ-1.2 have originated from my own experience and lack of relevant research on the topics they are concerned with. They have been defined through dialogue and cooperation with the client.

RQ-2 spurred from a literature review on current research on object tracking, which revealed that there is a significant amount of different methods and techniques for general purpose object tracking. I was, however, unable to find any conclusive research on suitable methods for the specific conditions this system will operate under, such as; low-light, active markers that can disappear and reappear and movement that is mostly restricted to arm-waving.

1.3. Approach

The research approach is a combination of experiments and traditional literature reviews. Some of the most likely and possible challenges have been discovered and defined during the preliminary research, while many challenges will likely be discovered during the development of the prototype, and solved using a combination of existing literature and new development. The approach for each research question is presented in Table 1.1.

The research strategy, data generation and data analysis for RQ-1 have been defined in cooperation with the client and involves the design and implementation and evaluation of a prototype of the system. For RQ-1.1 and RQ-1.2 I have selected to do software simulation and experiments to generate data, and use a quantitative evaluation.

RQ-2 will also use an experimental research strategy, where data will be generated through software simulation and testing and evaluated empirically based on quantitative performance measures, such as error-rates. Conclusions will be drawn based on the empirical quantitative analysis for RQ-1.1, RQ-1.2, and RQ-2, while the conclusion for RQ-1 will be based on qualitative measurements, including a small-scale real-world test and feedback from the client.

The overlying goal of the research is to determine whether a hypothesis, that the approached suggested by the client is feasible for their stated purpose, is correct. RQ-1 is subjective, the data analysis is qualitative, and my own beliefs will affect the analysis

RQ #	Source	Strategy	Data generation	Data analysis
RQ-1	client	design and implementation	client feedback, real world test	qualitative; client satisfaction
RQ-1.1	experience, dialogue with client	experiment	software, simulated test	quantitative; system performance measures
RQ-1.2	experience, dialogue with client	experiment	software, simulated test	quantitative; system performance measures
RQ-2	literature review, object tracking	experiment	software, simulated test	quantitative; object tracking performance measures

Table 1.1.: Research methodology

of the research, which suggests an interpretivist paradigm. However, the sub research questions are mostly based on experimental empirical testing with quantitative data analysis, where my own beliefs as a researcher are distinctly separate from the scientific results. This makes the sub-questions more in line with the positivist paradigm. RQ-2 share many of the traits with the sub-questions of RQ-1, and therefore also lends itself to the positivist paradigm.

1.4. Similar work

Throughout the research and work with this project, there has not been found any existing work that offers the same features as the Crowdstream system. Products do exist that offer random light shows through specialized hardware or smartphones, and a select few that offers location-aware light shows which require the user to submit their position manually.

1.4.1. Xylobands

Xylobands[2] was the first audience-based light show technology to gain any real traction. It was first used by Coldplay on their Mylo Xyloto tour in 2012 and has since been used at many large events and venues. The wristbands themselves contain light-emitting diodes and radio receivers. The bands can be controlled remotely through radio signals, instructing them to light up or blink. There exists an RGB version which can emit any color on the RGB spectrum and a single color version which only emits a single color. The bands are not location-aware but have been a great success nonetheless.

[2]http://www.xylobands.com

1.4.2. Glow Motion

Glow Motion[3] is very similar to Xylobands, but they also offer many other types of products besides wristbands. Examples of other types are large beach balls, medallions, and costumes. The technology in all products are the same, radio-controlled LED's, and none of them are location-aware.

1.4.3. Cue Audio (XTAudioBeacons)

Cue Audio[4], formerly known as XTAudioBeacons, is a data-over-audio method that uses sound waves in a similar way to how Bluetooth employs electromagnetic waves. This is a technology that has many use-cases, but it is primarily being marketed to events and venues as a means to offer synchronized light shows. Like Crowdstream, it requires users to install a separate application on their smartphone, and for it to be running during the event. Unlike Crowdstream it does not require an active internet connection during the event, because signals and data are transmitted to the application through ultrasonic waves from the venue's speakers, which are detected by the microphone of the device. This does, however, make it impossible to send data to specific devices, as all data are effectively broadcast to all devices in range of a given speaker. It is also not possible to send data from the device to the system, such as the GPS location.

Much like Bluetooth beacons, this system does support coarse-grained indoor position sensing by relying on the proximity to a given beacon, or speaker. This allows the system to divide a venue into sections, each within audible range of a specific beacon or speaker. It does, however, provide very rudimentary control, and devices could potentially be included in many sections or none at all, depending on their position. Still, the system has had some commercial success and has recently gone from an open-source project to a commercial product.

1.4.4. Dan Deacon App

The Dan Deacon App[5] is an application that emits light based on sound signals. In that sense, it is similar to Cue Audio, except it relies on audible sound instead of ultrasound. However, unlike Cue Audio which offers a general protocol for transmitting data, the Dan Deacon App merely reacts to the frequency and volume of the sound by emitting pre-configured light. An example could be that it flashes in sync with the bass of a

[3]https://www.glowmotiontechnologies.com
[4]www.cueaudio.com
[5]https://itunes.apple.com/us/app/dan-deacon/id536378735?mt=8

particular track, or emits a red and green light during the calm parts of a song. This solution is not location-aware.

1.4.5. Card Stunts Smartphone Light Show

This solution by Cards Stunts[6] is the only existing solution found capable of orchestrating high-precision location-aware light shows. This system is the solution that closest resembles the Crowdstream system, and they are almost identical in the following ways:

- Requires an application to be installed on the participants' device.
- Requires an active internet connection, either WiFi or Mobile Data.
- Can be remote controlled and used to stream almost any content.
- Can send and receive data to/from individual devices.
- Can be used to create sophisticated, location-aware light shows.

However there is one significant difference which sets the systems apart; Card Stunts solution requires each participant to manually submit their position in the venue, identified by a section, row, and seat. This imposes a significant restriction on the practical use of the system, as it can only be used in venues that have numbered seats.

1.5. Contributions

This research set out to investigate if it is possible to create a system that can be used to orchestrate a position-aware light show across the devices of an audience in a large indoor arena, using only an inexpensive, off-the-shelf camera and the devices themselves to determine their positions. In this report I propose a novel method for IPS that provides high relative precision between devices and is less susceptible to interference in very crowded indoor arenas than many existing solutions. I also provide a fully functional prototype of a system that can serve as a platform for further research and improvement. The prototype includes a server with a simple but extendable implementation of a new approach to IPS based on mobile devices and a camera, an Android client, and a Command Line Interface (CLI) for managing the server.

The pre-requisites and assumptions required make the system less useful for general

[6] http://www.cardstunts.com/smart-phone-light-shows/

indoor positioning problems, but it is still a valuable contribution for the client and hopefully also for others. This is described in more detail in section 1.6.

1.6. Limitations

This section describes some of the most significant limitations of the project, related to the study itself and the operational conditions of the resulting solution.

1.6.1. Research limitations

The research for a solution has some limitations. Some were imposed by the client; others are due to the nature of the project, all of which are discussed in the following sections.

Hardware

The objective was to find a solution using only a camera and the participants' devices, which eliminates many other possible solutions based on, e.g., Bluetooth, Ultrasound or Augmented Reality. It might seem redundant to state this as a limitation since it is already defined in the objective, but it is important to note that there might exist other solutions that can achieve the same result using other technologies.

Time

Due to the nature and circumstances of a master thesis project, it has a set duration and deadline. This imposes hard restrictions on how much time can be spent researching and implementing a solution, which is roughly 8 months in total. This has resulted in an attempt to strike a balance between the time spent on research, and the time spent on implementing a prototype. It was important to get this balance right, as too much focus on research could result in an unfinished prototype, which is an important part of the project for the client. However, as it is a master thesis, it is also important that the practical work is based on a good foundation of research and theoretical work.

Cost

The client wanted an inexpensive solution, which places some restrictions on the project and especially on the prototype. This restriction has the most impact on the type of camera used, which in the prototype is a commercially available middle-range web camera for desktop computers. Such cameras have severe restrictions on the ability to manually control the camera settings, such as shutter speed, iso-sensitivity, and aperture. The performance of the camera in terms of Frames Per Second (FPS) is also significantly impacted by light conditions, as the normal operating conditions for such cameras are significantly different from the operating conditions for the Crowdstream system. Some issues described in this thesis could likely be reduced or eliminated using better camera hardware.

Real-world tests

Due to time, cost and practical limitations, it has not been possible to perform a large-scale real-world test of the system. This requires cooperation with an event organizer and a venue, which was not possible to organize within the time frame of the project.

1.6.2. Operational conditions

There are several significant limitations that are imposed by the system and methods developed throughout this project. These limitations are a result of a couple of strong assumptions regarding the environment the system will operate in, as well as the assumption that the audience is willing to participate actively during parts of the positioning process. These assumptions were discussed and defined with the client beforehand and were deemed both reasonable and acceptable for their intended usage. However, the assumptions will greatly limit the scope of the system, and its use is limited to a particular scenario and environment.

Environment

The system will only be effective in dark environments where the glow from devices in the audience can easily be distinguished from the background. This greatly simplifies the object detection process and makes it possible to track devices during the positioning process reliably.

Camera motion and position

The system relies on a stationary camera, and can not compensate for camera motion during the positioning process. The camera should also be placed high above the audience, to minimize the chance of devices being occluded during the positioning process.

Relative position

The system does not measure absolute positions. For the intended purpose, the system only needs to know how the devices in the audience are positioned relative to each other. Most other uses of IPS relies on absolute positioning.

Audience participation

The audience needs to actively participate both before and during the positioning process. They need to install and start an application on their device beforehand, as well as connect to the system server. During the positioning phase, they need to actively participate by holding their device up towards a stage-mounted camera, with the device screen facing the camera.

One-shot positioning

The system will perform a time-limited positioning process, and can not track the positions of devices as they are moved to different locations after this process has finished. The positioning process can, however, be repeated an unlimited number of times.

1.7. Outline

The research focuses on the problems related to object detection, multiple object tracking, and indoor positioning. These problems are most likely the problems that are hardest to solve, and the success of the prototype will ultimately depend on how well the system can detect, track and position the devices in the audience. The other significant components of the system, such as client/server communication and streaming of data already have well established and working solutions that can be built on.

The thesis is outlined as follows:

Chapter 2 provides an overview of the current research and State of the Art (SOTA) on image segmentation and object detection for images. There exists a large body of research on the topic, and this chapter covers and summarizes the most successful and relevant areas of research.

Chapter 3 offers an introduction and overview of the current research and SOTA on multiple object tracking. As for detection, there already exists a large body of research on the topic. The field of object tracking, and in particular multiple object tracking, is still heavily researched and new contributions in the field are frequent.

Chapter 4 discusses some possible methods and approaches for how a camera, a computer and mobile devices can be utilized to position said devices in an indoor environment. It also discusses the most relevant methods and algorithms described in chapter 2 and chapter 3 in context of the Crowdstream system, and how they can be utilized to aid in device positioning.

Chapter 5 gives an overview of the prototype system, followed by a detailed description of the architecture, design and finally implementation based on the discussion in chapter 4. The system consists of several different components and clients, and each is covered individually.

Chapter 6 describes a set of tests and experiments that have been conducted throughout the development of the methods and prototype. It also presents the results of these tests.

Next, chapter 7 provides a summary, the final results and some conclusions that can be drawn from the research. In addition, it includes an evaluation and discussion of the project in a retrospective manner and discusses some of the most exciting discoveries.

Finally, chapter 8 concludes the thesis with a summary of the current research and state of the prototype. It also provides suggestions and ideas for future research and development of the system.

Appendix A is a test plan for the real world tests and experiments that were conducted during the project.

Appendix B contains the test results from the real world tests.

Appendix C describes how the system can be installed and deployed, and also how it can be set up for further development.

Appendix D contains a detailed description of all system parameters and their recommended values.

2. Object Detection

Object detection and object recognition is the act of detecting and recognizing semantic objects in digital images and videos. It is a well-researched domain and has many applications in areas of computer vision, including image retrieval, medical imaging, face recognition and video surveillance. Object detection and object recognition is however still a challenge for computer vision systems, and several approaches have been developed over multiple decades.

Object detection is an essential component of the Crowdstream system because the performance of most object tracking methods have a strong correlation with the performance of the object detection component [37]. And object tracking is an integral part of the system because it dictates how well it can recognize and position participant devices, which is a core aspect of the system.

Several object detection approaches have been developed, often combining statistical analysis of visual features with temporal analysis of the motion features. This chapter contains a basic introduction of the basic concepts and recent advancements in object detection.

Section 2.1 provides a short introduction to image segmentation, an important aspect of almost any object detection method. Next, section 2.2 and section 2.3 provides a description of common segmentation techniques based on visual and temporal information respectively. After that, in section 2.4 follows an overview of current state-of-the-art methods, which usually utilizes a combination of multiple methods and algorithms to provide state-of-the-art feature detection and feature extraction, which further can be used for object detection. Finally, section 2.5 contains a brief summary of the chapter.

2.1. Image segmentation

Image segmentation is the process of partitioning or dividing an image into multiple segments, in order to create a representation of an image that is easier to analyze [59]. The input is commonly the raw image data, and the output is a simplified representation of the image, divided into distinct regions. The pixels in a region are similar with regard to some defined characteristic, such as color, intensity or texture. Adjacent regions are typically significantly different with respect to the same characteristic [59].

A common use of image segmentation is to detect objects or edges in images, and these regions can then be used for further analysis and processing. Practical applications of image segmentation includes content-based image retrieval, machine vision, medical imaging [48] [17], object detection [13] (pedestrian detection, face detection) and video surveillance.

A video has primarily two sources of information that can be used for object detection:

- **Visual features:** color, texture, shape
- **Motion features:** position, movement, velocity, direction

Several different methods have been devised to utilize this information, some which use a single source of information while others use a combination. Some of these methods will be described further in the following sections.

2.2. Visual analysis

Visual segmentation methods rely solely on the visual information in an image, such as color, texture or shape, and can be utilized for single images as well as sequences of images such as videos. Visual segmentation algorithms are often based on one of the following two basic methods for partitioning:

- **Similarity** Partitioning an image into regions that are similar according to a set of predefined criteria
- **Discontinuity** Detecting boundaries of regions based on local discontinuity in intensity

Many general-purpose algorithms and techniques have been developed for single image segmentation, and they can be divided into several different categories [22]. Some of the most commonly used categories are[60]:

- Threshold-based methods
- Edge-based methods
- Region-based methods

- Clustering-based methods
- Match-based methods
- Neural Network (NN)- and Genetic Algorithm (GA)-based methods

These methods will be described in more detail in the following sections.

2.2.1. Thresholding (similarity)

Threshold segmentation is a straightforward method and can be used to create a binary image out of a grayscale image [59]. The most straightforward thresholding methods classify each pixel as background if the image intensity $I_{i,j}$ is less than some fixed constant T (that is, $I_{i,j} < T$), or foreground (object) if the image intensity is higher than that constant. Threshold segmentation can be divided into three different thresholding methods, based on how the threshold value is used:

Global threshold When using global thresholding, the image is divided into foreground and background using a single predefined threshold across the entire image [12]

Optimal global threshold Optimal global thresholding uses an algorithm to determine the threshold value, in an attempt to select a threshold value that optimally segments the image into foreground and background regions.

Local threshold A local threshold method divides the image into multiple non-overlapping sections, which are then assigned threshold values separately, either manually or based on some algorithm.

An important aspect of this method is to select the threshold value T, either manually or automatically, and several algorithms exist. These methods can be divided into six groups based on the information the algorithm utilizes [58]:

- Histogram-based methods
- Clustering-based methods
- Entropy-based methods

- Object Attribute-based methods
- Spatial methods
- Local methods

One of the most common methods is *Otsu's method* [58], which finds a globally optimal threshold by maximizing the histogram variance between the foreground and background.

Other popular methods commonly used in the industry are the *Maximum Entropy Method* [82], *Balanced Histogram Thresholding* [15], *Iterative Selection Thresholding Method* [51] and *K-Means Clustering.*

The primary advantage of thresholding as an image segmentation technique is its computational efficiency and low complexity. It is particularly useful for high-contrast images, where the foreground pixels can be easily separated from the background pixels based on the intensity histogram. The drawback is that it can be difficult to obtain good results when the foreground and background cannot be distinguished based on the intensity histogram alone, such as when there is a significant overlap in the grayscale values of the image.

Because it only considers the gray scale information of the image it is very susceptible to noise, and it is often necessary to combine it with other methods to achieve accurate results. It is still a popular method due to its low computational cost and complexity, and it can effectively be combined with methods that utilize spatial and motion information [46].

Most algorithms assume that images are multimodal, and these algorithms will often fail in the case of unimodal images. A few algorithms have been developed specifically to handle such images. Examples of unimodal image threshold selection algorithms are

T-Point Algorithm The tail of the histogram is fitted by two line segments, and the threshold is selected at their intersection [10].

Maximum deviation A straight line is drawn from the histogram peak to the end of the tail, and the threshold is selected at the point of the histogram further from the straight line [53].

Rayleigh distribution model This algorithm assumes that the peak of the histogram is noise, and lets the user specify an allowed proportion of noise from which the threshold is determined [68].

2.2.2. Edge-based segmentation (discontinuity)

Edge-based segmentation works by detecting edges and then dividing the image into regions based on the detected edges. Adjacent regions with different intensity values are separated by a discontinuity in intensity, and this discontinuity can often be detected using derivative operations, which can be calculated by using differential operators [57].

Several popular methods exist, and some of the most common ones are *Canny Edge Detection* [9], *Deriche-Canny Edge Detection*, Laplace of Gaussian (LOG), *Robert's cross detection*, *Prewitt detection*, *Marr-Hildreth edge detection*, *Shen-Castan edge detection* and *Sobel detection* [39]. Table 2.1 provides a comparison of the advantages and disadvantages of some of these methods.

For general purpose edge detection, *Canny Edge Detection* and it's derivatives generally outperform the other alternatives. This is mostly due to being more robust to noise but comes at the cost of increased computational cost [39], and other methods might provide better performance for specific types of images.

Operator	**Advantages**	**Disadvantages**
Classical (sobel, prewitt, kirsch)	simple, edge orientation	sensitive to noise, inaccurate
Zero Crossing (Laplacian)	edge orientation, fixed characteristics	sensitive to noise, double detection
LOG, Marr-Hildreth	edge position, few double detections	issues with corners and curves no edge orientation
Gaussian(Canny, Shen-Castan)	insensitive to noise, accurate	complex, computationally expensive, time consuming

Table 2.1.: Comparison of common edge detection methods

2.2.3. Region-based segmentation (similarity)

Region-based approaches are prevalent image segmentation methods, and the basic idea is to partition images into regions by grouping neighboring pixels of similar intensity [4].

Common approaches are *region splitting-and-merging* (top-down approach) and *region growing* (bottom-up approach). *Region merging* works by selecting a seed pixel and then merging nearby similar pixels into the region where the seed pixel is located.

Advantages of region-based methods are that they can separate the connected regions with similar characteristics accurately, and can provide boundary information. The idea is also relatively simple.

Some significant issues with this approach are that it can be difficult to determine good starting points (seed pixels), it is hard to automate, and it needs proper criteria for similarity [22]. It also has a significant computational cost [3], is susceptible to noise and does not handle shadows well [20].

2.2.4. Clustering-based segmentation (similarity)

Clustering-based methods perform the image segmentation by assigning pixels to separate, disjoint groups or clusters, based on their similarity with other pixels in the assigned cluster, and the dissimilarity with pixels in the other clusters.

One of the most popular methods for image segmentation based on clustering is K-means Clustering.

K Means Clustering and derivatives K-means is one of the simplest and best-known algorithms for solving clustering problems in general. It utilized unsupervised learning to minimize an objective function, which for image segmentation purposes is usually a squared error function. There are several derivatives and improvements of the k-means algorithm.

Fuzzy C-means Clustering and its improved version are also popular approaches.

2.2.5. Match-based segmentation (similarity)

Match-based segmentation techniques compare a template image to the input image in order to find images with a certain intensity distribution or shape and are best suited for applications where the intensity and configuration of the objects to be detected is roughly known in advance.

2.2.6. Neutral network and genetic algorithm based methods

Deep learning has had some significant breakthroughs and advances in recent years, and this includes the field of computer vision. Approaches based on Deep Convolutional Neural Network (DNCC) using weakly- and semi-supervised learning has shown excellent results, providing state-of-the-art performance on several challenging object recognition datasets, and continue to improve continuously. However, these methods are often complex and introduce a significant computational cost.

2.3. Motion analysis

Segmentation and detection of moving objects in videos or sequences of images, also known as background subtraction or foreground detection, is an important aspect of many modern multimedia applications. It is particularly important in the image processing field, and a lot of its research has been driven by the so-called second-generation coding techniques [31]. It is also important in other areas such as video surveillance, general object recognition, the medical field, and navigation.

Unlike visual analysis, motion analysis can utilize information from multiple images in a sequence to perform the segmentation. Motion analysis and the related segmentation methods generally work best with fast moving objects, and accurately segmenting sequences with slow-moving objects, and a dynamic background is a challenging task.

An efficient way to obtain foreground objects is background modeling, and a large number of background modeling methods have been proposed over the last few decades. And even though background modeling methods for background subtraction have been studied for several decades, no single optimal solution exists, and each method has it's own strengths and weaknesses [74].

The proposed methods generally share a common scheme:

1. Build a background model based on the first or previous frames

2. Compare the current frame with the background model to detect moving objects

3. Update the background model

The various background modelling methods can be categorized into *region-based* methods, *boundary-* or *pixel*-based methods, and a combination of the two [74].

2.3.1. Boundary-based methods

Boundary- or pixel-based methods utilize the information carried in each pixel separately, and assign them to regions such that pixels in the same region share certain characteristics with other pixels in the same region, and can as such detect the boundaries or edges of moving objects. Boundary-based methods are susceptible to noise but can produce accurate representations of the detected objects.

Examples of popular boundary-based methods are Gaussian Mixture Model (GMM) [63], Adaptive Gaussian Mixture Model (AGMM) and Pixel Based Adaptive Segmenter (PBAS) [24].

2.3.2. Region-based methods

Unlike boundary-based methods, region-based methods utilize the relations between pixels to segment images into regions and perform the background subtraction. Region-based methods are also less susceptible to input noise, but they can only provide rough outlines of the detected foreground objects.

Example of region-based methods have been published by Elgammal et al. [16], Liu [34] and Russel [55].

2.3.3. Hybrid methods

Hybrid methods, which use a combination of region-based and boundary-based methods, can often provide a better background model and handle changes to the illumination as well as dynamic backgrounds [44]. Even though such hybrid approaches can accurately segment an image into foreground and background regions, their computational cost and complexity are relatively high. Examples of hybrid approaches have been proposed by Cristani [11] and Toyoma [66].

2.4. Feature descriptors

Feature detection, extraction, and description is an active area of research and is one of the most studied topics in computer vision literature. Feature detection refers to the process of identifying points or regions in an image that can be used to describe an image's contents. There is no universal definition of what constitutes a feature, and it often depends on the problem domain, but typical features are edges, corners, ridges, and blobs. Table 2.2 provides an overview of common feature detection methods.

Category	**Classification**	**Methods and Algorithms**
Corner-based	Differential based	Harris, KLT, Shi-Tomasi, LOCOCO, S-LOCOCO
Corner-based	Gradient based	FAST, AGAST, BRIEF, SUSAN, FAST-ER
Corner-based	Template based	ANDD, DoG-curve, ACJ, Hyperbola, fitting
Corner-based	Contour based	NMX, BEL, Pb, MS-Pb, gPb, SCG, SE, tPb, DSC, Sketch Tokens
Blob (interest point)	PDE based	SIFT, SURF, CenSurE, LoG, DoG, DoH, Hessian, RLOG, MO-GP, DART,KAZE, A-KAZE, WADE
Blob (key point)	Template based	ORB, BRISK, FREAK
Blob (interest region)	Segmentation based	MSER, IBR, Salient Regions, EBR, Beta-Stable, MFD, FLOG, BPLR

Table 2.2.: A summary of the current state-of-the-art feature detectors [33].

Once a feature has been detected, the feature can be extracted to a feature descriptor or feature vector, which often involves a considerable amount of complexity and image processing. This process, including detection, extraction and the resulting descriptor, is known as feature description.

There exists a large number of feature description methods, and they vary widely in the kinds of features they detect, extract and describe, as well as the computational complexity and their applicability. Table 2.3 shows a comparison of the performance of the most popular state-of-the-art feature detection methods. There exists no single optimal feature detection method today, mainly due to the virtually infinite number of possible applications [56]. Examples of the variance of applications can be detecting one or multiple features, differences in image conditions (scale, viewpoint, illumination, contrast, image quality, etc.) and scenes (indoor, outdoors).

Most of the proposed state-of-the-art algorithms require intensive computations, and many rely on specialized hardware and processing capabilities to achieve acceptable computational performance. This limits their use for online applications and also to some degree for offline applications depending on the intended use.

Several feature detection algorithms, *Scale-Invariant Feature Transform (SIFT)*, *Speeded Up Robust Features (SURF)* and *Maximally Stable Extremal Region (MSER)* in particular, have shown excellent performance in surveys in the computer vision literature, e.g. [67]. These, as well as a few other popular methods, will be described in more detail in the following sections.

Feature detector	**Invariance**			**Qualities**			
	Rotation	Scale	Affine	Repeatability	Localization	Robustness	Efficiency
Harris	✓			+++	+++	+++	++
Hessian	✓			++	++	++	+
SUSAN	✓			++	++	++	+++
Harris-Laplace	✓	✓		+++	+++	++	+
Hessian-Laplace	✓	✓		+++	+++	+++	+
DoG	✓	✓		++	++	++	++
Salient Regions	✓	✓	✓	+	+	++	+
SURF	✓	✓		++	+++	++	+++
SIFT	✓	✓		++	+++	+++	++
MSER	✓	✓	✓	+++	+++	++	+++

Table 2.3.: Overview of the performance of the dominant feature detection algorithms [67].

2.4.1. Scale-Invariant Feature Transform

The SIFT algorithm is a Partial Differential Equation (PDE) based blob detector, which uses a local feature detector and a local histogram-based descriptor. It detects sets of interest points in an image and computes a histogram-based descriptor for each interest point [56]. It can identify objects in noisy images, and under partial occlusions, because the SIFT algorithm is invariant to changes in scaling, orientation, and illumination, as can be seen in table 2.3. A large number of improvements and extensions have been proposed to enhance and tailor SIFT to specific uses.

2.4.2. Speeded Up Robust Features

SURF is a very popular, patented local feature detector and descriptor, that is in part inspired by SIFT. Surveys have shown SURF to be faster than SIFT terms of speed, and it is less sensitive to illumination changes. However, SURF is more sensitive to variations in rotation, scale, and affine transformations, although both methods generally display poor performance in regards to affine transformations in general [27], as shown in table 2.3. Like SIFT, a large number of derivatives and improvements have been suggested.

2.4.3. Features from Accelerated Segment Test

Unlike SIFT and SURF, which utilizes blob-detection 2.2, Features from Accelerated Segment Test (FAST) [54] is a feature detector that uses corner-detection, specifically gradient-based corner detection. Its most prominent feature is its computational efficiency, which makes it very suitable for online (real-time) processing. It does, however, suffer in terms of accuracy and robustness compared to detectors such as SIFT and SURF, but for many applications that is an acceptable trade-off considering the major improvements it offers in terms of speed and computational complexity. This algorithm also has several derivatives and improvements, such as FAST-ER.

2.4.4. Maximally Stable Extremal Region

As can be seen in table 2.2, MSER is a method for blob-detection. It is a technique proposed by Matas et al. [40] to find corresponding elements in images taken from different viewpoints, e.g., by cameras in a stereo configuration, and is considered to be best of its kind. Several improvements have been proposed, including support for color images [18] and increased robustness to blur and scale changes [19].

2.5. Summary

Computer vision is one of the most active research fields in information technology today.

Image segmentation and filtering has been studied for a long time and has a large body of established research. It is usually part of the initial steps in computer vision systems, and the system performance is often directly related to the performance and accuracy of the pre-processing and image segmentation. The performance of the Crowdstream system will be highly dependent on the performance of the image segmentation and its ability to correctly detect objects.

Feature detection and description algorithms are fundamental in modern applications of computer vision. However, these algorithms are typically computationally expensive and complicated, which often prevents them from achieving the speed needed for real-time applications.

Object detection and recognition is an area that has undergone a lot of research, and it is still actively researched.

3. Object Tracking

Multiple Object Tracking (MOT) is an important computer vision problem which has gained increasing attention due to its academic and commercial potential. Although different kinds of approaches have been proposed to tackle this problem, it still remains challenging due to factors like abrupt appearance changes and severe object occlusions [37].

Object tracking is an essential aspect of Crowdstream because it lays the foundation for how well the server can correctly position devices in the crowd, which is essential for the function of the system.

The task of MOT mainly consists of attempting to detect and identify multiple objects in individual frames in a sequence and recover the identity information of the objects in subsequent frames. MOT tracking can be decomposed into two separate steps that address independent issues [5]. The first is time independent detection, as described in chapter 2. The second step relies on modeling detection errors and target motions to associate object detection to the most likely object trajectories.

Some examples of objects that can be tracked are vehicles [29], pedestrians[47], sport players [73] or groups of animals (birds [36], etc.).

MOT is a challenging task, especially in complex scenes where objects are occluded by background or other objects [65]. It becomes even more challenging when objects of interest have similar appearances, like similarly colored balls or insects. In that case, the motion cues are particularly useful for discriminating multiple objects [78].

This chapter provides an overview of the basics of object tracking as well as current state-of-the-art methods which are relevant in regards to the implementation of the Crowdstream system.

There are two major topics to be considered when developing a MOT method. The first is how to measure similarity between objects, eg. the modelling of appearance (section 3.3), motion (section 3.4), interaction (section 3.5), exclusion (section 3.6) and occlusion (section 3.7). The other is how to associate identity information based on the similarity of objects between frames in a sequence, which usually involves inference and data association, covered in section 3.8. But first, section 3.1 defines the problem of

MOT.

3.1. Problem definition

The problem of MOT has previously been defined in numerous different ways in the research literature, depending on the perspective of the authors. However, Luo et al. [37] attempts to provide a general formulation and argues that existing work and formulations can be unified under their definition. Their definition states that MOT can be viewed as a multi-variable estimation problem:

> The objective of MOT is to find the ”optimal” sequential states of all the objects, which can generally be modeled by performing Maximal a Posteriori (MAP) estimation from the conditional distribution of following states given all the observations. [37]

This formulation unifies MOT methods from previous works by viewing them as approaches to solving the above MAP problem, either through *probabilistic inference* or *deterministic optimization*, which will be described in more detail in subsection 3.2.3.

3.2. Categories

Most MOT methods combine existing research and methods in new ways, making it difficult to categorize MOT methods distinctively. It is therefore useful to categorize the methods according to three different criteria: *initialization method*, *processing mode* and *output*.

	DBT	**DFT**
Initialization	automatic, imperfect	manual, perfect
# of objects	varying	fixed
Applications	specific type of objects	any type of objects
Advantages	can handle varying number of objects, can handle objects as they appear	no object detector required, can disregard appearing objects
Disadvantages	performance depends on object detection, added complexity with appearing objects	requires manual initialization, fixed # of objects

Table 3.1.: Comparison of the most important differences of DBT and DFT in regard to Crowdstream, adopted from [37].

3.2.1. Initialization method

MOT methods can usually be grouped into one of two sets based on how objects are initialized [2]:

- Detection-Based Tracking (DBT)
- Detection-Free Tracking (DFT)

The following sections provide an overview of these two methods, and Table 3.1 provides a comparison of the most important differences of these two methods in regard to the Crowdstream system.

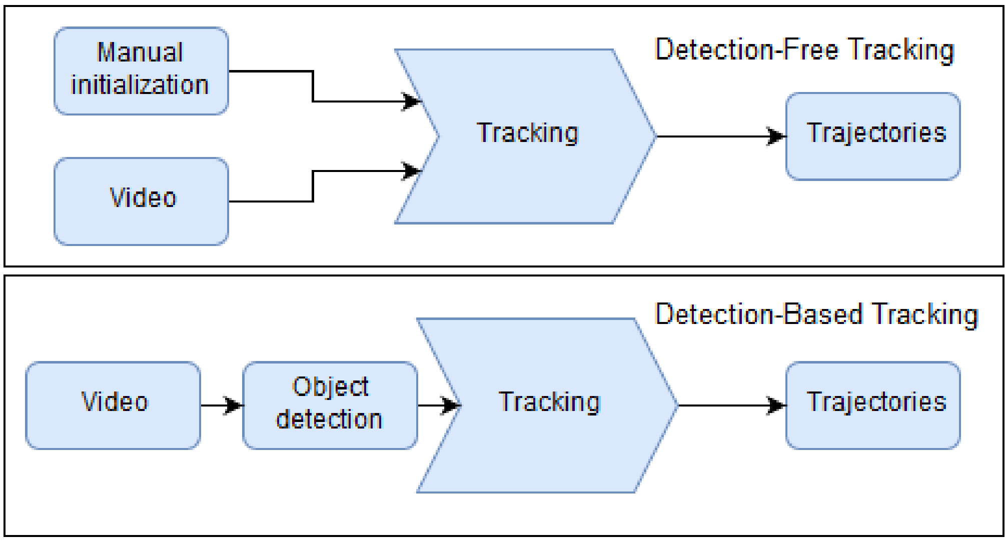


Figure 3.1.: Two common object initialization approaches for MOT. **Top:** Detection-Free Tracking (DFT), **bottom:** Detection-Based Tracking (DBT) [37].

Detection-Based Tracking (DBT)

When using DBT, also commonly referred to as "tracking-by-detection," objects are first detected and then associated with trajectories. This can be seen in Figure 3.1, where the video is first processed by the object detector, before being passed to the object tracker. There are two issues worth noting. First, since the object detector is trained in advance, the majority of DBT focuses on specific kinds of targets, such as pedestrians, vehicles or faces. Second, the performance of DBT highly depends on the performance of the employed object detection [37].

Detection-Free Tracking (DFT)

As shown in Figure 3.1, the tracking system requires both manual initialization in addition to the video that is being processed as input. DFT requires manual initialization of a fixed number of objects in the first frame and then proceeds with data association and tracking of these objects in subsequent frames [37]. A major issue with DFT is that it cannot handle objects that appear after the initialization phase, making it unfeasible for many practical scenarios. It does, however, work well in the cases where it can be used.

3.2.2. Processing mode

MOT can also be categorized into *online* and *offline* tracking, with the difference being whether observations from future frames are used when processing the current frame or not. Online, also called causal, tracking methods rely only on past information, while offline, or batch tracking approaches utilize observations both in the past and in the future [37]. Table 3.2 provides an overview of the most important differences between online and offline processing as it pertains to the Crowdstream system.

Online Tracking

In online tracking, object trajectories are estimated using only information from past frames as well as the current frame, e.g., [80], [80]. The trajectories are processed in a step-wise manner, and thus online tracking is also called sequential tracking. Online MOT algorithms are applicable to real-time applications such as advanced driving assistant systems and robot navigation [78].

Offline Tracking

Offline tracking can utilize detection results from both past and future frames, eg. [61] [76]. Sequences can be processed as one, or they can be split into batches, e.g., to reduce the computational load. Individual detections are linked together to form tracklets, which can then be combined either iteratively or in a time-sliding window [72], to construct longer trajectories that span the entire sequence.

	Online tracking	**Offline tracking**
Input	past and present observations	all observations
Methodology	append current observations to existing trajectories	link observations into tracklets, link tracklets into trajectories
Advantages	suitable for online tasks	can obtain global optimal solution, can utilize more data for tracking, no hard time-constraints,
Disadvantages	limited time for processing, limited detection data available,	potential processing delay, can not be used for online tasks

Table 3.2.: Comparison of online and offline tracking methods

3.2.3. Output

The last category classifies MOT methods based on their output, which can be either deterministic or probabilistic. The output of a deterministic tracking method is constant for the same input, while the output of a probabilistic method might vary between runs given the same input. The difference between these methods results from the optimization methods as mentioned in section 3.1.

3.3. Appearance model

Single object tracking methods often rely heavily on a visual appearance model to discriminate the foreground from the background and to measure the similarity between objects. While appearance modeling is an important aspect of MOT, it is usually not considered to be the core component.

Technically, an appearance model consists of two components, *visual representation* and *statistical measuring*. The visual representation describes the visual characteristics of an object, while statistical measuring is the computation of similarity between different observations of objects. Formally, the similarity between two observations i and j can be written as

$$S_{i,j} = F(o_i, o_j)$$

where o_i and o_j are visual representations of different observations, and F is a function that measures the similarity between them [37].

3.3.1. Visual representation

An object can be described according to a number of different visual features, using either a single one or a combination of several features. These features can be grouped into the following categories:

Local Features

Local features use local search to detect features in an image. The Kanade–Lucas–Tomasi feature tracker is an example of a method that utilizes local search.

Region features

Region features are obtained from a larger segment of an image, eg. a bounding box. Typical features are color histogram [26], raw pixel templates, gradient-based representations [7] and level-set formulation [43].

Others

There are a few other types of representations besides local and region features, such as the Probabilistic Occupancy Map (POM) [5] and gait features [64].

3.3.2. Statistical measuring

Statistical measuring is the second component of the appearance model and is used to measure the similarity between two objects. Some methods utilize a single measure, while other methods use a combination of measures.

Single measure

Single measures are simple to calculate and use, and several single measure similarity models exist. Some common measures are color, shape, depth and texture. The Bhattacharyya distance is commonly used to calculate the distance between two color histograms c_i and c_j, where the similarity S is calculated using Bhattacharyya distance

B as $S(T_i, T_j) = exp(-B(c_i, c_j))$ [72]. Another commonly used method is the Normalized Cross Correlation (NCC), which can be used to calculate the similarity between two objects based on the raw pixel template mentioned above in 3.3.1 [47].

Multiple measures

Combining multiple complementary measures can increase the models' robustness, but comes at the cost of increased complexity and computation. Fusing information from several measures can be difficult and error-prone. Fusing can be achieved using one of several available strategies for data fusion. Common strategies are boosting, concatenation, summation, product and cascading.

3.4. Motion model

The motion model captures the dynamic behavior of an object and predicts the next position of objects in the future frames [37]. Intuitively, it predicts the next position of each object along each track, and objects that are close to the predicted position are given a high similarity score. Also, some potential matches can be pruned early as they are highly unlikely. In many cases, objects can be assumed to move at a constant velocity and not change directions abruptly, thereby significantly reducing the search space. Below follows a brief description of three common motion models.

3.4.1. Static motion model

The static motion model is by far the simplest model and assumes a constant position with a bound on the maximum inter-frame motion. It works best when the velocity is low and constant, overall movement is small, and object overlap is infrequent.

3.4.2. Linear motion model

The Linear Motion Model is by far the most popular model, and it strikes a good balance between performance and complexity [7]. This model assumes that objects move with constant velocity V, eg. $V(t) = V(t+1)$ where t is a point in time. An example of a basic model using the linear motion model is Three-Frame Constant Velocity Prediction. Other commonly used implementations are velocity smoothing[42], position smoothing [72] and acceleration smoothing [32].

3.4.3. Non-linear motion model

In cases where the linear model cannot accurately model the object dynamics, a non-linear motion model can be considered. It can be deployed to model objects that move freely or in a non-linear fashion, e.g., objects that are being affected by gravity. However, Non-linear Motion Models are complex and can be computationally expensive.

3.5. Interaction model

The *interaction model* captures how objects interact and affect each other. The interaction model is also known as the *mutual motion model*, and an example of such interaction is when a crowd of people moves across a street; as each person follows someone, they are also guiding others. Another example is when a pedestrian walks along the boardwalk, they adjust their velocity and direction in order to avoid collisions. These are examples of two common interaction models, namely *crowd motion pattern models* [25] and *social force models* [23] respectively.

3.5.1. Social force models

Social Force models, also known as Group Models, considers all objects to be dependent on other objects and environmental factors. These models can assist in tracking objects in a group by modeling objects as agents with behavior based on two aspects *individual forces* and *group forces*:

Individual forces. Two types of forces are considered for each object in a group:

- *fidelity*, an agent should not change his desired destination
- *constancy*, an agent should not suddenly change momentum, including velocity and direction

Group forces. Three types of forces are considered for a whole group:

- *attraction*, agents moving in a group will stay close to the group
- *repulsion*, agents moving in a group will keep some distance from other members of the group to be comfortable

- *coherence*, agents moving in a group will move with similar velocity and direction

Several publications exist where one or multiple forces have been modeled successfully, usually implemented as one or multiple energy-minimizing objective functions, e.g., [47].

3.5.2. Crowd motion pattern models

Crowd Motion Pattern Models can often assist in tracking objects in over-crowded scenarios, where tracking individual objects is difficult. In such cases, individual appearance- and motion-models may prove inadequate due to objects being hard to detect and distinguish, and crowd motion models may provide additional cues to aid tracking. Motion patterns are often learned through machine learning methods and applied as prior knowledge to aid in object tracking.

3.6. Exclusion model

The exclusion model is a constraint that is employed in order to avoid physical collisions when searching for solutions to a MOT problem and comes from the fact that two objects cannot occupy the same physical space. This is usually modeled by two separate constraints [41]:

- **detection-level exclusion** two devices detected in the same frame cannot be associated with the same object
- **trajectory-level exclusion** two trajectories cannot be infinitely close to each other

3.6.1. Detection-level exclusion

Two different approaches can be utilized to model detection-level exclusion:

Soft constraints Exclusion constraint is modeled by minimizing a cost function that penalizes any hypothesis that maps multiple trajectories to a single object in the same frame [41].

Hard constraints Enforces the constraint by disregarding any hypothesis that maps multiple trajectories to a single object in the same frame [38].

3.6.2. Trajectory-level exclusion

Similar to *soft detection-level constraints*, trajectory-level exclusion is often modeled by minimizing a cost function that penalizes any hypothesis where two objects that are close together have different trajectories. The penalty can be inversely proportional to the distance between two objects, or proportional to the spatial-temporal overlap between two trajectories [41]. In either case, this will generally suppress one of the two trajectories if they are close enough.

3.7. Occlusion handling

Occlusions are likely the most critical challenge in MOT and are a primary cause for loss of objects or tracks. Several different methods have been proposed in order to overcome the issues caused by occlusion.

3.7.1. Part-to-whole

The part-to-whole strategy builds on the assumption that parts of the object are still visible as the occlusion occurs, which is a reasonable assumption in many cases. Based on this assumption, steps are taken to observe and utilize the visible part of the object to infer the occluded parts. Next, as affinities between objects in subsequent frames are calculated, the occluded parts are ignored or given a lower weight than the visible parts [26].

3.7.2. Hypothesize-and-test

This strategy eliminates most challenges posed by occlusion by proposing and testing a multitude of hypotheses according to the actual observations at hand. There are several methods that can be used to generate the hypotheses, eg. Zhang et al. [81] generate a hypothesis pair by considering objects that are close and of similar scale as occlusions of one another. Further, these hypotheses can then be used as input where MAP can be used to find the optimal solution, as described in section 3.1.

3.7.3. Buffer-and-recover

The buffer-and-recover strategy stores the last known state of objects as they become occluded, and uses this information to recover their trajectories once the occlusion ends. Mitzel *et al.* [43] keeps trajectories alive for a maximum of 15 frames during occlusion, and extrapolates the position during occlusion based on velocity and direction before the occlusion. When the occlusion ends, the track is matched with the trajectory and the object identity is recovered.

3.8. Inference

Inference in MOT can generally be categorized as using one of two approaches, *probabilistic inference* or *deterministic optimization.*

3.8.1. Probabilistic inference

Methods that are based on probabilistic inference usually represent the states of objects as a distribution with some uncertainty. The goal of the tracking algorithm then is to estimate the probabilistic distribution of the target state based on all existing observations. [37]. Probabilistic inference is particularly well suited for online tracking, as it requires only the past and present observations, and a Markov Property is often assumed.

Several different methods for probabilistic inference have been utilized with great success in MOT, but the most popular methods are Kalman filter [50], Extended Kalman Filter [43] and Particle Filter [7].

3.8.2. Deterministic optimization

Unlike approaches for probabilistic inference, methods based on deterministic optimization attempts to find the MAP solution, using observations from all frames or a sliding window. To achieve this, the problem of MOT is usually modeled as an optimization or energy minimization problem. Such methods are often better suited for offline tracking because observations from some or all frames need to be known in advance.

In practice, approaches based deterministic optimization have proven more popular than probabilistic methods. Despite probabilistic methods providing a more intuitive and optimal solution to the problem, optimization methods often provide a good enough solution in less time and are easier to infer.

Given the observations from all frames or a window of frames, these methods aim to associate all observations from a unique object with a trajectory, ideally finding the optimal association. Several approaches have been used successfully, and some of the most popular ones are *Bipartite Graph Matching* using a greedy assignment algorithm [7] or the optimal Hungarian algorithm [72], *Dynamic Programming* [5], *Min-cost Max-flow Network* [81], *Conditional Random Field (CRF)* [76] and *Maximum-weight Independent Set (MWIS)* [8].

3.9. Summary

Both Multiple Object Tracking and Multiple Target Tracking play an essential role in modern computer vision and have gained increasing attention due to their academic and commercial potential. The objective is to estimate object trajectories and maintain their identities through sequences of images. It has a wide range of applications, such as visual surveillance, sports analysis, robot navigation and autonomous driving.

The majority of recent research and progress in regards to MOT has been in the field of tracking-by-detection, where object detections from an object detector are linked in order to form trajectories and associate them with the identities of the objects. However, other methods exist, such as tracking-by-decision[71], based on Markov Chains and Monte Carlo Data Association.

The task of MOT becomes significantly more challenging when the objects of interest have similar appearances.

4. Device Positioning and Association

This chapter describes the details of the method that has been developed and implemented in the Crowdstream prototype. section 4.1 explains the general approach that was used, and also discusses some of the most important choices and decisions that were made during the development of the solution. section 4.2 and section 4.3 discusses the most relevant and applicable theory from chapter 2 and chapter 3 respectively. section 4.4 provides a brief summary of the conclusions and final approach that has been used.

Some decisions were made despite knowing that they are not optimal solutions, often due to time constraints or because the client agreed that the proposed solution was adequate. In the cases where sub-optimal solutions were chosen, the alternatives are mentioned and discussed in chapter 8.

4.1. Positioning

As stated in section 1.2, the client wanted to investigate if a solution using only a stage-mounted camera was feasible, which narrows the possible approaches. Given this requirement, a few possible solutions were proposed and discussed. In co-operation with the client, it was decided that an approach based on discrete-time signals should be attempted, as it seemed to be the most viable option.

4.1.1. Discrete-time signals

The idea is that each device emits a unique discrete-time signal, which can uniquely identify the device in a recording. This allows the system to capture a recording during a specific time when each device is emitting a unique signal, and then process the recording in order to detect the relative placement of each device in the crowd. This position can then be mapped to a device, identified by its IP-address, by matching the position of each detected sequence in the recording to the IP-address where each respective sequence was sent to.

The initial solution attempted to combine display colors with discrete-time signals, but this proved to be ineffective due to the fact that at longer distances these colors appear almost indistinguishable, such as in Figure 4.1.

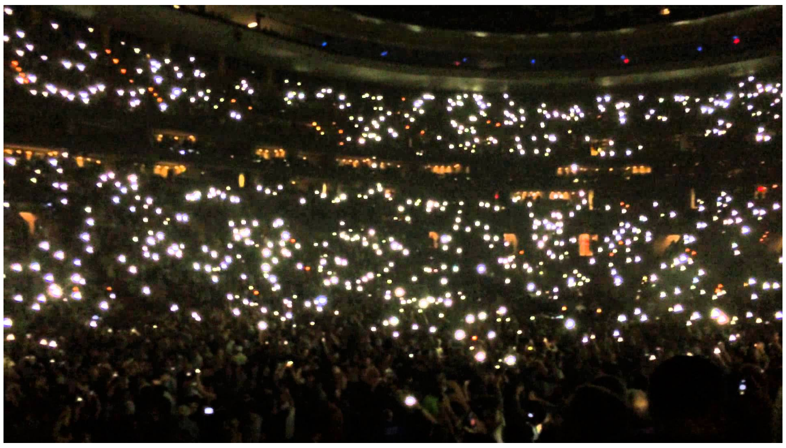


Figure 4.1.: Example of how colors from a LED screen appear indistinguishable at a distance.

The final solution is based on binary discrete-time signals or binary modulation. In practice, this is a sequence of bits, which uniquely identifies each device. Each bi‘ in the sequence, hereafter called *symbol*, signifies a white, ‘1‘ or black ‘0‘ screen.

4.1.2. Sequence generation

The sequences are generated randomly, one for each connected device, with some restrictions. The sequence has a pre-set, configurable length, and the length of the sequence needs to be large enough such that it can assign a unique sequence to each device. Also, in order to avoid certain types of detection errors, the sequence generation algorithm will output a maximum of 3 symbols of a given type in succession. This has to be accounted for when setting the length of the sequence, as it reduces the number of devices a sequence of a given length can identify.

In addition, each sequence has a pre-configured prefix of length 3, which is used by the algorithm to detect the start of a sequence, but these are added in addition to the configured sequence length.

4.1.3. Symbol rate

Due to randomness introduced by the client emitting the signals, described in section 5.4, the timing of each symbol the signal is not entirely constant or predictable. To account for this, the client emitting the signal must track the difference between the target duration for each symbol, or target unit interval, and the actual duration of each symbol.

The target duration can be calculated as follows, where the *target fps* is provided by the server, and is sent to each client together with the sequence:

$$target\ duration = 1000ms/target\ fps$$

The actual duration must be tracked by the client using a timer or similar construct, and the difference (Δ) can then be calculated as:

$$\Delta_n = target\ duration - actual\ duration_n$$

The client then has to adjust the duration of the next symbol $n + 1$ in the sequence according to this difference:

$$duration_{n+1} = target\ duration + \Delta_n$$

By doing so, the estimated end of $symbol_{n+1}$ in the sequence occurs at a time relative to the start of the sequence

$$t_{end_{n+1}} = n + 1 * target\ duration$$

,

and the whole sequence of length m ends at

$$t_{end_m} = m * target\ duration$$

This predictability is vital in order to accurately sample the signal, as described in subsection 4.1.5. Figure 4.2 shows how this affects the signal in practice. The signal starts at $t = 1$, but the duration of the first symbol is too short due to random factors. The target duration of the signal at $t = 2$ is therefore increased to compensate, which in this case results in a correctly timed transition at $t = 3$. The transition to $t = 4$ is correct, so no adjustments are made to the duration of the symbol starting at $t = 4$. The duration of the symbol at $t = 4$ ends up being too long, so the duration of the symbol at $t = 5$ is reduced to compensate. However, this the duration of the next symbol is also extended by the device.

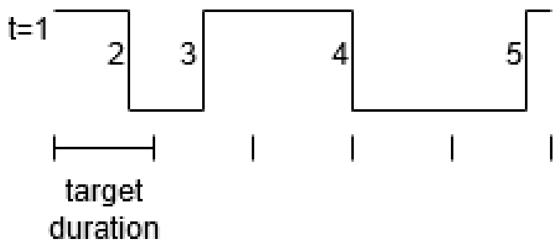


Figure 4.2.: Symbol rate and symbol adjustments for the sequence 101010.

4.1.4. Capture FPS

Another issue is caused by the fact that the camera does not record at a constant FPS. The FPS during recordings will usually vary between 15-30 FPS, depending on the environment and lighting conditions. This is caused by the camera adjusting the shutter speed according to the lighting conditions, in order to avoid over- or under-exposure during the recording. Given more advanced hardware the shutter speed could be controlled manually, but such low-level control is not available for the simple hardware used in this project.

To account for this, the system uses a capture rate that is at least three times the signal rate in order to minimize errors. Specifically, the symbol rate is adjusted such that

$$symbol\ rate \leq capture\ fps/3$$

Because the capture rate can vary between 15 and 30 fps depending on conditions, the signal rate should in most circumstances be set to

$$15/3 = 5hz$$

or less, to account for the worst possible lighting conditions. This allows the system to sample each unit interval at least three times, possibly up to 6 times when capturing at 30 FPS. This allows the system to utilize *triple modular redundancy* or *N-modular redundancy*, described in subsection 4.1.6, in order to detect and correct transmission errors. Because the system uses binary modulation, the symbol rate is also the bit rate of the signal.

While the frame rate can vary, the timing between each frame at a given frame rate is relatively constant and predictable, such that an fps of 15 will roughly capture frames at times $t = 0ms$, $t = 33ms$ and $t = 67ms$. It is important to note that, due to the variance described in the previous subsection, this does not ensure that each symbol is sampled three times, only that the system captures at least three frames each unit interval.

Figure 4.3 illustrates an example of when frames might be captured using a variable frame rate between 15 and 30 frames, on a sequence of 10101. The frame rate is close to 30 for the first two symbols, capturing 6 frames per symbol. Then it decreases to 4 and 3 for symbol number 3 and 4 respectively. While the frame rate increases again for the last symbol, capturing 5 frames during the unit interval, it only manages to capture 2 samples of the actual symbol. This is due to the timing of symbol 5 being off by more than half the unit interval.

Such significant differences between the target and actual duration will likely lead to errors in the detection algorithm, but this rarely occurs in practice. Possible solutions can be better hardware that is able to capture at a higher FPS, optimizing the signal emitter to better match the target unit interval, or changes to the detection algorithm itself.

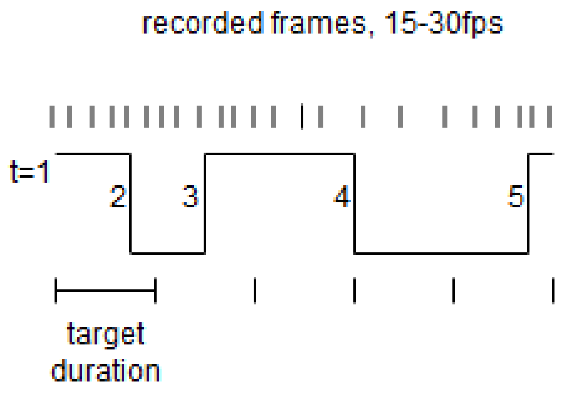


Figure 4.3.: Recorded frames with a variable FPS between 15 and 30 frames.

4.1.5. Discrete-time signal sampling

Because the signal is binary and discrete, if the unit interval was constant and predictable, the system could efficiently sample each symbol exactly once at each interval. However, in this case, as the unit interval is neither constant nor predictable, this is not a viable option. Therefore, the system utilizes information from all the captured frames in an attempt to minimize errors.

Since the FPS the device captures at varies, it is not possible to determine precisely when a specific frame in the video sequence occurred based on the recording itself. To accurately sample the signal at specific times, the system generates a list of when, relative to the first frame, each subsequent frame in the sequence occurred. This data is generated while the camera is recording by utilizing a timer and comparing the captured time of each individual frame with the time of the first frame, and storing the result.

Through this mechanism, it is possible to accurately sample the signal at specific intervals in order to detect the actual sequence.

4.1.6. Triple to N-modular redundancy

The system utilizes Error Correction Codes (ECC)/Forward Error Correction (FEC), specifically modular redundancy, in order to detect and correct errors in the transmission from the clients. By using a capture FPS that is at least three times the signal frequency, the system is guaranteed triple modular redundancy, and potentially higher redundancy if the FPS is higher.

Starting with the first detected symbol, the system will use information from every recorded frame in order to accurately sample the signal. The detected symbol is corrected by a *majority vote* or *democratic voting* where the number of positive and negative results of each sample within the calculated time-span of a symbol are counted and compared. If there is a tie, the sample that lies closest to the center of the calculated time-span is chosen.

Figure 4.4 demonstrates this process in detail. The process proceeds as follows, where Symbol N starts at $t = N$:

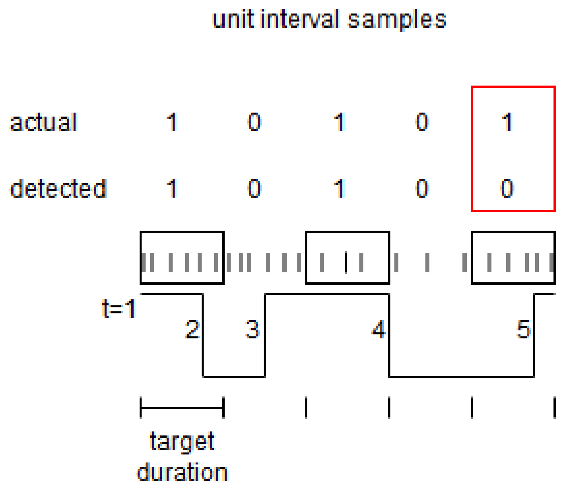


Figure 4.4.: Sampling and detected symbols

Symbol 1 The capture FPS is 30, yielding 6 frames within the unit interval of the symbol which corresponds to six samples. Five samples detect ‘1‘, one sample detects ‘0‘, the result is ‘1‘. The actual unit interval of the symbol is too short, but this does not impact the outcome of the result.

Symbol 2 The FPS is still 30, resulting in six samples. Three samples detect ‘0‘, the next three detect ‘1‘, which is a tie. The sample closest to the center of the unit interval is sample number four, which detects ‘1‘, therefore the result is ‘1‘. The actual unit interval is off by half, but the detection is still correct.

Symbol 3 The capture FPS is reduced, capturing four frames in the unit interval. All samples detect ‘1‘, the result is correctly determined as ‘1. The signal also transitions to the next symbol at the correct time for the first time during the sequence.

Symbol 4 The FPS is further reduced, yielding only three samples. Due to the transition being correct, all three samples correctly detect ‘0‘, and the result is ‘0‘.

Symbol 5 The capture FPS increases again, providing five frames during the unit interval. The detected samples are three ‘0‘ and two ‘1‘, the result is therefore ‘0‘, which is incorrect. This is caused by the fact that the transition from Symbol 4 to Symbol 5 is delayed by 3/4 of the unit interval. Such large discrepancies between the *target* and *actual* unit interval will result in incorrect results, displaying a possible flaw with the detection algorithm. However, differences of this magnitude have *not* been detected in practice thus far, but are in theory possible due to the ‘best-effort‘ scheduling provided by the operating system on the devices.

4.1.7. Network latency

Network latency is by no means a new problem, but it nonetheless has to be accounted for by the Crowdstream system. In practice, network latency will affect when each device receives the sequence signal, and therefore also when each device can begin signaling its sequence. One possible approach would be to include a time stamp of some time in the near future together with the signal, which indicates when the signaling should begin. While the solution is feasible, it introduces unnecessary constraints and complexity on the clients.

Another approach is to let each client start the signal at their leisure, provided it is within a certain amount of time after the sequences were sent, while the camera is still capturing. This effectively eliminates any issues regarding network latency, and as a side-effect also any other problems that could cause the sequence to start too early or too late. This is the approach used by the Crowdstream system.

The system begins recording just before the sequences are sent, and will record for a limited amount of time, e.g., twice the time it takes to display the full sequence. All sequences are prefixed with a single known sequence-prefix starting with ‘1‘, for example ‘101‘, which signals the start of a sequence. As the tracking algorithm processes each frame it uses this prefix to detect the beginning of each individual track, disregarding the prefix itself. Next, it stores the capture time of the last frame of the prefix as the start time of the actual track. Then, the timing of all subsequent samples for that track is treated relative to this stored start time, ignoring the start of the actual recording. This lets all tracks operate with separate start times within the captured video, such that it does not matter when devices begin signaling their sequence as long as the whole sequence can be captured by the camera.

4.1.8. Summary

The result is an indoor positioning method which in theory can be used to determine the relative position of multiple devices in a crowded area, something that has generally been considered a tough problem. It accomplishes this using only a camera, a server and the devices themselves. The method is general enough such that it and can easily be extended, improved or customized by changing or replacing one or more of the components or algorithms involved.

The next chapters discuss how this method has been combined with methods for device detection and tracking in a prototype system to solve the exact problem as stated by the client, to position devices in a large audience at a concert or in an arena environment.

4.2. Device detection

This section provides a detailed discussion of some of the most relevant methods mentioned in chapter 2, in context of the Crowdstream system. It covers their applicability, potential issues, advantages, and drawbacks. It also mentions which methods that were ultimately chosen and implemented in the prototype, and why.

4.2.1. Visual analysis

As mentioned in section 2.2, there exists a large number of methods and algorithms for visual analysis. For this particular task, simple thresholding appears to be a viable option. It has low complexity and is computationally efficient, while also providing good results for specific inputs. Low complexity is important due to the limited time constraints of the project, while computational efficiency is necessary to avoid making the audience wait for prolonged amounts of time while processing the input. The images that will be processed by this system also happen to be a good fit for simple thresholding, because the devices in the image are bright, and easily distinguishable from the black background in a histogram. The following section describes the thresholding implementation in detail.

Thresholding method

The most commonly used threshold selection algorithms assume that the intensity histogram of the image is multi-modal, typically bi-modal such as the histogram in Figure 4.5. However, in the case of Crowdstream, the input images are essentially uni-modal, as can be seen in Figure 4.6. This is caused by the fact that the background pixels present the majority of the pixels present in the image, and the black background, therefore, dominates the foreground pixels in the intensity histogram.

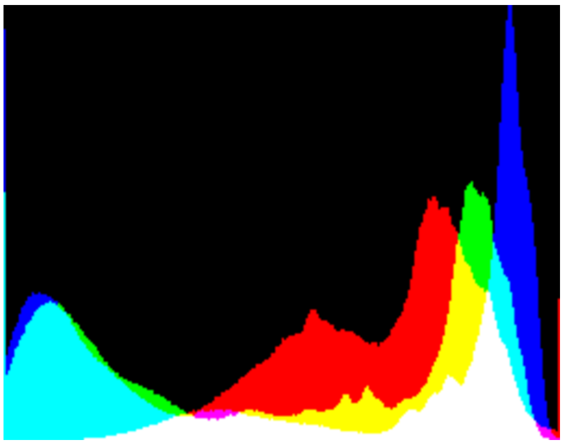


Figure 4.5.: Example of a multi-modal histogram, based on a nature photography.

Under these circumstances, many of the commonly used algorithms will fail or perform poorly, such as *Otsu's method*, as can be seen in Figure 4.7. It shows how some nearby devices are detected as a single device, resulting in false negatives. It also incorrectly detects devices at the top of the image, false positives, which are in fact other sources of light or reflections with much less intensity than the mobile devices. Compared to the

Figure 4.6.: Example of uni-modal histogram, based on an image captured with Crowdstream.

detection result in Figure 4.8, which uses a manually defined threshold, one can see that using a manual value provides a more accurate result, with both less false negatives and less false positives.

Input	**257 devices**		**253 devices**		**675 devices**	
	FN	FP	FN	FP	FN	FP
Manual threshold	5	3	8	4	24	19
Otsu's Method	12	15	10	18	36	95
Guassian Blur + Otsu's	13	18	19	14	107	36

Table 4.1.: Comparison of commonly used threshold selection algorithms on 3 different real-world sample images. It shows the amount of False Negatives (FN) and False Positives (FP) for each algorithm as they are applied to each image.

Tests on real-world sample images showed that both a direct application of Otsu's Method and a combination with a Gaussian Blur filter was being outperformed by simple manual global thresholding, as can be seen in Table 4.1. For these images and this specific use-case, Otsu's method generally selects a threshold value that is too low. This causes it to detect devices where there are none, leading to a large number of false positives. It also causes devices that are clustered near each other to be combined into a single device, missing the other nearby devices, which increases the number of false negatives. A commonly used technique, which involves pre-processing the image with a Gaussian blur filter before applying Otsu's Method thresholding, only seems to magnify the issues by combining even more nearby devices into a single detection.

There does exist a few threshold selection algorithms that are specifically designed to

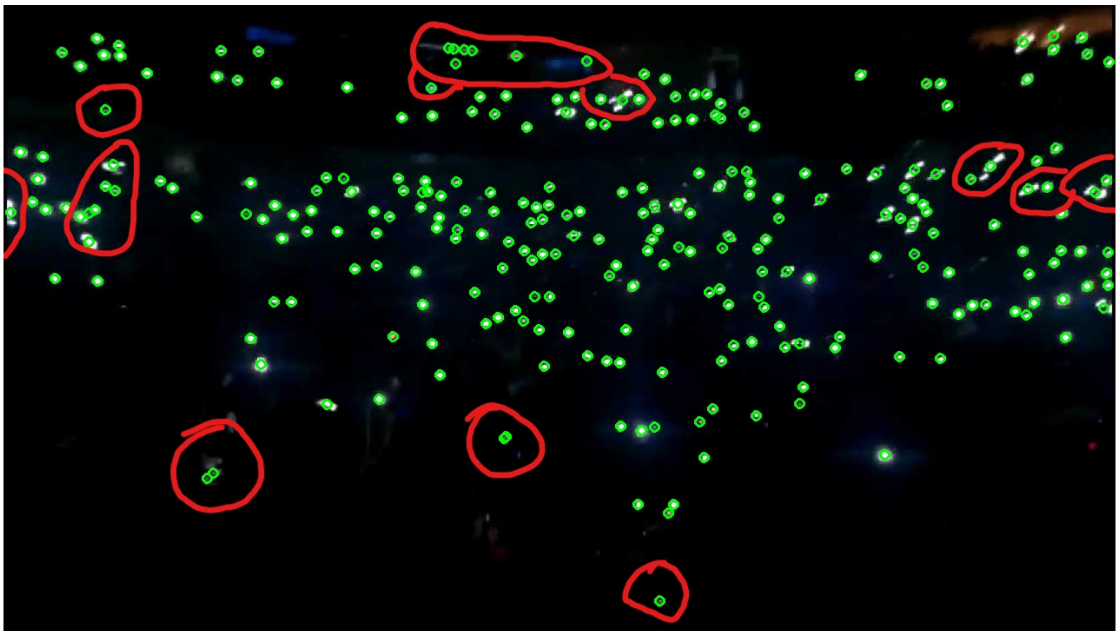


Figure 4.7.: Example of Otsu's Method thresholding applied to a real-world image sample with 257 devices. Green circles indicate devices that were detected by the algorithm. Red circles highlight incorrect detections.

handle uni-modal images, as mentioned in subsection 2.2.1. However, because all the histograms of the images that are being processed by the Crowdstream system are simple and predictable, see Figure 4.6, a simple binary thresholding algorithm performs well.

The system uses a manual global threshold which can be tuned to the specific environment the system is operating in. Depending on the brightness of the surroundings, a value between 200 and 240 will usually provide adequate results.

4.2.2. Motion analysis

Research has shown that sophisticated methods do not always produce more precise results than their simpler counterparts, and they also introduce a significant computational cost and complexity. It has also been shown that most background modeling methods can handle static backgrounds fairly well, while the real challenge lies in adapting them to dynamic backgrounds and environments[44]. The video recordings that will be processed by the system will have a static background and a static camera, reducing the need for complex motion analysis.

Also, as described in section 2.3, motion analysis provides best results when working

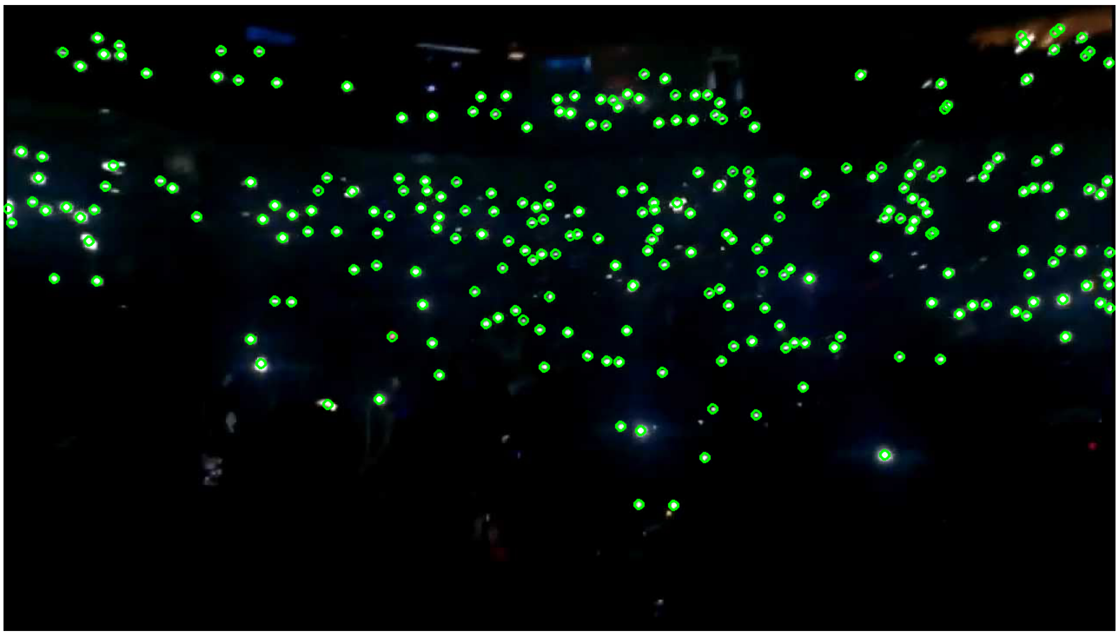


Figure 4.8.: Example of manual binary thresholding, using a value of 240, applied to a real-world image sample with 257 devices.

with fast moving objects. However, this system will be processing video captured during a time where the audience is asked to hold their devices still, which does not lend itself well to any motion analysis segmentation algorithms.

Feature descriptors

Modern feature detection and extraction algorithms will likely introduce a lot of complexity to this prototype of Crowdstream, and given the time constraints of the project, it is not feasible to attempt to implement any such algorithms as this point.

4.2.3. Summary

For the purpose of this prototype, the detection method will be based on manual thresholding. Thresholding is simple and effective and works well with the types of input this particular system will process.

4.3. Device tracking and association

Crowdstream differs from other common use cases of device tracking and association because the objects that will be tracked by the system are expected to disappear and frequently reappear, due to the nature of the sequence and positioning phase that occurs during tracking. Many tracking methods do not handle occlusion well, and some not at all, which effectively makes many of them unusable for this application.

4.3.1. Initialization method

The difference between DBT and DFT is whether a detection model is adopted (DBT) or not (DFT), which ultimately dictates whether the tracking algorithm can track new objects as they appear during the tracking process. Because the Crowdstream system involves network latency and other random factors, it cannot guarantee that all objects will be detectable in the first frame. The system must, therefore, be able to track objects that appear during the tracking process, which in practice makes a DBT based method the only viable option.

4.3.2. Processing mode

The difference between online and offline methods is the way they process observations, which can also affect the tracking performance and computational requirements. Offline tracking is not suited for applications that require real-time tracking, but as Crowdstream has no such hard requirements, both options remain viable. There are usability aspects to consider in regards to both the speed and performance of the method used, so a solution has to strike a balance between the time required to complete the tracking, and the performance of the tracking method. Because real-time tracking is not a requirement, an offline method is likely the most viable option as it allows the algorithm to utilize more data and computation-time to improve the performance.

4.3.3. Output

The output of the tracking method in the prototype is deterministic, partly because it is easier to implement and also because it is easier to test and compare the results of different detection algorithms and changes in their parameters.

4.3.4. Appearance model

The color histogram is simple, and one of the most studied similarity measures, but it ignores the spatial features of the objects. The spatial features are often important for multiple object tracking, especially when objects have similar visual appearances such as in the case of Crowdstream. Gradient-based features can consider the shape of objects and are robust to certain transformations, but cannot handle occlusion well, which is paramount to the performance of the system. There are other more robust measures, but they are often computationally expensive or require multiple cameras or other setups.

Because the objects that will be tracked by the Crowdstream system are all rectangle-shaped lights, it is likely that shape and texture will not be able to contribute much towards a more robust similarity model. Likewise, because all the lights are white, color and color histograms will also most likely not be effective similarity measures.

Considering how objects being tracked by Crowdstream are almost visually identical, alternatives to appearance models, especially models that rely on spatial awareness and application-specific domain knowledge, have been emphasized.

4.3.5. Motion model

The targets tracked by Crowdstream will be hand-held devices. The objects might move abruptly and in a non-linear fashion, which reduces the general applicability of linear motion models. A non-linear motion model might be applicable, but will likely require a great amount of effort and resources, probably too much considering the scope of this project.

It is however likely that most objects will remain within a static region during the tracking phase, bounded by twice the length of the participants' arms. This lends itself well to a linear motion model, specifically a static motion model. This can be combined with a simple constant velocity or three-frame velocity prediction linear motion model to handle cases where two objects overlap.

4.3.6. Interaction model

Interaction models are difficult and time-consuming to implement and configure correctly and has therefore not been considered for the current prototype. However, social force models and crowd motion patterns could be viable for future development.

4.3.7. Exclusion model

As mentioned in section 3.6, exclusion modelling is usually handled by two separate constraints, detection-level exclusion and trajectory-level exclusion.

Detection-level exclusion The prototype utilizes a hard constraint to enforce detection-level exclusion. It disregards any hypothesis that maps multiple trajectories to a single object in the same frame, and will instead attempt to find a better hypothesis. This approach was chosen because of time constraints, it is simple to implement, and provides adequate performance. Soft constraints could possibly provide better results and should be considered for future work.

Trajectory-level exclusion The system does not use a model for trajectory-level exclusion, also due to time constraints. As with detection-level soft constraints, this requires an algorithm that utilizes cost functions, so implementing one will also enable the other with little effort. Trajectory-level exclusion should be considered for future work.

4.3.8. Occlusion handling

As mentioned in section 3.7, occlusion handling is one of the most critical parts of MOT. It is particularly important in the Crowdstream system because objects are expected to disappear and reappear at regular intervals, making many of the commonly used tracking and occlusion-handling algorithms unsuitable. The part-to-whole method, described in subsection 3.7.1, is one such example because the targets being tracked will intentionally be completely occluded and not detectable for significant parts of the tracking phase.

The prototype uses a Buffer-and-Recover method, similar to what was described in subsection 3.7.3. It uses prior trajectory, velocity, and a maximum radius, based on an arm's length, to attempt to detect and associate appearing devices with recently occluded ones. If a device cannot be rediscovered within a set amount of frames, it will be discarded for the remains of the session, to avoid it interfering with other trajectories.

4.3.9. Inference

Inference is a very effective tool in MOT, and would likely have a large impact on the performance of the tracking algorithm. However, due to time constraints and the fact that preliminary tests without inference provided adequate performance for the prototype, an inference model has not been implemented.

Both *probabilistic inference* and *deterministic optimization* could prove effective for this system. There is a lot of domain-specific knowledge and information that could be used to drive an inference model.

Most notably, the fact that the system knows all valid sequences in advance enables it to aggressively prune any hypothesis that would result in an invalid sequence, or in two devices having the same sequence.

The inference engine could also incorporate knowledge about how the devices are likely to move during the initiation phase, e.g., devices might be more likely to move horizontally than vertically. It is also plausible that devices will move within a defined area restricted by an arm's length, that their velocity will slow down as it nears the edge of the said area, and then abruptly change direction. Such information could enhance the accuracy and performance of the tracking algorithm if encoded and modeled successfully.

4.4. Positioning summary

The system uses detection-based offline-tracking with a deterministic output. In addition, it utilizes a simple linear-motion model, hard detection-level exclusion, and a method based on Buffer-and-Recover to handle occlusions.

5. Prototype Implementation

The project has resulted in a basic functional prototype system. The system includes a server application, a CLI for server administration, a client application for Android devices and a TCP/IP based protocol for communication. This chapter provides an overview of the implemented system as well as details on the implementation of each component.

To start, section 5.1 provides an overview of the system architecture and all its components. Next, section 5.2 describes the protocol that has been developed for communication between all involved components. Then section 5.3 provides a detailed description of the implementation of the server, which is the core of the system. Following that, section 5.4 and section 5.5 covers the implementation of the mobile client and the administrator client, respectively. Finally, section 5.6 details the implementation of a comprehensive debugging and testing framework that has been created to support the development of the system.

5.1. System architecture

The system consists of a server application and any number of clients, using a traditional client/server architecture. Figure 5.1 shows the overall architecture of the system, and how clients can communicate with the server. The clients can connect to the server over a network, and it uses a simple customized linefeed TCP/IP protocol for communication. Clients can connect to the server through a local WiFi connection at the venue, or by using a mobile network (cellular network). The admin CLI is only available as a python script, so it must be run on a platform that supports Python.

5.2. Crowdstream protocol

The protocol is a simple linefeed delimited Transport Connection Protocol (TCP) protocol, which receives and sends lines of text delimited by Carriage Return Line Feed (CLRF).

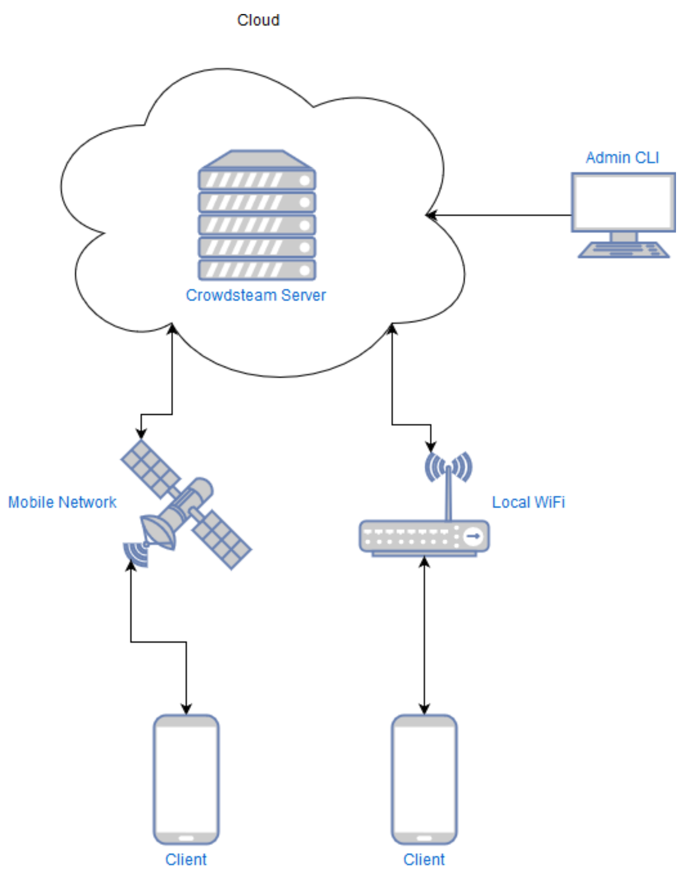


Figure 5.1.: The Crowdstream architecture

Each line is a list of Comma Separated Values (CSV), where the first token is a command, and the following tokens are arguments.

5.2.1. Server-to-mobile client communication

The server can send three different commands to a client:

Position response (*pos,[response]*)
The position response is sent as a response after receiving a clients GPS position and has the following parameter

- **response** A single value of ’0’ or ’1’

The response describes whether the device was rejected or accepted, where ‘0‘ indicates that the device was rejected and ‘1‘ indicates that it was accepted. If the device was rejected, the application should gracefully disconnect and notify the user.

Sequence (*s,[fps],[sequence]*)
The sequence command is sent by the server to initiate the sequencing phase, and includes the following two parameters:

- **fps** The target FPS the sequence should be displayed at
- **sequence** A sequence of ‘0‘ and ‘1‘s of arbitrary length

Once received by the device, the sequence of ‘0‘ and ‘1‘ should be displayed by the device as black and white, respectively. Each frame should be displayed for the duration indicated by the FPS parameter, where the duration of each frame is calculated as

$$duration\ in\ ms = 1000/fps$$

Color (*c,[r],[g],[b]*)
The color command defines a single color, using the Red Green Blue color model (RGB) scheme, that should be displayed by the device, and has the following parameters:

- **r** The value of red in a RGB value
- **g** The value of green in a RGB value
- **b** The value of blue in a RGB value

The color should be displayed immediately, and its duration is infinite. In practice, this color should be displayed until a new color is received or when the device is disconnected from the server.

5.2.2. Mobile client-to-server communication

The server only supports a single command coming from the client.

Position (*pos,[lat],[long]*)
The position command is used to send the current position of the client to the server. The position of the device is compared to the position of the server, which can be manually

set or automatically detected, and the result is used by the server to determine if the client is located in the correct area and allowed to participate in the event. It has the following two parameters:

- **lat** A GPS latitude value
- **long** The GPS longitude value

5.2.3. Admin-client to server communication

The server supports persistent administrator connections, but it can also be used with a simpler request/response model. To support a request/response model, the server will always reply with a single response for each command, regardless of what happened during the execution of the command. If the command was successful, the server will respond with any data requested by the command, or "OK" if no data was requested. If the command is unsuccessful, the server will respond with an error message describing what went wrong. As can be seen in Figure 5.2, the server can handle multiple commands over the course of a single admin session. It also shows how the server will notify the user of invalid commands if the user is already authenticated. However, if the server receives any commands other than 'pos' or 'auth', it will immediately terminate the connection. Finally, if the user tries to authenticate with an incorrect password, the server will respond with 'auth unsuccessful' before terminating the connection.

The server supports an authentication command as well as four different administrator commands.

Administrator commands

Authenticate (*auth,[password]*)
The authentication command should be issued at the start of an admin session to identify the client as an administrator. The command accepts a single parameter:

- **password** The administrator password

The password is used by the server to determine if the authentication is successful. If the password is incorrect, the connection is terminated immediately. The password must be configured on the server in advance and is transmitted to the server as plain text.

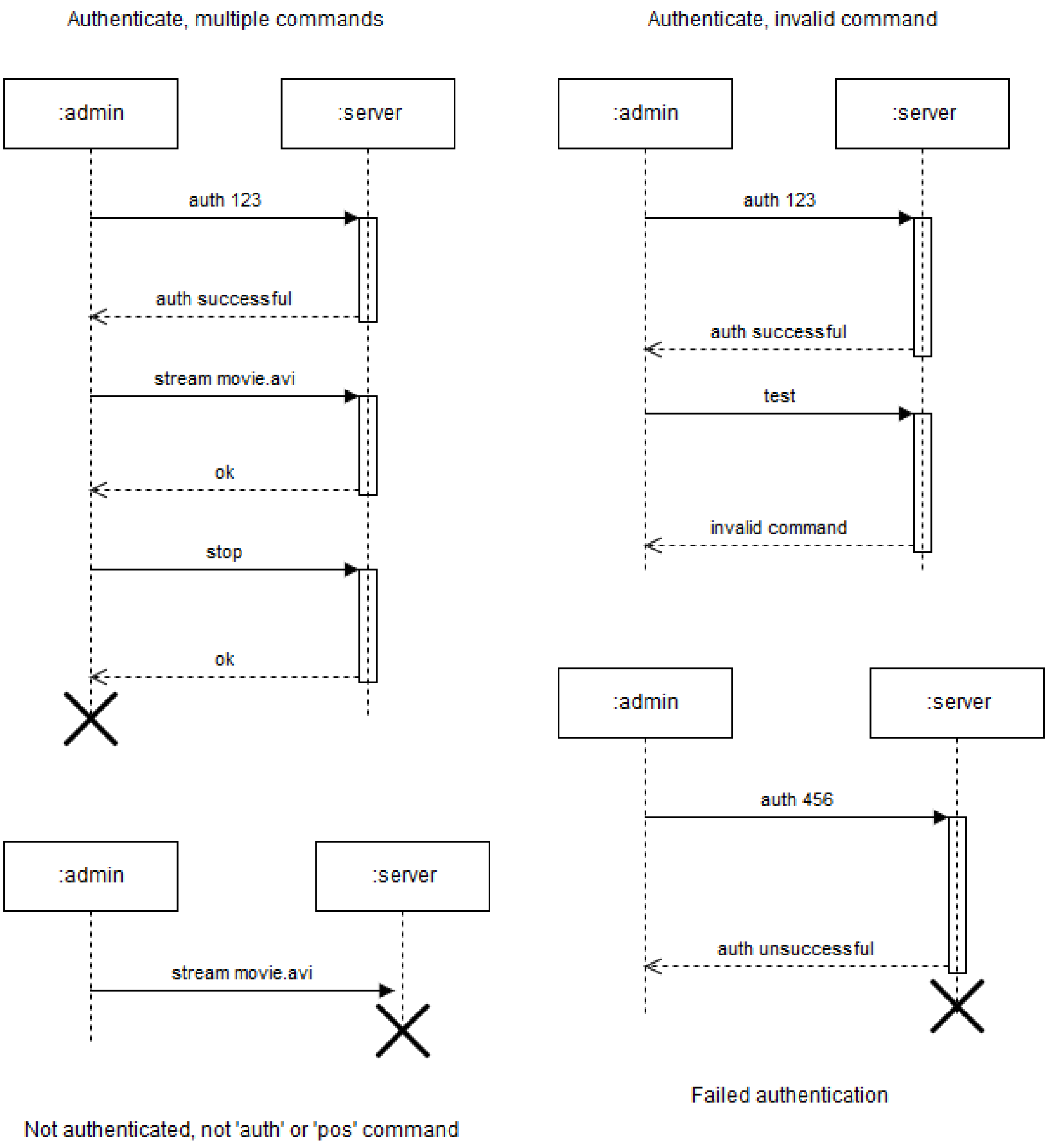


Figure 5.2.: Administrator protocol sequence diagram

Initiate (*init,[rounds]*)
The initiate command is used to start the sequencing phase, where a unique sequence of bits is sent to each connected device. The command accepts a single parameter:

- **rounds** The maximum number of sequencing phases/rounds to run

The ‘rounds‘ parameter can be used to run multiple sequencing phases, to maximize the number of devices detected. A value between 3 and 5 is recommended.

Status (*status*)
The status command can be used to retrieve the status of the server and the clients and accepts no parameters. The response includes a list of all connected devices and their status as ’detected’ or ’not detected’.

Start stream (*stream,[source]*)
The stream command is used to start the streaming of content to all currently detected devices, and accepts a single parameter:

- **source** The path of the content to stream, relative to the designated ‘content‘ folder on the server.

The source can be either an image or a video, and it will begin streaming immediately to all connected devices.

Stop stream (*stop*)
The stop command can be used to stop any current streaming, and it does not accept any parameters. The broadcast will stop immediately and is followed by a single request for all devices to display a black screen.

5.3. Crowdstream server

The server is implemented using Python 3.6. Python was chosen for its features and capabilities with image processing, networking, and rapid prototyping. It is also great for rapid deployment and is easy to set up and use. Lastly, it is one of the most popular languages in the scientific community, along with Matlab[1] and R[2].

The server is entirely asynchronous, spawning new threads to perform any background tasks and commands such as capturing and analyzing video, streaming and providing status reports.

[1]https://www.mathworks.com/products/matlab.html
[2]https://www.r-project.org/

Image processing is mostly handled through the use of OpenCV[3] and NumPy[4].

OpenCV is an open source computer vision and machine learning software library, while NumPy is a library that assists general scientific computing in Python.

The server optionally uses IPData[5] to retrieve its current location, which is used to determine whether mobile devices are close enough to participate in the event or not.

5.3.1. Architecture

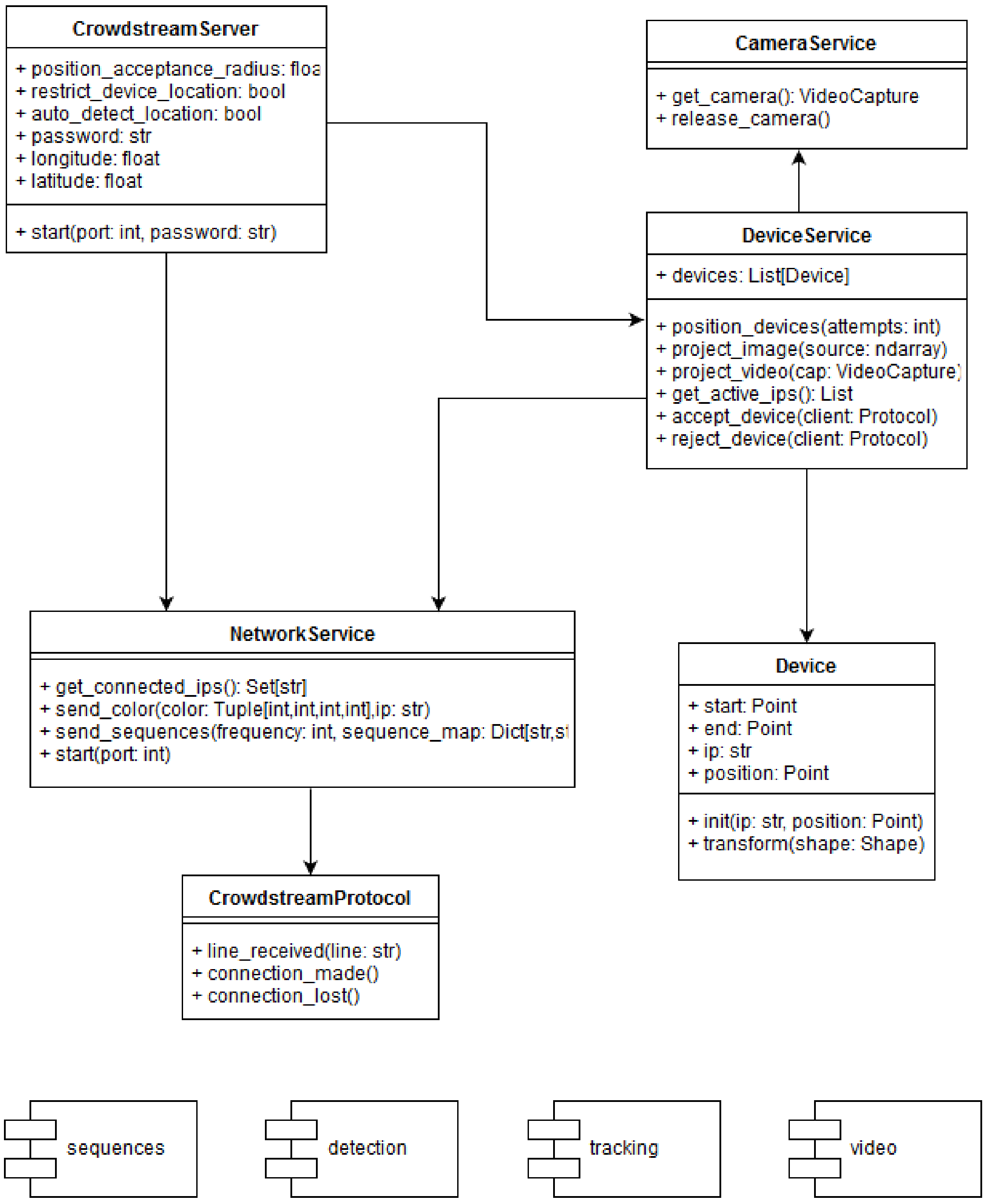
The overarching design is based on an object-oriented programming paradigm. However, some components such as the image processing and tracking components utilize functional programming. This is done to leverage the compositional nature of functional programming, which works well for pipelines such as image processing. Figure 5.3 shows the architecture of the server and how the different classes interact.

Classes

CrowdstreamServer `CrowdstreamServer` and its *start* method is the entry point for the server application. It has some configurable fields that reads its default values from the global configuration, but which can be overridden per instance before calling *start*. It has two dependencies, `NetworkService` and `DeviceService`, which handle network communication and the devices respectively. `CrowdstreamServer` processes all commands coming through the `NetworkService`, creating a new thread to execute the task if asynchronously if required. Some tasks might require interaction with the `DeviceService`, such as streaming video. It then sends the response back through the `NetworkService`. `CrowdmstreamServer` is also responsible for accepting or rejecting new connections, authentication, access control and server location.

CrowdstreamProtocol The `CrowdstreamProtocol` acts as a wrapper for each connection, managing the TCP/IP socket, connection, and protocol communication. It abstracts away the connection details and lets higher layers in the application interact with complete lines of text instead of individual TCP packets.

[3]https://opencv.org
[4]http://www.numpy.org/
[5]https://api.ipdata.co


CrowdstreamServer
+ position_acceptance_radius: floa
+ restrict_device_location: bool
+ auto_detect_location: bool
+ password: str
+ longitude: float
+ latitude: float
+ start(port: int, password: str)
CameraService
+ get_camera(): VideoCapture
+ release_camera()
DeviceService
+ devices: List[Device]
+ position_devices(attempts: int)
+ project_image(source: ndarray)
+ project_video(cap: VideoCapture)
+ get_active_ips(): List
+ accept_device(client: Protocol)
+ reject_device(client: Protocol)
NetworkService
+ get_connected_ips(): Set[str]
+ send_color(color: Tuple[int,int,int,int],ip: str)
+ send_sequences(frequency: int, sequence_map: Dict[str,s
+ start(port: int)
Device
+ start: Point
+ end: Point
+ ip: str
+ position: Point
+ init(ip: str, position: Point)
+ transform(shape: Shape)
CrowdstreamProtocol
+ line_received(line: str)
+ connection_made()
+ connection_lost()
sequences
detection
tracking
video
Figure 5.3.: Server architecture, simplified UML Class Diagram

NetworkService The `NetworkService` handles all communication with the clients. It is started when the server is started and will listen for connections and incoming data. As it receives new connections or data, it uses callbacks to signal events to other components.

It also manages all active connections as a collection of `CrowdstreamProcotols` . The network stack is described in more detail in subsection 5.3.2.

CameraService `CameraService` is responsible for managing the camera and its connection, making sure related resources are obtained, managed and released correctly.

Device The `Device` class holds information about a single, *located* device, such as the IP and its position. It also has a single method, *transpose*, that transposes the *detected position* of the device onto a provided matrix, and stores the result on the object.

DeviceService `DeviceService` has a fairly small API, but it and its dependencies are responsible for the majority of the complexity in the application. It manages all connected devices and provides methods to run the detection phase as well as controlling the stream and accepting or rejecting connecting devices. The implementation of this class is described in subsection 5.3.4 and subsection 5.3.5

Modules

The entities at the bottom of the diagram are implemented as stand-alone modules and are mostly used by `DeviceService`.

Detection This module contains functions to perform object detection on an image. The module uses a functional paradigm and is described in detail in subsection 5.3.5.

Tracking Contains functions and a class, `Track`, for tracking objects in a video, which is further described in subsection 5.3.5.

Video Utilities for working with video or images, such as writing a capture to disk and creating empty frames.

Sequences Helper functions for generating, encoding and decoding sequences.

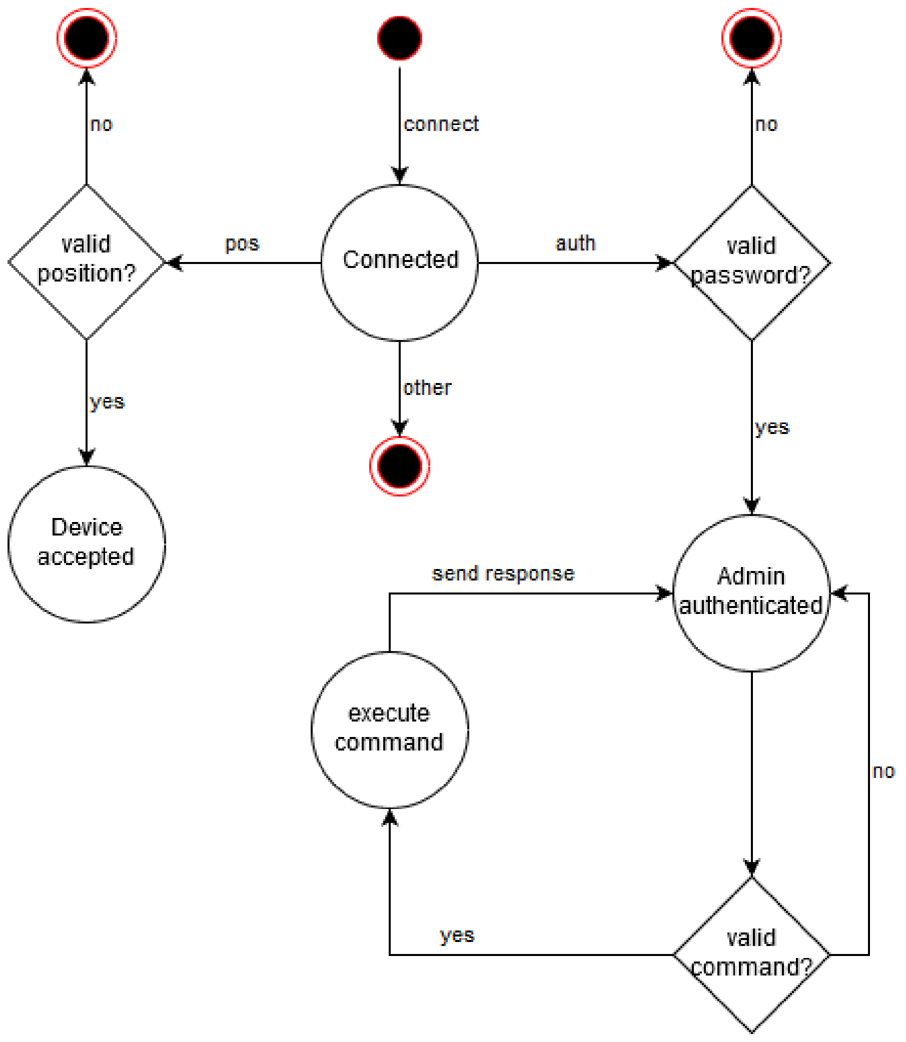


Figure 5.4.: Server state diagram

5.3.2. Networking

The network stack is built using Twisted[6], an event-driven, low-level networking engine. It offers high performance and provides access to low-level networking features and functionality. It works through a single 'main loop', the *event loop*, which blocks until an event arrives, and then calls the relevant event handler. The current implementation of Crowdstream uses a *select()* based loop, as it is supported on all platforms. Other, more optimized loops could easily be used instead if required.

Figure 5.4 shows a state diagram of the server and how it handles incoming connection

[6]https://twistedmatrix.com/trac/

and task execution.

5.3.3. Initialization phase

The initialization phase is started by calling `DeviceService.position_device()` with a single parameter, indicating the maximum number of attempts to locate all devices, which creates a new thread for the task. This begins a complex process of generating and sending sequences, capturing a recording and storing it to disk together with generated metadata, performing object detection, tracking, and association, and if necessary repeat the process a set number of times or until all connected devices are detected.

The process begins by checking how many undetected devices are currently connected, to avoid unnecessary work in case all devices are detected already. Next, it stops any ongoing tasks such as streaming and broadcasts a black color to all devices, to prepare them for the initialization phase.

Next, it will generate a unique sequence for each non-detected device, and assign it to their IP-address. Concurrently, a new thread will be spawned in an attempt to acquire a lock on the camera to warm it up, which includes adjusting the focus, shutter speed, ISO-sensitivity aperture, as well as capturing a small number of test frames. Once the camera and sequences are ready, the camera will start recording, and the sequences are sent to their assigned IP-addresses.

The camera will continue recording for the duration of the full sequence plus a few seconds of an extra buffer to account for network delays. During the recording, the exact time of each frame relative to the first frame is stored with millisecond precision. When the recording is complete, the recorded frames and the corresponding metadata is written to disk, the camera released, and the thread recycled.

The next part of the process is the detection and tracking of devices. It starts by opening both the recently captured recording and the related metadata file which holds the timing information for each frame. It performs object detection and object tracking, as described in subsection 5.3.4 and subsection 5.3.5, on each frame that was captured. The result is a list of `Track` objects, which holds the detected sequence and a history of all positions for each track.

Using this information, the system will then attempt to map the detected sequences to the original sequences that were broadcast. If a match is found, the IP-address corresponding to where the original sequence was sent is looked up and associated with the track. Finally, a new `Device` is instantiated using the IP-address combined with the position of the device track, resulting in a detected device with an associated IP and an estimated position.

Once all tracks are processed, if any connected IP-addresses were not successfully mapped to a position, the whole process will be rerun, up to a specified number of maximum attempts. Subsequent runs will exclude already mapped devices, and the system will attempt to keep these devices dark, to avoid interfering with the next attempt. This process has proven effective in tests, as it becomes substantially easier to map devices when there are fewer of them.

When all devices are detected, or the maximum number of attempts has been reached, the system is ready for streaming.

5.3.4. Detection

Object detection and related functionality is contained in the *detection* module. It has a fairly simple API, implementation and no external dependencies, such that it can easily be improved, extended or replaced. The current implementation utilizes OpenCV3 for image processing, but this is not exposed to the module's clients and can easily be changed.

The main function, `detect_devices`, accepts four input parameters:

- **input_image**: The input image to perform object detection on
- **threshold_func**: A function accepting an input image and returning an output image
- **find_contours_func**: A function accepting an input image, returning nested list of points indicating the detected contours.
- **find_contour_centers_func**: A function accepting a nested list of points, and returning a list of points

The result of applying all these operations in succession is a list of the centers of all detected objects.

The module contains several available implementations:

Thresholding

The thresholding function must accept a single parameter, the input image, and return the result of applying the threshold function.

- **Simple Threshold** Threshold function that applies a manually defined global threshold to the whole image.
- **Otsu Threshold** Implementation that uses Otsu's Method, as described in subsection 2.2.1.
- **Gaussian Otsu Threshold** Applies a Gaussian Blur filter before applying Otsu's Method.

Find contours

The contours function must accept the input image as a parameter and return a single contour for each image, where a single contour is represented by a list of points.

- **External Contours** This implementation finds the external contours of each object.
- **External Approximated Contours** This implementation finds the external contours of each object and uses a simple approximation to simplify the contour representation.

Find contour centers

A contour centers function must accept a list of contours and return a list of points, one for each contour provided as input.

- **Minimum Enclosing Circle** Finds the minimum enclosing circle for each of the provided contours.
- **Contour Moments Centroid** Computes the image moments of each contour, and calculates the centroid from these moments.

The default implementation uses *Simple Threshold*, *External Approximated Contours* and *Minimum Enclosing Circle*. The module also provides functionality for debugging and visualizing detections.

5.3.5. Tracking

Object tracking, specifically multiple-object-tracking, is by far the most complex task involved in positioning of the devices. The current implementation is kept as simple as possible, adequate for a proof-of-concept, but can be improved upon in many areas.

Object tracking is kept entirely separate from object detection, such that one can easily replace one without affecting the other. Tracking and related functionality are implemented in the `tracking` module, which defines a single public method.

The `perform_tracking()` method accepts 2 parameters:.

- **detected positions**: A list of lists of `Point`, representing all detected positions in all frames
- **timestamps**: The time stamps of all frames, relative to the first frame in the sequence

The current implementation of the tracking algorithm is relatively simple. It only incorporates information from the previous and current frames, and in practice functions much like an online tracking algorithm. It would, however, be easy to implement more advanced tracking if necessary as all information about all detections, both present, and future, are provided to the algorithm.

The algorithm iterates through all frames and processes each frame individually. Information from previous frames is carried forwards through mutable `Track` objects, which contain a complete history for each track, including the states (hidden or visible), positions and time stamps.

The frame processing algorithm starts by creating an empty list for all the devices that were occluded during this frame, and a complete list of previously detected devices that have yet to be mapped to a track for this frame. It also creates an empty map for all taken positions, used to enforce the hard constraint for detection-level exclusion as described in section 3.6. Lastly, it generates a K-Dimensional Tree (K-D tree), based on the SciKit[7] library, which is used to represent all detected positions. A k-d tree is a binary search tree that is optimized for searching in spaces with k dimensions and is used to perform queries such as finding the nearest or N-nearest co-ordinates for a given position. This dramatically enhances the performance of the algorithm compared to a naive nearest neighbor search, especially when the number of devices passes 1000.

Then it iterates through each existing track, processing one track at a time. Tracks

[7]http://scikit-learn.org

that have been discovered but have since been lost are initially skipped, in order to map currently visible devices first. This approach has shown a small but consistent positive effect on the results of the tracking algorithm in tests.

It begins by searching for all detected positions in the current frame within a small search radius of the device. The search radius, centered around predicted position of the device, scales linearly with the amount of time that has passed since the device was last seen, and is limited by a maximum search radius. This mechanism is in place to handle cases where the device is signaling ‘0‘ and then moved short distances. It allows the search to start with a minimal search radius, assuming participants will not move excessively during initialization as they will be asked to hold their devices still.

```
time_since_last_detection = timestamp - track.last_seen
if time_since_last_detection > MAX_HIDDEN_TIME:
    continue
search_distance = min(SEARCH_RADIUS_EXPANSION_RATE *
   time_since_last_detection, MAX_SEARCH_RADIUS)
distance, index = tree.query(track.position,
   distance_upper_bound=search_distance)
```

If no match is found in range, the device is considered hidden, and the algorithm continues to the next track. If one or more matches are found, it will attempt to greedily associate the track with the closest match. If the position has not yet been claimed by a track in this frame, the current track will claim the position, and the algorithm continues to the next track.

However, if the position is already claimed, it will determine which track has the closest predicted location and let it claim the position. It will then attempt to claim the second nearest position for the track that was rejected. If this position has also been claimed, it will recursively attempt to greedily assign each position to the nearest track, until all tracks have been assigned or a track has no unassigned positions within its search radius, whereby it is marked as hidden. This is a simple greedy approach, but is very efficient and provides adequate results.

Once all previously visible tracks have been processed, it will process all tracks that have previously been discovered but since been lost. These tracks will have an increased search radius compared to the tracks processed in the previous phase as they have been hidden for at least 1 frame. A new K-D tree is generated for the remaining, unclaimed positions, and the process described above is repeated.

Finally, any positions that have not been claimed by any existing tracks, visible or hidden, are considered new tracks and will be included in the next iteration.

Once all tracks and detected positions have been accounted for, the updated state is applied to the existing tracks, and new tracks are created. As each `Track` is updated or created the algorithm will determine whether it should attempt to decode a signal symbol from a track history. Based on the symbol rate and time stamp it is possible to determine when the signal for a given track is supposed to transition, and then use all samples between the last two transitions to perform a majority vote and detect the symbol. The detected symbol is then appended to any previous symbols on the same track.

The whole process is then repeated for all remaining frames. When all the frames have been processed, each `Track` has a complete history of its state, position, time stamps, and the detected sequence. The subsequent steps, which involves mapping the sequence to IP-addresses, have already been described in subsection 5.3.3.

5.3.6. Streaming

Once the initialization phase is complete, all detected devices will receive streaming data once an administrator begins streaming content. The system supports both images and videos in various formats and resolutions. The server will process the input and send data in a separate thread so that it does not interfere with the normal operation of the server.

Images

When streaming images it will first map the position of each detected device to a location on the input image, e.g., a device that was detected in the center of the capture will receive colors from the center of the input image. Once this mapping has been done, it will not remap any devices until the resolution of the input changes.

Next, it will iterate through all devices, mapping a color from the input image to each device, and finally broadcast the designated colors to all devices. The color for each device can be the value of a single pixel or the average of a region of pixels centered around the transposed location.

Stop must be used manually to hide the image once it has been broadcast.

Video

When streaming video, each frame will be extracted and processed as a single image, as described above. If no other content has been streamed, or the resolution is different from previous content, all devices will be transposed to the new resolution of the first frame. No processing is required for subsequent frames of the same resolution.

The FPS of the stream can be configured and can be set to either a manual value or based on the FPS of the input video. The stream will end once the video is finished streaming or stop is used.

5.4. Mobile client

The client is implemented as a native Android application. It is implemented using Kotlin[8], a relatively new language that was recently announced as being officially supported on Android.

5.4.1. Connection

After launching, the application will automatically attempt to connect to a server, which can be configured with a hostname and port in settings, until it is connected or manually disconnected. It uses a simple exponential back-off algorithm to avoid draining the battery and overloading the server in case there many clients attempting to connect at the same time.

Once connected, the device will send its current position to the server and wait for a response. The server can either reject the device, in which case it will disconnect, or accept the device.

If accepted, the application will adjust the screen to maximum brightness and display a plain, black surface. It will also prevent the screen from turning off while the app is running in the foreground, in order to make it easier for the detection algorithm to correctly detect the device during the sequencing phase.

If the client is disconnected, not rejected, at any time, it will immediately try to reconnect. If it is able to reconnect, it can continue to display incoming data if it had already been detected before the disconnect, given that it has the same Internet Protocol (IP) address as it had before it disconnected.

[8]https://kotlinlang.org/

5.4.2. Initialization

Once the device is ready, it will wait for data from the server according to the protocol as described in section 5.2. When the client receives a sequence from the server, it will set up an asynchronous task to display the sequence at the given FPS. Because Android, the operating system, takes some liberties in how and when it will render updates to the User Interface (UI), it is not possible to use the regular Application Programming Interface (API) to display this sequence. Android tries to batch screen updates together, and because the application is attempting to display several frames per second, it will lead to some frames being completely missed, while others are rendered out of sync.

To overcome this, all colors are being rendered using a 'SurfaceView', which provides lower level access to the rendering API. This allows the application to render frames at a near constant FPS. However, because it is not truly constant, a drift may occur during the display of the initial sequence. This drift is entirely random and may be caused by background tasks or garbage collection, and will lead to cumulative delays in all subsequent frames. To account for this drift, the sequencing task will attempt to adjust for any incorrect timings between two intermittent frames, such that the sequence as a whole is displayed using a constant FPS. That is, $requested\ fps \approx actual\ fps$ as the total duration of the sequence increases.

This allows the device to be detected correctly despite minor variances in the frame rate during the initial sequence, as long as the variance between any two frames is no longer than a maximum, defined as $max\ variance\ in\ ms \leq 1000/fps$.

Once the full sequence has been displayed, the screen will turn black to avoid interfering with any subsequent sequencing phases. This process can be repeated any number of times for any number of devices, and the device will wait for input from the server.

If the device was not successfully detected during the sequencing phases, it will be disconnected, reset the brightness and allow the device to turn off its display. If the device was successfully detected it will be waiting for further input from the server, and display any incoming data according to the protocol.

5.4.3. Streaming

During regular streaming mode, the device will display any data it receives instantly using the 'SurfaceView' API, and not attempt to correct for any timing issues. There will likely be many devices that are slightly out of sync by a few milliseconds, but given the environment and usage, the client has agreed that small discrepancies in the synchronization between devices are tolerable. The network overhead and amount work involved in attempting to synchronize all devices would not be feasible nor important to the intended use of the system.

5.5. Administrator client

The administrator interface comes in two variants; both implemented in Python version 3.6.

The first is a TCP/IP network client that uses a persistent connection. It requires the user to authorize once at the beginning of the session and lets the user send any command directly through the CLI after that point.

The other interface, named 'CWS-CLI', is a CLI tool that utilizes a simpler request-response model. It is implemented using python-fire[9] and python-requests[10]. The server was initially only implemented with support for a persistent connection client, but a request-response model was added to provide a better user experience, which is important to the project owner. Unlike the persistent client, the admin CLI provides complete documentation of all the available commands and arguments and is able to provide rich feedback to the user. It is also completely stateless and can be invoked directly from the command-line on any system with a python interpreter. It provides a user experience similar to that of any other tool that can be invoked from a command-line. It comes bundled as an installable package, and after installation, it can be invoked as 'cws-cli' followed by a command, or 'help' for help.

Each command results in two requests being sent to the server, as can be seen in Figure 5.5. The first request is always an 'auth' request, which is needed in order to authorize the user. Because the CLI terminates the connection between each command, this request has to be included with each command. The tool has configurable settings which let the user set a default IP, port, and password in order to avoid having to type these details for each command. If the authorization is successful, the CLI will send the actual command to the server and await a response. The server will always attempt to send a response to each request, but failing to do so, the CLI will terminate the connection after a short timeout.

[9]https://github.com/google/python-fire
[10]http://docs.python-requests.org/en/master/

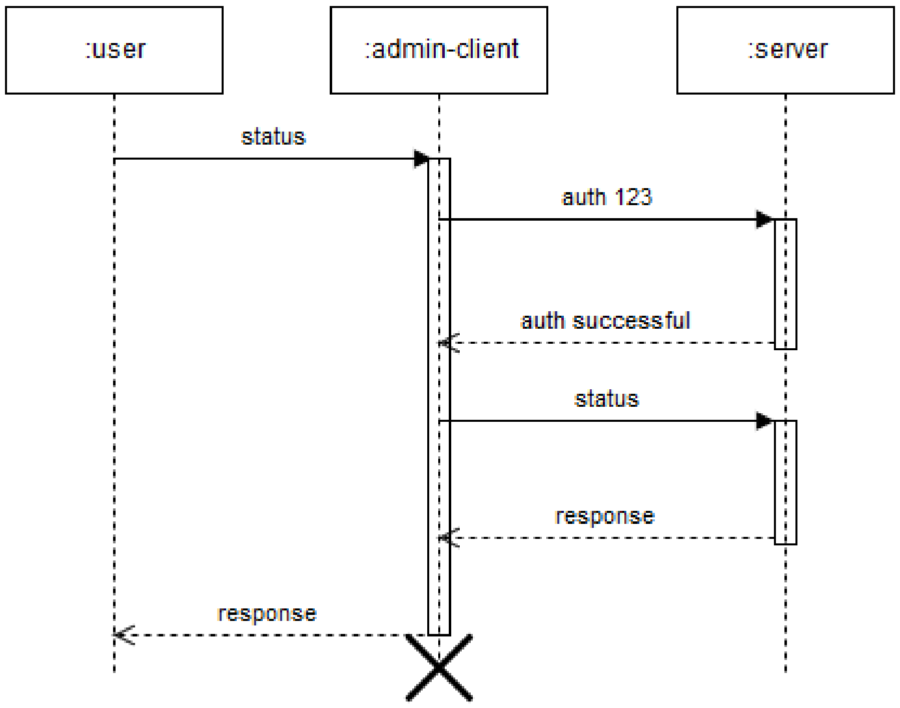


Figure 5.5.: CWS-CLI request-response model

5.6. Testing framework

The prototype comes with extensive support for programmatic and synthetic testing at various levels, including unit tests, integration tests, and system tests. This has proven invaluable during the research and development of this project as it has allowed rapid development, testing, and comparison of changes to any part of the system.

There are several components that can be used by themselves or in conjunction in order to isolate, mock and test specific parts of the system. The essential components are described in the next subsections. Together, these components provide a robust framework for testing and further development of the system.

5.6.1. Test video generator

The test video generator is the most important part of the test bed. It can create test videos that can be used as input to the system in a variety of ways, and it has an extensive set of features and settings which can be used to customize the generated videos.

It requires at least 2 arguments:

- **device_shape**: A tuple, indicating the number and shape of the devices, e.g. (20,20) for 20x20 devices
- **path**: A relative or absolute path, where the output file and related data will be stored

Using these parameters, the generator will produce a video with the indicated number of devices in a rectangle, broadcasting a randomly selected sequence for each device. The video, along with the sequences and timestamps for each frame, are then stored to disk at the provided location.

This video, along with the generated time stamps and actual device positions for each frame, can then be provided to the system in order to test the detection, tracking and mapping algorithms. However, in order to make the generator more useful and realistic, it also supports the following additional parameters:

- **sequences**: A list of sequences, one for each device. Default: randomly generated sequences.
- **fps**: The FPS of the recording. Default: 30
- **movement**: A value indicating how far the devices can move per frame. Default: 2
- **symbol_frequency**: Symbols per second. Default: 5
- **device_color_variation**: Allowed color variation for devices, 0-255. Default: 50
- **device_size_range**: Allowed size variation range in pixels. Default: (1,5)
- **output_resolution**: Resolution of generated video. Default: (1080, 720)
- **max_movement_distance**: Maximum allowed movement from initial position. Default: 20
- **max_start_delay**: Maximum start delay for devices to begin broadcasting, in frames. Default: 20

Figure 5.6, Figure 5.7, Figure 5.9 and Figure 5.11 show different variations of the produced test videos, and also display how the generator evolved from simple grids of devices to complex randomized positions and movements. Some of the devices in the figures are

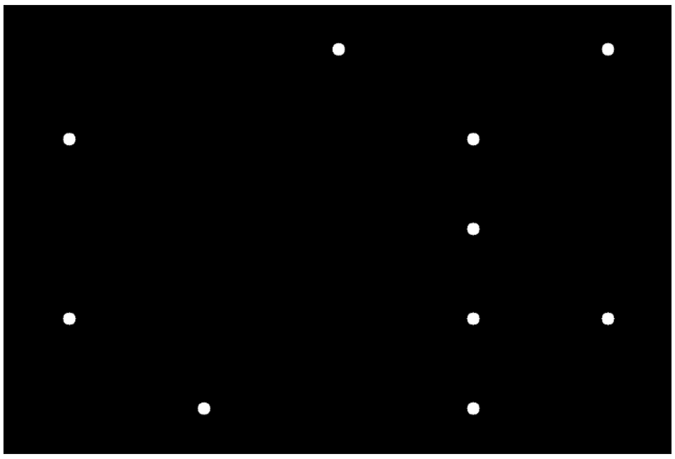


Figure 5.6.: Generated test video with 25 devices

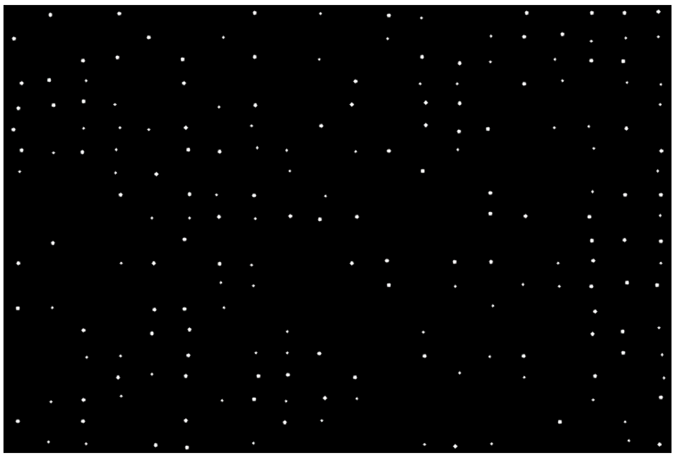


Figure 5.7.: Generated test video with 100 devices in square formation

not visible in the images, depending on the symbol they are signaling.

In combination, these parameters allow the generator to produce reasonably realistic captures. It is able to simulate random movement, changes in detected device colors due to external light sources and changes in device size due to partially occluded or tilted screens. It also simulates network latency with delayed starts as well as technical details such as capture fps, symbol frequency and resolution.

The resulting video is visually very similar to real-world videos of crowds with smartphones at venues after thresholding has been applied. Figure 5.8 and Figure 5.10 display how real images appear after thresholding has been applied, and demonstrates the similarity between the generated output and the real images. The overall pattern of the crowd is different, but the individual devices can be made to be quite similar, both in

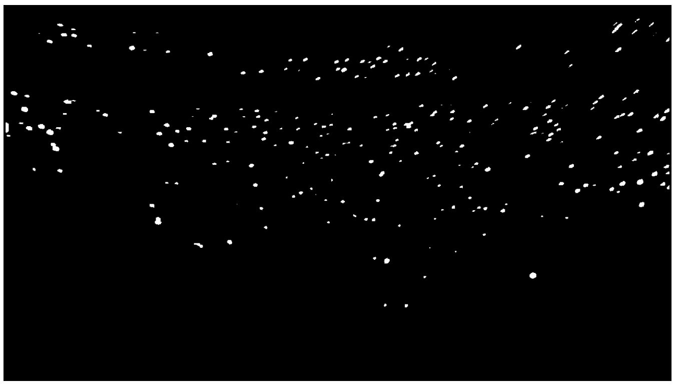


Figure 5.8.: Real image after thresholding, medium sized arena

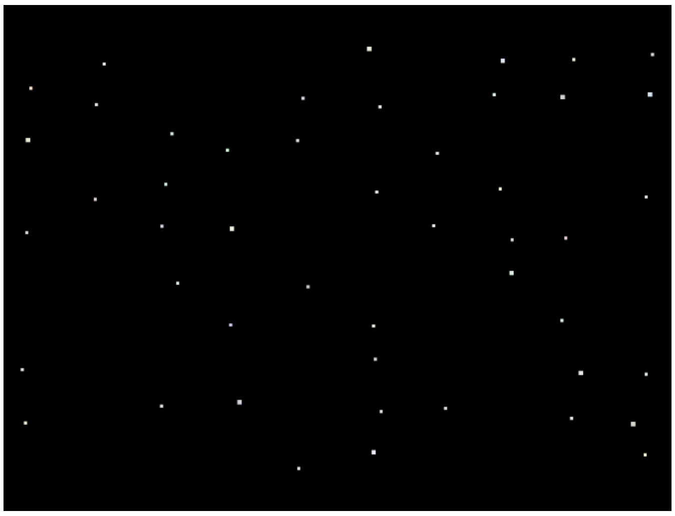


Figure 5.9.: Generated test video with 100 devices, random movement

appearance and behavior.

The video can then be used as input to the system, either directly or as input to the capture component. The video generator is also able to produce annotations that are compatible with Sloth[11], which is a well-known tool for reading and writing annotations. This enables it to be processed by a large number of existing tools and libraries.

[11]http://sloth.readthedocs.io/en/latest/

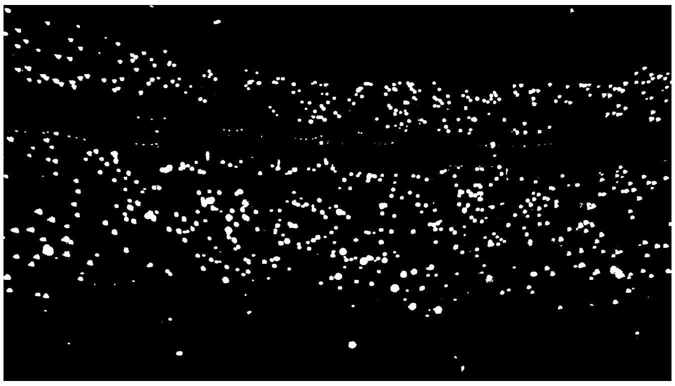


Figure 5.10.: Real image after thresholding, large arena

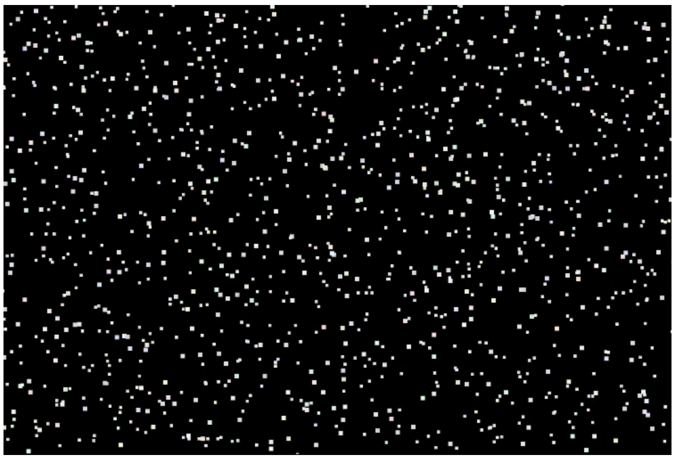


Figure 5.11.: Generated test video with 2500 devices, random movement

5.6.2. Simulated camera

The `CameraService` has been mocked to provide a video generated by the test generator instead of providing a real video stream. This allows seamless interaction with a simulated environment. The mock can be configured to provide a specific video or generate a new one. The generated video can be random, or more importantly, during system tests, it can hook into the sequence generation of the server and generate a video which simulates the exact sequences used internally by the server.

This allows easy and realistic simulation of any number of devices, as the system is completely unaware that the video is being generated. It can generate videos simulating thousands of devices in various configurations and can be used to test the detection, tracking and mapping components of the system.

5.6.3. Simulated network

`NetworkService` has been mocked in order to provide a simulated environment of connected devices. It takes two arguments, the number of clients that should be simulated and a callback for received data. The mock will then simulate the specified number of clients while the other parts of the system are completely unaware whether the connections are real or not. All simulated devices will be connected and accepted by default, and therefore will not send any further data to the server.

The `NetworkServiceMock` be used in conjunction with the `CameraServiceMock` in order to adequately simulate the server environment in a realistic manner. This setup will simulate a number of connections, while still allowing an admin to connect interact with the server. When an admin starts the initialization phase, the CameraService mock will intercept the request for the camera, and instead generate and return a video that simulates the sequences that were sent by the server. Once one or more devices have been successfully detected, the administrator can begin streaming content to the fake devices.

Streaming data coming from the server will be displayed in a window as a circle of the correct color. Each device will have it's own designated position on the screen, simulating a position in the crowd. This effectively shows how the stream would look through a camera from above if all participants held their devices pointing upwards. Figure 5.12 shows the output of streaming to 400 simulated clients in a 20x20 grid.

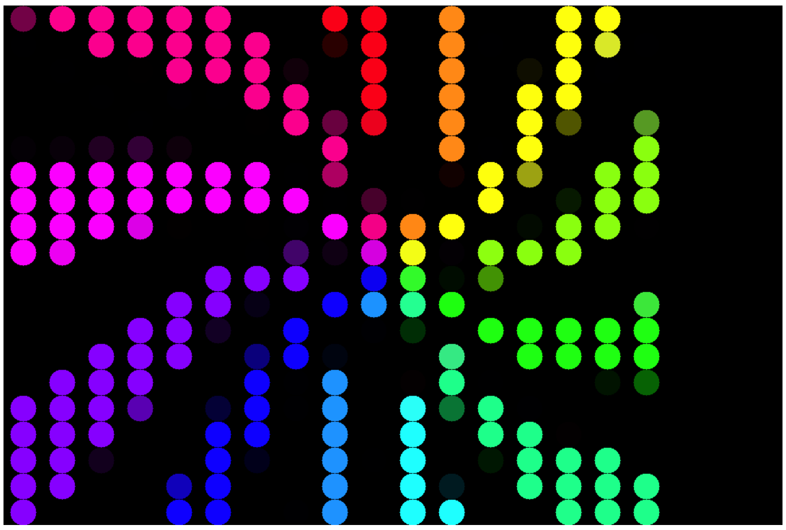


Figure 5.12.: Output using simulated clients.

5.6.4. Mobile test client

The mobile test client is functionally the same as a mobile device and connects through TCP/IP, but it can be run from a local terminal. It accepts input from the keyboard which is sent to the server, such as the POS command with GPS coordinates. It will also accept and display sequences and streamed content on the screen. Multiple of these can be scripted and run in parallel in order to set up and test real TCP/IP connections. It can also be combined with the CameraService mock in order to generate a test video based on the sequences sent to each client.

6. Experimentation and Testing

This chapter describes a variety of experiments and tests that have been conducted during the project and discusses relevant results and discoveries. These experiments and tests range from fully simulated and synthetic tests, with thousands of devices, to real-world experiments with a small number of devices. It was not possible to conduct a large-scale real-world test without assistance from an event organizer, which was not possible to arrange due to time restrictions.

Based on the results from the simulated tests and initial real-world experiments, increasing the number of devices beyond two alone has no impact on the performance of the algorithm. Seemingly, the factor that most impacts the performance is whether the devices are close or overlapping, and this effect can be tested using only two devices.

Also, given the similarity of the generated video and the real recordings after the threshold has been applied, real experiments have focused on particular scenarios using only a few devices, while large-scale tests have been conducted using generated input.

A number of automated tests have been implemented in order to detect possible issues, bugs, and flaws during the development of the system. Another set of automated tests have been developed to test, measure and track the performance of the system, both in terms of detection results, tracking results and the execution speed.

These measurements have then been used to compare the performance of the system using different parameters or different algorithms altogether. This section covers some of the most significant tests and provides a discussion of their results.

Section 6.1 provides an overview of some of the most common methods for evaluating MOT systems. Next, section 6.2 and section 6.3 describes and discusses the tests and results for device detection and device tracking, respectively. Then, section 6.4 covers the tests and experiments that were conducted using a real camera, real participants, and real devices. Finally, section 6.5 briefly describes various other tests that have been implemented in order to verify the functionality of the system.

6.1. Evaluation

Metrics and data sets are required in order to evaluate the approach and its performance quantitatively. This is particularly important in order to measure the influence of various components and their parameters, and how they affect the performance of the system. This helps guide towards and design a better system. However, performance evaluation for multiple object tracking is not an easy task.

6.1.1. Metrics

The choice of evaluation metrics is crucial, as they provide the means for the comparison. The current approach utilizes tracking-by-detection, so some of the metrics should evaluate detection performance in addition to the metrics for tracking performance. The metrics can then be categorized into metrics for detection and metrics for tracking. This section provides a brief overview of some of the most common metrics and measures for multiple object tracking. It is important to note the difference between "precision" as a metric for calculating accuracy, and the category of metrics called "precision. The former relates to the ratio between true positives and total predictions, while the latter relates to the physical position of the detected object.

Detection

Detection metrics are commonly grouped further into metrics for accuracy and metrics for precision.

Accuracy The well-known Precision and Recall metrics are commonly used, along with False Alarms per Frame (FAF). Another popular metric, called Multiple Object Detection Accuracy (MODA), considers the relative number of false positives and miss detections and is the most comprehensive metric for accuracy [28].

Precision Multiple Object Detection Precision (MODP) measures the quality of alignment between predicted detections and the ground truths and is the most commonly used metric for precision. [28].

Tracking

Like the detection metrics, tracking metrics are commonly divided into groups based on what is being measured.

Accuracy These metrics measure how accurately a method can track its targets. Multiple Object Tracking Accuracy (MOTA) is by far the most commonly used metric, and combines the rate of false positives, false negatives, and mismatches into a single number [6]. It provides a fairly reasonable measure of the overall tracking performance but has received some criticism. Another commonly used metric is ID switches, which counts how many times the algorithm switches the associated ID for a track [75].

Precision These metrics describe how precisely the tracking algorithm can track the objects, either by bounding box overlap or distance. Common measures are Multiple Object Tracking Precision (MOTP) [6], Tracking Distance Error (TDE) [30] and Optimal Subpattern Assignment (OSPA) [52].

Completeness Completeness measures how many of the ground truth trajectories are tracked, and includes the number of mostly tracked, partly tracked and mostly lost trajectories. Mostly tracked are defined as tracks that are covered for at least 80% of their lifetime, while mostly lost are tracks that are covered for maximum 20% of their lifetime. Everything in between is counted as partly tracked.

Robustness Metrics for robustness measure how well the algorithm can recover from occlusion. Only two such metrics, Recover from Short-Term Occlusion and Recover from Long-term Occlusion, have been presented [61].

6.1.2. Data set

Consistent input is important to be able to compare different tracking components and their parameters. Because there are no existing data sets with devices emitting Crowdstream sequences, a data set had to be created.

Because the data-set had to be created from the ground up during this project, it has not been possible to accurately annotate the data-set with ground truths and some of the other data that is necessary to compute many of the most popular measures. Most of the metrics can only be applied to the generated tests, as there is not enough data available to compute any valuable metrics for the other tests in the data set.

An overview of the generated dataset can be seen in Table 6.1. *Ground Truth* (GT) indicates whether the data includes the absolute truth regarding the position and tracks, while *Sequence Truth* (ST) refers to if the actual sequences that were broadcast are available.

As can be seen in Table 6.1, the generated tests has the most comprehensive annotations to compare against and includes the actual position of all devices in each frame, as well as their trajectories throughout the recording.

The recordings from real life tests and experiments are annotated with the actual sequences that were broadcast during the test but has no data on the positions or tracks of the recorded devices. The real videos and images have no attached metadata, and can only be evaluated visually.

	GT	**ST**
Real world images from people holding their devices at events		
Real world videos of people holding their devices at events		
Single images from generated videos with varying parameters	✓	✓
Generated videos with varying parameters	✓	✓
Single images from real tests with Crowdstream		✓
Captured recordings from real tests with Crowdstream		✓

Table 6.1.: Crowdstream data-set, with Ground Truth (GT) and Sequence Truth (ST)

6.2. Detection

A number of detection tests have been implemented, and these are used to track and compare the performance of different algorithms and parameters on a set of images. There are mainly four types of images that have been tested:

- Real world photos of people using their smartphones at events
- Single frames from real-world recordings at events
- Single frames from video captured with Crowdstream during tests

- Single frames from generated videos

Each image has been tested against different combinations of detection algorithms and parameters, as described in subsection 4.2.1, which illustrates the test and comparison of 3 common thresholding algorithms.

6.2.1. Metrics

Accuracy Because the detection algorithm utilizes a manual global threshold to detect objects, it is not meaningful to compute any accuracy metrics based on the generated tests. The reason is that any pixel with an intensity above the threshold is detected as an object, and any value below the threshold is not. And since the video would be generated, the resulting metric would simply indicate where the generator used color values below or above the threshold, which has no value in itself.

Accuracy metrics would be more valuable if they were computed for the real images but requires the images to be manually annotated with ground truths first. Due to time restrictions and other priorities, this has not been done. Therefore, no accuracy metrics have been used for detection.

Precision The algorithm used in Crowdstream does not provide a bounding box for the detected object, but rather the centroid of the bounding box. Therefore, it is not possible to use any of the most commonly used metrics that are based on the intersection of the bounding box. Instead, precision is measured as the Euclidean distance from the detected centroid to the actual centroid of the device.

Computational performance The computational performance is measured by the number of frames processed per second (Hz). The hardware used for this test is an Intel Core i7-7700K at 4.2GHz with 32GB of RAM.

6.2.2. Tests

Real images

The detection algorithm has been tested on a large number of real images, such as the image in Figure 6.1. The range of images vary in many ways, such as:

- Number of devices
- Distance between camera and devices
- Light conditions
- Colors on the screens of the devices
- Distance between devices

As can be seen in Figure 6.1, the detection algorithm is reasonably accurate and precise. The factor that has the most significant on the detection algorithm is the amount of surrounding light. Some light can be accounted for by tweaking the threshold value, but the detection algorithm does not work well in broad daylight.

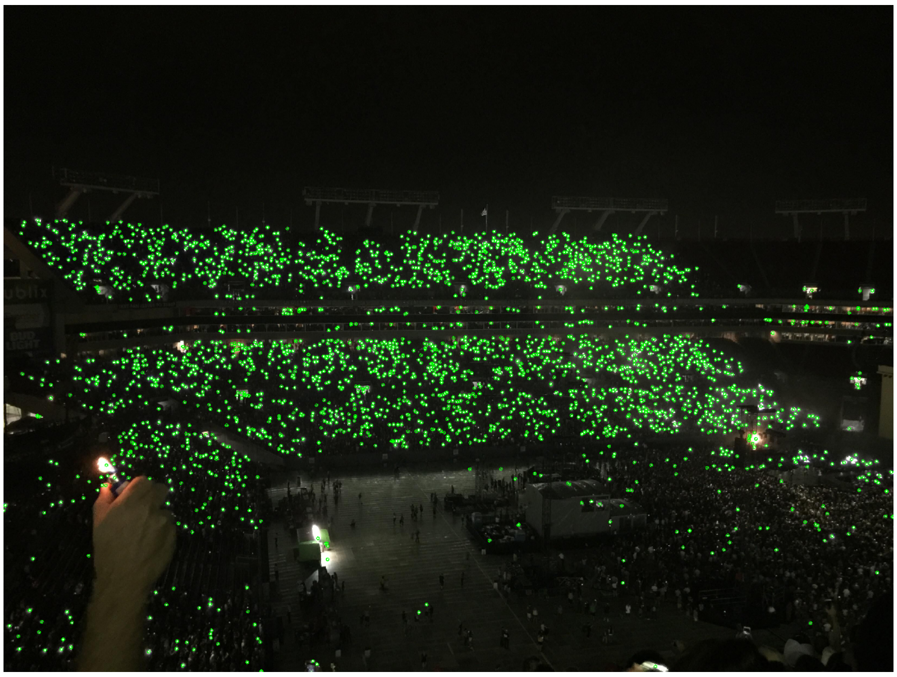


Figure 6.1.: Detection at long range with many devices

Captured images

The detection algorithm has been tested on a number of frames from recordings made during real tests and experiments. As with the tests in subsection 6.2.2, these tests indicate that the algorithm is susceptible to surrounding light, and particularly other direct sources of light or high-intensity reflections. This, however, is expected, s algorithm does not utilize any other information than the color intensity to detect objects. The image in Figure 6.2 is an example of an image that has been used for these types of tests.

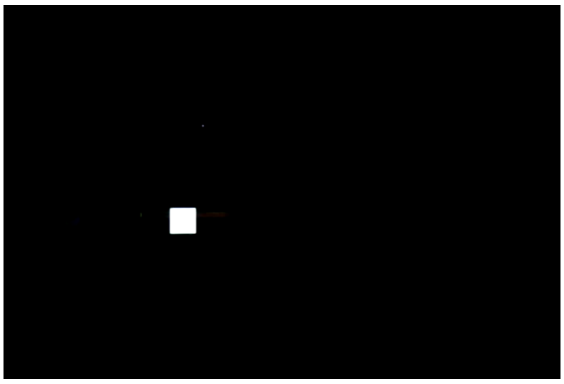


Figure 6.2.: Detection and tracking of real device at close range in low light environment

Generated images

Generated images are used for testing the precision of the algorithm. The precision error for a single object is calculated as the Euclidean distance between the position of an actual object and the nearest detected position.

$$\mathbf{error_i} = \|d(\mathbf{gt_i}, \mathbf{p_i})\| = e(i)$$

where g is the ground truth position and p is the detected position for an object i.

The average error for a frame f is then computed

$$\mathbf{avgerror_f} = \frac{\sum_{i=0}^{nf} e(i)}{nf} = fe(f)$$

where nf is the total number of detections in the frame.

Finally, the average error over a collection of images c is then calculated:

$$\textbf{avg error for collection} = \frac{\sum_{i=0}^{nc} fe(i)}{nc}$$

where nc is the total number of frames in the collection.

Figure 6.3 shows the results of plotting the average error for a collection of images against the average size of the devices in the collection.

In practice, the size of actual devices is usually in the range of five to 15 pixels. It shows that error distance increases as the size of the devices increases when testing with 100 detectable objects.

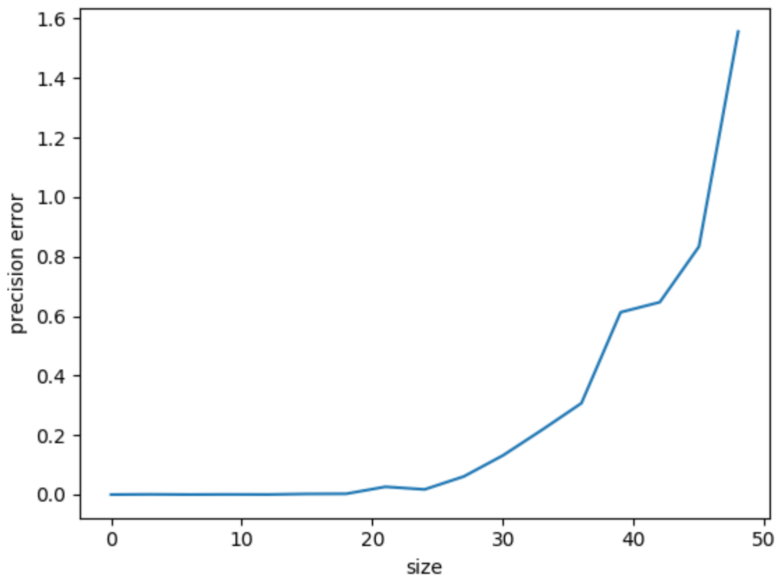


Figure 6.3.: Distance from actual object to detected object (precision error) plotted against the object size in pixels (size), with 100 objects.

Sequences of generated images have also been used to calculate the precision of the detection algorithm after tracking.

This is done by comparing the position of actual objects to the detected position *for the actual track*, instead of the nearest detected object.

Also, instead of summing the error for all detections in a frame and dividing by the number of frames, we sum the error for all detections for *a given track* and divide by the number of detections for the specific track.

That is, for a track t, the total error is calculated as

$$\mathbf{error_t} = \frac{\sum_{i=0}^{nt} e(i)}{nt} = te(t)$$

where nt is the total number of detections for the track. Next, the total error for a sequence s is calculated by summing the error for all tracks and dividing by the total number of detected tracks. When ns is the total number of tracks in a sequence, the average error of all tracks in the sequence becomes

$$\textbf{avgerror for sequence} = \frac{\sum_{i=0}^{ns} te(i)}{ns}$$

Figure 6.4 shows the average error for a sequence plotted against the amount of random movement in pixels. All sequences have 400 objects at a random size between five and ten pixels. In practice, the amount of random movement is usually in the range between ten and 30 pixels for the conducted experiments.

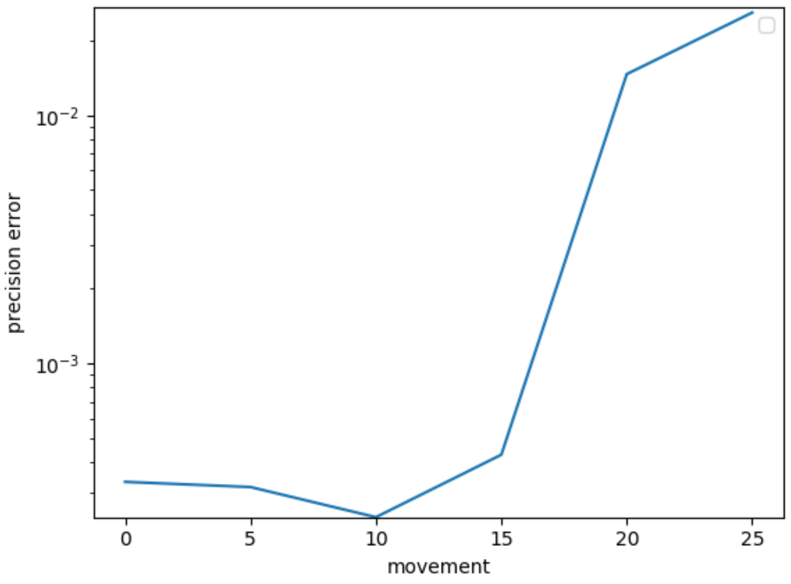


Figure 6.4.: Euclidean distance (precision error) plotted against the amount of movement. (400 objects with a random size between five and ten.)

Performance

Detection performance has been tested by repeatedly performing detections on a sequence of 500 frames using an ever decreasing threshold value in steps of 1, such that the number of detections increases for each consecutive run. In Figure 6.5, the average number of frames processed per second (HZ) has been plotted against the average number of detections in each frame.

As can be seen from the figure, the performance of the algorithm decreases as the number of objects in each frame increases. However, it is still able to process well over 1000 frames per second with 250 devices in each frame on average.

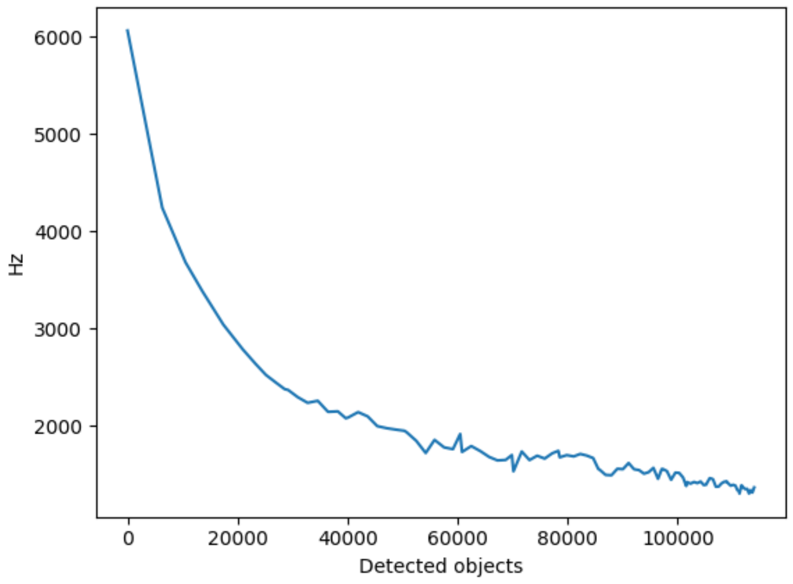


Figure 6.5.: Frames processed per second (Hz) plotted against avg. number of devices in each frame.

6.2.3. Discussion

The results from the detection tests indicate that a manual threshold works well, but requires some adjustments according to the environment. A reasonable default value is 200 for a medium sized venue with some stage lighting. If the venue is large or has a lot of indoor lighting, the value must likely be reduced in order to avoid too many false negatives. This will however also increase the number of false positives, and very bright venues will have a significant negative impact on the performance of the detection algorithm. The threshold value should be increased if the venue is dark, as this provides a significant improvement to the number of false negatives and false positives.

The computational performance of the detection algorithm is directly impacted by the amount of participating devices but is able to process over 1000 frames per second with 250 devices. This should be more than enough for any scenario the system is intended for.

The detection algorithm is not directly impacted by the amount of participating devices in terms of detection performance. However, it is not able to separate two devices that are very close, which is a likely side effect when introducing more devices and will detect only one of those devices.

The precision of the algorithm is excellent as long as it is able to separate each device from the others. If it can distinguish between objects correctly, the distance between the actual and detected object is marginal. However, when multiple objects are detected as a single object, the precision decreases. This effect scales with the size of the detected objects, because the size affects the distance from the centroid of the object to the centroid of the union of two objects.

6.3. Tracking

The tracking tests consist of a variety of different videos, which can be categorized as follows:

- Real world videos of people using their smartphones at events
- Generated videos with varying amount of devices and parameters
- Real captures of mobile devices executing the initialization sequence

The real world videos do not have a ground truth to compare the results against, so the tests have to be examined and evaluated visually. The tracking tests support comprehensive visual debugging tools, which will display all new, lost and currently tracked devices, as well as many other features such as the current search radius.

The generated videos and real captures have been stored along with the original sequences that were used at the time, as well as time stamps. In addition, the generated videos include the actual positions of all devices in all frames, which can be compared against the detected and associated positions.

6.3.1. Metrics

Accuracy Unlike the accuracy metrics for detection, accuracy metrics for tracking can still provide value even if they are collected from generated tests. Because the tracking algorithm is more complex than the detection algorithm, the randomness introduced by the video generator will significantly impact the result.

MOTA is by far the most popular metric for accuracy for multiple object tracking approaches. MOTA is calculated as

$$MOTA = 1 - \frac{\sum_t (m_t + fp_t + mme_t)}{\sum_t g_t}$$

where m_t, fp_t ,and mme_t are the number of misses, of false positives, and of mismatches, respectively, for time t. It can be seen as a combination of three ratios; the ratio of misses in the sequence, the ratio of false positives and the ratio of mismatches [6]. Note that MOTA can also be negative in cases where the number of errors made by the tracker exceeds the number of all objects in the scene.

Metrics for track precision and recall are also calculated as follows.

Precision measures how many of the hypothesis tracks are correct, and is calculated as

$$Precision = \frac{\#GT}{\#HT}$$

where $\#GT$ is the number of ground-truth tracks and $\#HT$ is the number of hypothesis tracks.

Recall measures how many of the ground-truth tracks that were covered by a hypothesis, and is calculated as

$$Recall = \frac{\#TGT}{\#GT}$$

where $\#TGT$ is the number of correctly tracked ground truths and $\#GT$ is the number of ground-truths.

Precision MOTP is the most common metric for tracking precision, and measures the overlap in the estimated position for matched object-hypothesis, averaged by the total amount of matches made.

"It shows the ability of the tracker to estimate precise object positions, independent of its skill at recognizing object configurations, keeping consistent trajectories and so forth." [6]

MOTP is calculated as follows:

$$MOTP = \frac{\sum_{i,t} d_i^t}{\sum_t c_t}$$

where d_i^t is the overlap in positions between a hypothesis i and its corresponding ground-truth at time t, and c_t is the number of matches at time t.

Completeness Measures for completeness have not been included in these tests, due to how the algorithm works. All objects that are detected are also covered by tracks, which means that the metrics would merely be another measure for detection, which is already covered.

Robustness Measures for robustness have not been included in these tests. This is due to the unique behavior related to occlusion for this algorithm, such that existing measures are not applicable.

Computational performance The computational performance is measured as the number of frames processed per second (Hz), excluding the detector. The hardware used for this test is an Intel Core i7-7700K at 4.2GHz with 32GB of RAM.

6.3.2. Tests

Real video

The tracking algorithm is able to track devices in recordings from real events reliably. It is not possible to utilize any metrics or measures for these types of videos without manually creating ground truths, so these tests have been evaluated visually. Figure 6.6 shows an example of the tracking algorithm as it is tracking devices in a video. Cyan circles indicate active tracks while green circled indicate newly discovered tracks. Red circles indicate lost tracks.

As can be seen in Figure 6.6, the algorithm is able to track the devices reasonably accurately. Temporarily occluded objects will be re-associated with its existing track if it re-appears within the search radius and also within a given time frame for the track. If it does not, it is considered lost. This might not make sense when doing general tracking such as in the figure, but for Crowdstream it is not possible to associate the device with a sequence if it has been occluded during for more than a defined number of unit intervals. Therefore, such tracks are discarded immediately when detected.

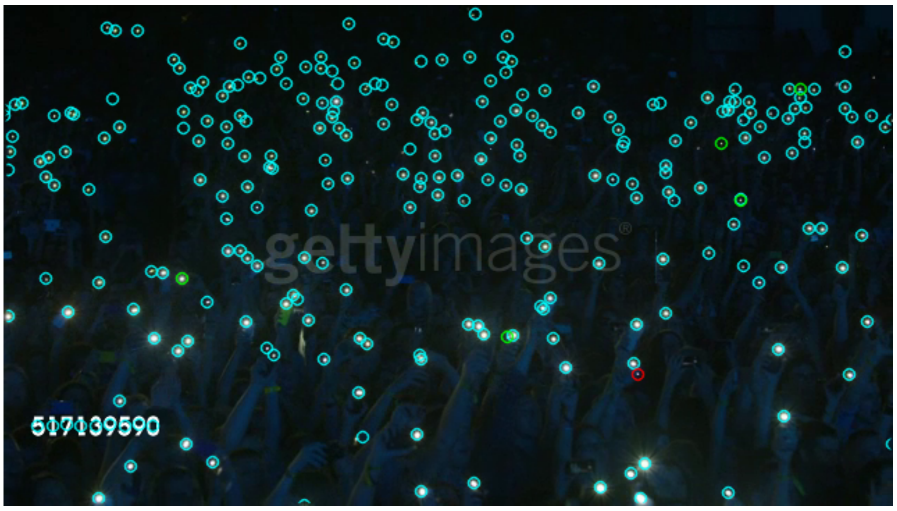


Figure 6.6.: Tracking in real video recording. Copyright belongs to GettyImages.

Captured video

The tests using captured video as input are actually re-runs of previously conducted experiments and real-world tests, where the recording and associated metadata has been stored for reuse. The test framework makes it possible to recreate and replay the test scenario exactly as it was executed at the time it was recorded. This data includes the time stamp of each frame as well as the actual sequences that were generated and broadcast.

Figure 6.2 is a frame extracted from a video that has been used in these tests. These videos could be manually inspected frame by frame in order to annotate them with additional data such as the actual device positions and symbols, in order to collect and calculate more advanced metrics. This has not been possible due to time restrictions, and it is therefore not possible to calculate any meaningful metrics for these tests.

However, they still provide value as these tests can assert whether the algorithm is able to correctly track and position the devices after making changes to the implementation.

Generated video

As previously mentioned, generated videos are annotated with ground truths provided by the video generator. This enables various metrics to be calculated, which can be seen in Table 6.2.

The tests have been generated with moderate movement between 0 and 3 pixels per frame and a maximum of 20 pixels from the origin point. This allows some overlap of devices as the number of devices passes approximately 200 in total. The tests also have a randomized sequence start delay for each device of one second, and a random color intensity between 190 and 250. The threshold is 200, such that some false negatives are produced in order to more closely simulate a real scenario.

The columns in the table are as follows; Number of devices, ground-truths, false positives, misses, mismatches, MOTA, correspondences, overlap, MOTP, precision and recall.

	GT	FP	Miss	MM	MOTA	Corr	Overlap	MOTP	Prec	Rec
25	2472	229	91	87	0.83	2381	2336	0.98	0.45	1
100	9780	870	341	491	0.82	9439	8955	0.94	0.56	1
225	21864	2367	925	1092	0.79	20939	19675	0.93	0.51	1
400	39378	4109	1866	1889	0.80	37512	35261	0.93	0.54	1
625	61692	7096	3933	3087	0.77	57759	54083	0.93	0.53	1
900	88782	11385	8779	4535	0.72	800003	74626	0.93	0.57	1

Table 6.2.: Tracking results for generated tests

The algorithm produces a lot of false positives, which significantly impacts the precision. However, false positives are pruned when the tracks are compared against valid sequences, and therefore has almost no impact on the performance of the system. The most important metrics are recall and the number of mismatches. MOTA is less valuable as it includes the number of false positives, which are reliably pruned at a later stage.

The results for MOTP indicate that the algorithm detects almost the precise location of the objects, which is expected due to the nature of the threshold detection approach.

Performance

The computational performance of the tracking algorithm has been tested by executing the tracking algorithm repeatedly on a single video, while decreasing the detection threshold value by five between each run. The video has been generated such that more devices are detected for each decrease in the threshold value. The performance timing of algorithm does not include the detection phase. The results can be seen in Figure 6.7.

The tracker generally performs well terms of computational efficiency and is able to process well over 100 frames per second while tracking over 700 targets. The performance drops drastically from 0 to 100 devices, which is likely due to the increased overhead

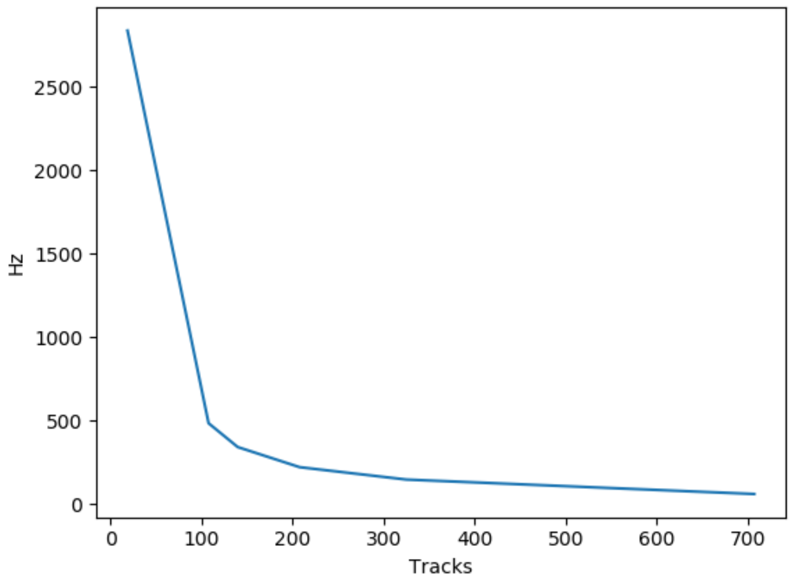


Figure 6.7.: Frames processed per second (Hz) plotted against the number of tracks

of creating the K-D trees without performing a significant amount of look-ups for each tree.

However, as the number of tracks increases further this initial overhead is marginalized by the time spent on processing and look-ups of the nearest neighbors, such that additional tracks have less impact on the overall performance. Without a K-D tree or similar implementation, the processing time would scale exponentially with the number of tracks.

6.3.3. Discussion

The results of the tracking tests indicate that, besides the performance of the detection algorithm, the amount of movement has the most significant impact on the ability to correctly track devices.

This is closely related to the tracking parameters for *max search radius* and *search radius expansion rate*, which control the search radius of the algorithm as it attempts to relocate a track after it has been occluded. As the average amount of movement increases, the values of these parameters must increase in order to allow the algorithm to associate the tracks once the devices re-appear correctly. However, doing so also increases the chances that a track will incorrectly be associated with another nearby device.

Values for these parameters must be carefully selected such that it minimizes the amount of incorrectly associated devices, while still allowing some random movement. Setting the value too low is preferable to setting it too high, as that will produce only a few false negatives for devices that are moving excessively, without affecting other devices. Setting the value too high will cause incorrect associations between all nearby devices and can cause all devices to be tracked incorrectly.

The tracking algorithm works well when there is little to moderate movement and not too many overlapping devices. It is often able to correctly track all devices in generated tests with roughly 1000 devices and moderate movement.

It is also able to correctly track and associate devices in real-world tests, which can be seen in Figure 6.8. This is a very bright environment which includes false positives, but it is still able to track both devices correctly.

The precision of the tracking algorithm depends on the results of object detection. The amount of movement, which can cause devices to overlap, and the size of the detected objects, are the two most significant factors for precision.

6.4. Real world experiments

6.4.1. Setup

The test plan in appendix A describes a number of tests that were planned. Only the tests listed under *Device Positioning* have been conducted. The client pointed out that usability and performance are not important, and these tests were therefore not prioritized.

A number of small-scale tests and experiments were executed in order to test the device positioning functionality, using combinations of the following:

- **Stationary devices** The devices are statically placed near the ground
- **Low movement** The devices are held and moved slowly within a range of 1 meter
- **Moderate movement** The devices are held and moved moderately fast within a range of 1 meter
- **High movement** Devices are held and moved fast within a full arms length, roughly 2 meters

- **Low light conditions** No direct light sources or sunlight
- **Moderate light conditions** Some sunlight, no direct light
- **High light conditions** Daylight and direct light from roof lighting
- **Short distance** Devices placed approximately 1 meter from the camera
- **Medium distance** Devices placed approximately 5 meters from the camera
- **Long distance** Devices placed approximately 10 meters from the camera
- **Partial short-term obstruction** Device is partially obstructed for 0.5 seconds
- **Complete short-term obstruction** Device is completely obstructed for 0.5 seconds
- **Partial long-term obstruction** Device is partially obstructed for 2 seconds
- **Complete long-term obstruction** Device is completely obstructed for 2 seconds

The tracking, detection and system parameters have been constant for all tests.

6.4.2. Metrics

In order to measure the effects the various parameters have on the algorithm, some key metrics have been chosen for evaluation. These experiments are not annotated on a frame-per-frame basis, so metrics for detections in a frame cannot be utilized. Therefore, all metrics described here relate to successfully mapped tracks.

- **True Positives** The total number of True Positive (TP)
- **False Positives** The total number of False Positive (FP)
- **False Negatives** The total number of False Negative (FN)
- **Precision** Ratio of true positives to number of detected tracks. $(TP/(TP+FP))$
- **Recall** Ratio of correctly detected tracks to total number of ground truth tracks. (TP/GT)

6.4.3. Discussion

The results of the tests are available in appendix B. The results of the obstruction tests were all negative, and are therefore not included in the table. Because the devices are signaling roughly five signals per second, any obstruction that lasts as long or longer than the unit interval of 0.2 seconds will cause the algorithm to fail.

There are however some exciting results from the other tests that could help guide further development. First of all, it is worth noting how *Precision* has little or no impact on the overall performance of the system. Intuitively this makes sense, because false-positive tracks are not successfully mapped to devices because the resulting sequence is not recognized.

Recall carries more value, as the system must be able to track the device in order to attempt to map the sequence.

The system performs well with no or low movement, as well as low and moderate light conditions. However, bright light at close range with low movement causes the algorithm to fail. It was discovered that the detection algorithm is very susceptible to reflections of light at close range, especially on devices where the frame is white or metal.

This is not an issue as the distance increases because the intensity of the reflection is reduced, and the frame is no longer detected as an object. This effect also has no impact during stationary tests, because there are no ID switches since the device itself is always its nearest neighbor. But once movement is introduced, there are ID switches between the actual device and the reflection from its frame, causing the algorithm to fail.

Depending on the intensity of the reflection, the issue can be mitigated by increasing the threshold, as long as the intensity of the screen is higher than that of the reflection by some margin.

Figure 6.8 shows that the detection algorithm is able to detect the devices during the initialization phase in very bright conditions with no movement, but also includes some false positives on the frame of one device. Once movement is introduced, this causes the algorithm to fail for that particular device.

This effect is magnified by increased movement. Moderate movement combined with bright conditions also causes issues, because the detection algorithm detects reflections and surrounding items as objects, causing ID switches between them and the device. The system fails entirely during high movement as it was not configured to handle such cases. This could be remedied by increasing the nearest neighbor search distance, but that would introduce other errors in practical use where there are more devices and less distance between them.

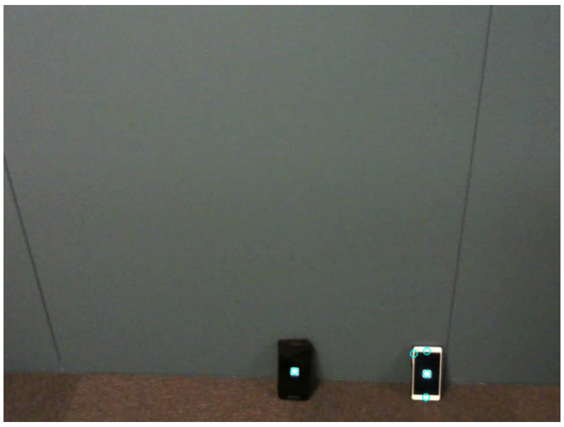


Figure 6.8.: Detection and tracking using real devices in bright environment.

6.5. Other tests

6.5.1. Positioning and streaming

Because positioning depends on the sequence being displayed, it is not possible to use real-world videos from other events to test this part of the system. The tests, therefore, rely solely on a set of generated videos and recordings of actual devices during the initialization phase.

The tests also assert that the devices are mapped to the correct positions, and that the mapping from the input content onto each device is correct.

Once devices have been detected and correctly tracked, it is relatively trivial to map the IP-address of a device to a relative position in the venue. This algorithm works well and is always able to associate the IP with a location if the device has been tracked correctly. It is also able to map the input image or video to the correct devices in the venue, and correctly handles differences in resolution.

An example of such tests asserts that the colors from a low-resolution image, shown in Figure 6.9, maps correctly to a set of devices that have been generated, recorded and processed using a resolution of 1080x720 pixels.

6.5.2. System and integration tests

There are no completely automated system tests, but through the use of the mock classes described above it is possible to start and run a real server in a completely or partly

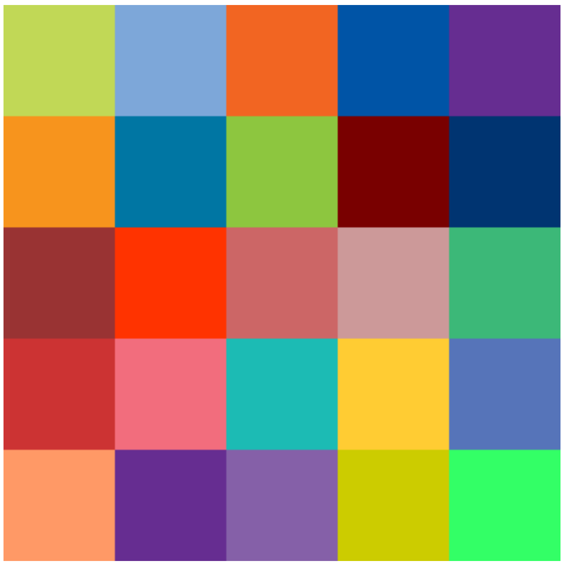


Figure 6.9.: 400x400 pixel image used to verify that colors are mapped correctly to devices.

simulated environment. This allows complete testing of the protocol, the system and the integration between the admin client, the server, and mobile clients.

7. Conclusions

This chapter provides a brief summary of the work and results of this project. First, section 7.1 provides a brief summary of the background, research, and objectives of the project. Next, section 7.2 presents the most significant results, and discusses them in relation to the research questions defined in section 1.2. Then, section 7.3 contains an evaluation of the research and project as a whole, as well as an evaluation of some of the decisions that were made and approaches that were used. Finally, section 7.4 provides a discussion of the results and contributions of the project in a broader perspective.

7.1. Summary

The background for the research and system presented in this document was an inquiry from Zedge, a company specializing in mobile applications. Over the recent years, light-emitting wristbands and other wearable items have become increasingly popular at concerts and other large events, indicating that there is a significant market for such technologies.

Zedge wanted to know whether it was feasible to utilize smartphone devices to add positional awareness to such systems, in order to create better and more sophisticated light shows than what is currently possible with existing technologies. Specifically, they wanted to find out if a solution that utilizes a simple camera in some way was a feasible approach to achieve adequate positional precision.

A review of existing literature and technologies found that there are no existing methods or research on attempting to position devices of large crowds using a camera. There exists a large body of research on indoor positioning systems in general, and the field is still actively researched. There is also a large variety of existing technologies for this purpose, but no single solution works well under all circumstances.

There is, however, one particularly important issue with all existing research and technologies, which is the fact that all current solutions are highly susceptible to interference from the human body. This poses a significant problem for the scenario described by Zedge as this will by its definition include a large number of people, and renders the existing solutions unusable without modification.

Searching for existing solutions, regardless of technology, revealed a few systems that attempt to achieve a similar result to what was described by the client. However, only two solutions provide positional awareness, and both are very limited. One only does so under particular and limited circumstances and requires manual input of position, the other requires significant setup costs and only provides coarse segmentation of sections.

7.2. Results

This section presents a brief summary of the objectives and the work that has been done. It also discusses the results in the context of the research questions defined in section 1.2.

7.2.1. Objective

The objective of this research was to determine whether it was at all possible to locate devices at an indoor arena using a solution based on a stage-mounted camera, and if the solution has the adequate precision for it to be used to orchestrate light shows using the devices. The scenario described by the client was defined in section 1.2 as:

- Indoor concert hall or arena
- Crowd of 100 - 2500 people
- Low or no-light conditions
- Stage-mounted camera
- The participants have the required app pre-installed
- The participants are willing to participate in a setup procedure for a short duration, announced over speakers
- One-time positioning is adequate, and the ability to continuously track locations is not required

Furthermore, the following research questions were defined:

- RQ-1: Is it possible to locate devices at an indoor arena using a stage-mounted camera, for the purpose of displaying light-shows across the devices?

- RQ-1.1: Can a camera and a unique discrete-time light signal sent from each device be utilized to uniquely identify the position of the devices under these conditions?
- RQ-1.2: How does the camera hardware specifications affect the performance of the system, and what are the minimum requirements for the camera?
- RQ-2: Which methods and techniques are most appropriate for object detection and object tracking under these conditions?

7.2.2. Research questions

RQ1

The client has stated that they are very satisfied with the research, and especially the resulting outcome. They were hoping to gain some insight into whether the camera-based approach was feasible and if so – how. This research question has been covered in-depth in chapter 4, which provides a detailed description of one possible approach. Combined with the results of chapter 5 and chapter 6 it is fair to say that a camera-based solution for the described scenario is a feasible approach worth investigating further.

RQ1.1

Referring back to section 4.1 and the results of chapter 5 and chapter 6, it is likely that a unique discrete-time light signal sent from the devices is one possible approach for detecting and identifying devices using a camera.

RQ1.2

As described in chapter 4, the camera hardware has a significant impact on the system. Some of the limitations imposed by the hardware can be accounted for to a certain degree, such as the variable fps and lack of low-level camera control. However, as described in chapter 6, there are some absolute minimum requirements for the camera that must be met for the system to function reliably.

The camera must be able to record with a frame rate that is minimum three times the symbol rate in order to provide enough samples for reliable detection and tracking. The symbol rate can, in theory, be any value, but in practice, a minimum value of 5 is recommended to improve the participant experience. A lower value requires a longer initialization phase, which in turn also increases the chance of detection or tracking failures. This imposes a minimum requirement of roughly 15 frames per second on the camera, which is achievable even with low-end web cameras.

The resolution of the camera has less impact on the result, slightly depending on the distance from the crowd, given that it is above some undefined minimum. Realistically, any modern camera will have adequate resolution for small venues; larger venues might benefit from at least 1080x720 resolution.

RQ2

Reviewing the literature for object detection (chapter 2) and object tracking (chapter 3) revealed that most existing solutions are not suitable for detection and tracking under the conditions defined by the client. However, some of the basic components described in the literature are still applicable and can be utilized with some modifications, as described in chapter 4.

7.3. Evaluation

This section provides an evaluation of the progress and the different phases of the project. Is also includes a short evaluation of the chosen technologies and the impact they had on the project.

7.3.1. Project phases

Research review

The initial phase of the project consisted mostly of literature reviews, and has resulted in a comprehensive review of existing and relevant research in object detection and object tracking. The results of this review can be seen in chapter 2 and chapter 3, respectively.

This phase was very long and thorough, perhaps even too much so. The majority of the literature that was reviewed was not applicable to the project, mostly due to the nature of the objects and the environment they will be detected and tracked in. The basic methods for image segmentation and the components of multiple object tracking have been instrumental, but the state-of-the-art feature descriptors are not applicable without significant modifications.

This phase could possibly have been shorter in order to leave more time for the other phases.

Develop positioning approach

The next phase consisted of a combination of further research and experimental prototyping, in order to determine if the suggested approach was actually feasible at all. This phase resulted in some very promising results, with preliminary tests suggesting that the approach was viable. The results of this phase are described in chapter 4.

The first steps involved finding suitable photos and videos of people holding their devices at large events on the internet. These were then used to create necessary components for object detection and object tracking, which was able to detect and to some degree track these devices reliably.

The next steps included creating a basic sequence and video generator which could produce some simple videos with stationary, flashing dots, mimicking devices during the initialization phase.

Then began the process of sampling the video to track the lights across multiple frames, in order to detect the sequences they were emitting. Once the underlying algorithm was working for stationary targets, emitting lights at pre-defined intervals, the test generator was layered with additional complexity. This included movement, randomized sizes, randomized intensities, randomized starts and more. For each layer that was added, the algorithm was improved to handle the newly introduced element.

This phase was very experimental in nature, and involved a lot of trials an error. It was by far the most complex and demanding part of the project, but also the most interesting. It should possibly have started earlier than it did, on account of a shorter research phase.

Once the algorithm could reliably track roughly 90% of the devices in a simulated test with 2500 units and moderate movement, it was time to start looking at the other factors of the system.

Implement prototype

After some preliminary testing using the above-mentioned approach, the next phase consisted mostly of software implementation and testing. This resulted in a complete, fully functional system based on a client/server architecture. The system includes a server, an Android client application, two administrator clients and a custom protocol for TCP/IP based communication between the above components. It also includes a comprehensive testing framework with complete support for simulated environments and a desktop version of the client application. The system has been described in detail in chapter 5.

This phase was not as complicated as the previous phase, but it was very time-consuming. Most of the work, except for the positioning approach, has been researched and implemented many times before. However, it still requires a lot of thought and effort to design and implement a server, a mobile client, administrator clients and a custom TCP/IP protocol for communication between all the above-mentioned components.

The required amount of thought and planning was magnified by the fact that it is difficult and very time consuming to perform real, physical tests. This made it very important to develop a good framework for testing early on, which could be used to test isolated parts or the system as a whole. However, this also imposes some requirements and complexity onto the prototype itself, as the test framework must be able to isolate test fixtures or simulate components without affecting other parts of the system.

The implementation phase was a continuous cycle of developing components for the system, developing testing features to exercise the component, and then refactoring the system to allow seamless execution of the tests.

The testing framework proved immensely useful, and made it extremely simple and quick to test different algorithms and changes in the system. The ability to quickly test and isolate various components of the system has been invaluable and integral to the development of the system. It has allowed rapid development and extensive testing of the communication between the server and devices, without having to use actual devices which can be very time consuming to work with. Using the framework alone, it was possible to develop a prototype that required very few changes once testing with real devices began.

This phase was very time consuming, but the effort was required in order to deliver a prototype that would be of use to the client, and to be able to test the system in a real-world scenario.

Real-world tests, experiments, and evaluation

Once the system was functional, some small-scale real-world tests and experiments were performed. The first few experiments were done early, using only a single device. These recordings were then stored, along with all relevant meta-data, so that they could be used as input to the test framework later. This allowed the system to easily be tested against real recordings without having to go through the whole setup with physical devices again.

These initial experiments revealed some unforeseen issues, that were not possible to detect during the simulated test. Solving the problems required some changes in the implementation of the server and mobile client. The exact nature of the problems was unforeseen, but the fact that there would be some issues was expected. The prototype was still surprisingly close to being functional considering it had only been tested against generated videos in simulated environments during development.

The most significant difference was the variable frame rate of the camera, which required the system to gather some additional metadata during the recording. The other issue was related to the mobile client, and how unreliable the screen refresh rate is. Both of these issues were solved, however, and required few or no modifications to the core algorithm.

Once the system was able to detect and stream content to a few devices reliably, some larger scale tests and experiments were conducted.

It would have been advantageous to arrange and perform a large-scale test, not only to test the prototype itself, but also to test the concept and the idea. It would be interesting to find out what potential participants think of the concept, the length of the initialization phase, their willingness to participate and so on. However, in agreement with the client, most of the user-oriented features and interaction has been left out in the prototype. The focus of the project was to find out whether a solution was technically possible, and any future user research, user interface and user interaction are left to Zedge if they find that the concept is worth investigating further.

7.3.2. Technology

Programming language

The choice of programming language, Python 3.6, was mostly based on objective criteria such as the excellent support for image processing, networking, and rapid development. The fact that it is a high-level weakly typed language with a completely dynamic type system makes it extremely easy to iterate and make changes quickly. This likely saved a lot of time, especially during the initial stages of development, as it allowed rapid changed to some parts of the system without having to worry about compilation issues in other parts that were no longer important.

The nature of the type system also made it a lot easier to design and create the test framework, as it allows injecting, wrapping or replacing any function, method or property from anywhere. In a statically typed language, this would require a lot more thought, effort, and ceremony in order to create the necessary interfaces and extension points.

Python also makes it very easy to distribute and deploy the server software, both standalone and through docker, as it is supported on a large variety of platforms and has sophisticated installation and setup tools.

However, as the system matured and the rate of significant changes decreased, some of the disadvantages of a dynamically typed language slowly began to surface. The lack of static types makes it more difficult to refactor without breaking anything, and also decreases the correctness of refactoring tools. A newly introduced feature, type hints[1], does help alleviate this but it is rather verbose.

Another problem surfaced when attempting to utilize concurrency and parallel execution, as the concurrency model in Python is somewhat complicated. It offers a lot of different tools for concurrency, and requires selecting the right tool for the job. That is not to say that the concurrency model is terrible, but it certainly is a bit complicated and requires a lot of research and effort to work with.

Using Python for prototyping and experimental software development has been a success, but it required some adjustment and time to get used to and fully utilize the power of its dynamic nature. It is recommended for future development and prototyping, but perhaps other options should be considered for a production system.

Image and video processing

OpenCV (Open Source Computer Vision Library) is one of the most popular computer vision libraries available, and it has a mature Python interface.

> The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms.

It is written in C++, and uses the excellent NumPy package, which offers highly optimized matrix operations among other things. Together, these libraries provide excellent computer vision capabilities and performance, interfacing directly with highly optimized CUDA[2], OpenCL[3] and C/C++ functions.

[1]https://docs.python.org/3/library/typing.html
[2]https://docs.nvidia.com/cuda/
[3]https://www.khronos.org/opencl/

OpenCV has proven to be an excellent library during the development of the prototype, and is the recommended choice for further development.

Networking

Twisted[4] is an excellent “event-driven networking engine written in Python and licensed under the open source MIT License“. It makes it fairly easy to implement custom network applications and protocols, while still offering all the power of a low-level networking engine if needed. It is able to handle at least 2500 request per second with 2000 concurrent connections[5], which is more than what is required for the Crowdstream system.

While the essential features are simple to use, the low-level network features are still complicated. Implementation of all the various clients, two administrator clients and one basic client for testing, proved to be harder than expected.

Summary

The most compelling reasons for continuing with Python are the excellent libraries and the simple deployment to a large variety of platforms. However, OpenCV is available for many other languages, and most languages offer good networking libraries with capabilities similar to Twisted.

7.4. Discussion

The approach used in this project has been a combination of literature reviews and experimental software design. In this thesis, I propose a new and novel approach for indoor positioning systems, and also present a basic prototype system that utilizes said approach.

The system has proven reliable in small-scale tests and large-scale simulations. However, it has not been tested in a real, practical scenario at an indoor venue with hundreds of participants. It is unlikely that the system will fail to function entirely, but depending on the surrounding environment the reliability of the detection, tracking and positioning algorithms could be severely impacted.

[4]https://twistedmatrix.com/trac/
[5]https://programmingzen.com/benchmarking-tornado-vs-twisted-web-vs-tornado-on-twisted/

The most significant threat is caused by surrounding light sources causing reflections in the device frames themselves to be detected and mistaken for devices. Because of the proximity between the false positives and the devices, the system will not be able to separate the two, and the device will not be detected correctly. However, such high-intensity reflections have only been observed when the devices are in close proximity to the camera, and the device frame is exposed to direct light.

In general, the system does not seem to be impacted by false positive detections as long as these detections are not within the search radius of an actual device. It is however impacted by movements in the crowd, and large movements will cause the respective devices to fail. However, as the participants will be asked to hold their devices still during this short period, this will hopefully not pose a significant problem.

As mentioned in section 1.6, the system and particularly the detection and tracking components, have been developed for a particular scenario. It is unlikely that this project will contribute anything to the general problem of multiple-object tracking. The most significant contribution is in the area of Indoor Positioning Systems, as it seems like this approach has not been used or described before. The general approach can easily be applied in other areas which require indoor positioning, as long as the underlying conditions described in section 7.4 hold true.

The experimental results and the implemented prototype indicates that the suggested approach, which utilizes a camera to detect devices in a crowd, is able to produce reliable results for the intended scenario.

8. Future

This section provides some recommendations for possible extensions, improvements, and further research. Section 8.1 contains an overview of the current state of the research and the prototype, while section 8.2 provides some ideas, pointer, and suggestions for future development, research, and improvements for the system and the prototype.

8.1. Current state

8.1.1. Research

The preliminary research was thorough, and there does not seem to be any existing research that directly relates to this project. However, the possibility of using colors instead of the current on/off scheme was not tested thoroughly enough, and should be revisited.

8.1.2. Server

The current state of prototype bears clear indications that it is still a prototype. It is lacking in documentation, and could use some refactoring of the code. Overall, however, the quality of the code is good, and the separation of concerns between the various components is also good.

The test framework is working well, but could also use some refactoring and documentation. It could also be extended to generate even more realistic tests, which account, e.g., perspective distortion, motion patterns and colored symbols.

Detection

The implementation details of the object detection algorithm are hidden behind a well-defined interface, and can easily be replaced or modified if required.

Tracking

The tracking algorithm is also hidden behind a well-defined interface, but it should perhaps be integrated more with the rest of the server implementation to utilize more domain knowledge. This includes knowledge about the sequences in particular. Improvements in the tracking component would likely provide the most short-term benefit to the system, as described in section 8.2.

8.1.3. Mobile client

The implementation of the mobile client is very rudimentary. It provides the required functionality to function properly, but does not offer a very good user experience. The implementation of the network protocol is also very basic, and should be refactored or rewritten to be more robust. However, the implementation of the surface view, which displays the colors, and the logic which synchronizes the display of the sequences, is working well.

8.1.4. Administrator clients

The two administrator clients work well, but should likely be extended to provide additional functionality, e.g., the ability to manually disconnect devices or retrieve more information about the current state of the server.

8.1.5. Protocol

The protocol functions well enough to serve a few hundred clients, but a linefeed protocol might be unnecessarily verbose for a production system. It could be replaced with a much more compact protocol based on raw bytes, in particular for streaming data. The color values could easily be compressed to 3 bytes, RGB. The streaming protocol could also be replaced with a UDP protocol as it does not need the extra features, and overhead, provided by TCP.

8.2. Future Development

8.2.1. Multiple cameras

Occlusion and articulated motion are two of the most detrimental factors of the current system. One possible solution to overcome this problem could be to utilize multiple cameras mounted at different positions. This does add considerable complexity, but has shown noticeable improvements over single camera approaches for general object tracking[14].

8.2.2. Perspective distortion

Because the camera will be mounted near or above the stage, at some distance from the crowd and at an angle, there will be perspective distortion. Because of this distortion, as seen in Figure 8.1, participants closer to the camera require a large search distance than participants farther away, as their movement will be more significant in the captured video. It also affects the mapping of content onto the participant devices, which should be corrected for this distortion.

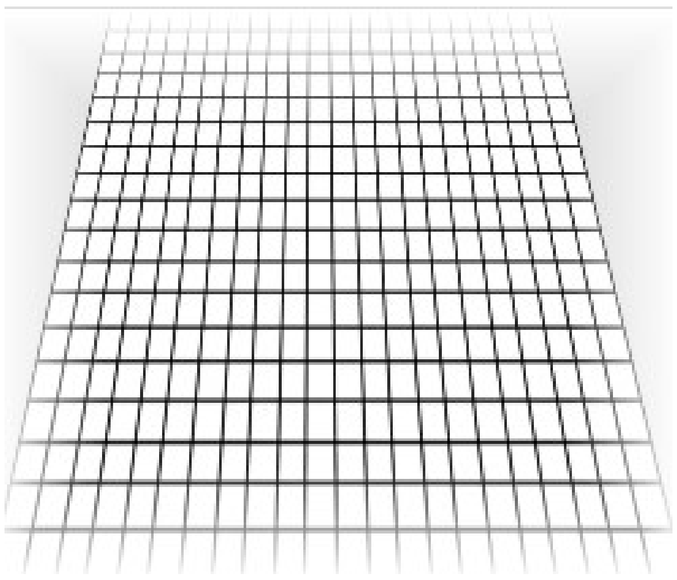


Figure 8.1.: Example of perspective distortion

8.2.3. Interaction model

An interaction model, as mentioned in section 3.5, could increase the accuracy of the tracking algorithm. It could, for example, include the interaction of people moving together as a crowd, perhaps in motion with the music. It could also include devices ”bumping” off each other, modeling how the participants cannot physically occupy the same space, and their hands or bodies are likely to bump off each other if they get too close.

8.2.4. Inference

The current inference method utilizes a basic greedy nearest neighbor approach, and could possibly be improved. As mentioned in section 3.8, two general approaches exist; probabilistic inference and deterministic optimization. Methods based on Kalman filter, Extended Kalman Filter or any of the methods based on deterministic optimization, particularly a min-cost max-flow network, could potentially provide better results at the cost of some complexity and speed.

8.2.5. Feature descriptors

Modern feature detection and extraction algorithms will likely introduce much complexity to this prototype of Crowdstream, and given the time constraints of the project, it is not feasible to attempt to implement any such algorithms as this point. However, it could be considered again in the future if the performance of the current object detection scheme proves to be insufficient.

8.2.6. Colored symbols

At the initial stages of the project, the possibility of using colored symbols instead of the current on/off implementation, was investigated and researched. The conclusion at the time was that it would be difficult to distinguish the different colors at longer distances, partly based on existing images of large crowds holding their devices. However, it is possible that careful selection of the possible color values, for example, one carefully selected hue of red, green and blue respectively, for a total of 3 colors, could be viable. This would most likely reduce the amount of failed tracks, as there is more information available to the tracking algorithm which can be utilized to associate a detection to a track.

There are two possible approaches that utilize colors:

Assign one color to each sequence or device Using this scheme, each device would receive a single color that should be used for the whole sequence. This would let the tracking algorithm discard any detections that do not match the color of the existing track.

Assign one color to each symbol In this scheme, each symbol in all the sequences is assigned a specific color. The algorithm would then utilize its knowledge of all the generated sequences in order to determine if a considered association would result in an invalid sequence.

The first approach is the easiest to implement and would require very little modification the algorithm itself. The second approach requires slightly more modification, but is still reasonably easy to implement. The second scheme is most likely more robust, as the first scheme would still fail when devices assigned the same color are near each other.

8.2.7. Motion model

The motion model could be improved to model the motion pattern of devices when held by humans. There are motion bounds for the range of movement, bounded by the length of a humans arms. The device will also most likely move more or less horizontally within these bounds, and abruptly change directions near these bounds. This could be modeled explicitly to increase the accuracy of the predicted position of a device.

8.2.8. Android Instant App

In order to lower the barrier for participation, it is possible to implement the Android application as an "Instant App"[1]. These applications do not require installation, and can be launched directly from, e.g., a link. This could potentially increase the number of participants. This application is a good candidate for an instant app as it is minimal in size and has a limited number of screens.

[1]https://developer.android.com/topic/google-play-instant/

8.2.9. General tracking

There are many general improvements that could be made to the tracking component of the system. These include improvements to the existing algorithm, or replacing with another algorithm altogether. Some possible alternatives are methods based on Joint Probabilistic Data Association Filter, Kalman filters, or Multi-Hypothesis Tracking, possibly with Murty's K-best method.

8.2.10. Admin panel

The administrator interface leaves a lot to be desired. One possible solution is to add an HTTP interface to the server, and build a web front-end for it. Python and Twisted offer excellent HTTP support, and it would be fairly easy to implement the API for it.

8.2.11. Security

The security of the system has not been prioritized, but should be improved before using it in production. The administrator password is sent in clear-text on every login or request in the case of Admin client and Admin CLI, respectively. If an HTTP interface as suggested above is created, one possible solution would be to use standard HTTPS for administrator access.

9. Bibliography

[1] M. A. Al-Ammar, S. Alhadhrami, A. Al-Salman, A. Alarifi, H. S. Al-Khalifa, A. Alnafessah, and M. Alsaleh. Comparative survey of indoor positioning technologies, techniques, and algorithms. In *Proceedings - 2014 International Conference on Cyberworlds, CW 2014*, pages 245–252. IEEE, 10 2014. ISBN 9781479946778. doi: 10.1109/CW.2014.41. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6980768`.

[2] A. Andriyenko, K. Schindler, and S. Roth. Discrete-continuous optimization for multi-target tracking. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1926–1933, 2012. ISBN 9781467312264. doi: 10.1109/CVPR.2012.6247893.

[3] S. Angelina, L. P. Suresh, and S. H. K. Veni. Image segmentation based on genetic algorithm for region growth and region merging. *2012 International Conference on Computing, Electronics and Electrical Technologies, ICCEET 2012*, pages 970–974, 2012. doi: 10.1109/ICCEET.2012.6203833.

[4] B. G. Batchelor. Edge-Region-Based Segmentation of Range Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(3):314–319, 1994. ISSN 01628828. doi: 10.1109/34.276131.

[5] J. Berclaz, F. Fleuret, E. Türetken, and P. Fua. Multiple object tracking using k-shortest paths optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(9):1806–1819, 2011. ISSN 01628828. doi: 10.1109/TPAMI.2011.21. URL `https://infoscience.epfl.ch/record/164041/files/top.pdf`.

[6] K. Bernardin and R. Stiefelhagen. Evaluating multiple object tracking performance: The CLEAR MOT metrics. *Eurasip Journal on Image and Video Processing*, 2008, 2008. ISSN 16875176. doi: 10.1155/2008/246309. URL `https://link.springer.com/content/pdf/10.1155%2F2008%2F246309.pdf`.

[7] M. D. Breitenstein, F. Reichlin, B. Leibe, E. Koller-Meier, and L. Van Gool. Robust tracking-by-detection using a detector confidence particle filter. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1515–1522, 2009. ISBN 9781424444205. doi: 10.1109/ICCV.2009.5459278.

[8] W. Brendel, M. Amer, and S. Todorovic. Multiobject tracking as maximum weight independent set. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1273–1280, 2011. ISBN 9781457703942. doi: 10.1109/CVPR.2011.5995395.

[9] J. Canny. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986. ISSN 01628828. doi: 10.1109/TPAMI.1986.4767851.

[10] N. Coudray, J. L. Buessler, and J. P. Urban. Robust threshold estimation for images with unimodal histograms. *Pattern Recognition Letters*, 31(9):1010–1019, 2010. ISSN 01678655. doi: 10.1016/j.patrec.2009.12.025.

[11] M. Cristani, M. Bicego, and V. Murino. Integrated region- and pixel-based approach to background modelling. In *Proceedings - Workshop on Motion and Video Computing, MOTION 2002*, pages 3–8, 2002. ISBN 0769518605. doi: 10.1109/MOTION.2002.1182206.

[12] L. S. Davis, A. Rosenfeld, and J. S. Weszka. Region Extraction by Averaging and Thresholding. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-5(3): 383–388, 1975. ISSN 21682909. doi: 10.1109/TSMC.1975.5408419.

[13] J. a. Delmerico, P. David, and J. J. Corso. Building facade detection, segmentation, and parameter estimation for mobile robot localization and guidance. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1632–1639, 2011. ISBN 978-1-61284-456-5. doi: 10.1109/IROS.2011.6094778. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6094778`.

[14] S. L. Dockstader and A. M. Tekalp. Multiple camera fusion for multi-object tracking. In *Proceedings - 2001 IEEE Workshop on Multi-Object Tracking, MOT 2001*, 2001. ISBN 0769511716. doi: 10.1109/MOT.2001.937987.

[15] A. Dos Anjos and H. R. Shahbazkia. BI-level image thresholding - A fast method. *BIOSIGNALS 2008 - Proceedings of the 1st International Conference on Bio-inspired Systems and Signal Processing*, 2:70–76, 2008. URL `http://www.scopus.com/inward/record.url?eid=2-s2.0-55649113938&partnerID=40&md5=eb43decceb1c924d8d80be0f86e01b51`.

[16] A. Elgammal, R. Duraiswami, D. Harwood, and L. S. Davis. Background and foreground modeling using nonparametric kernel density estimation for visual surveillance. *Proceedings of the IEEE*, 90(7):1151–1162, 2002. ISSN 00189219. doi: 10.1109/JPROC.2002.801448.

[17] M. Forouzanfar, N. Forghani, and M. Teshnehlab. Parameter optimization of im-

proved fuzzy c-means clustering algorithm for brain MR image segmentation. *Engineering Applications of Artificial Intelligence*, 23(2):160–168, 2010. ISSN 09521976. doi: 10.1016/j.engappai.2009.10.002.

[18] P. E. Forssén. Maximally stable colour regions for recognition and matching. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2007. ISBN 1424411807. doi: 10.1109/CVPR.2007.383120.

[19] P. E. Forssén and D. G. Lowe. Shape descriptors for maximally stable extremal regions. In *Proceedings of the IEEE International Conference on Computer Vision*, 2007. ISBN 978-1-4244-1631-8. doi: 10.1109/ICCV.2007.4409025.

[20] L. Garcia Ugarriza, E. Saber, S. R. Vantaram, V. Amuso, M. Shaw, and R. Bhaskar. Automatic image segmentation by dynamic region growth and multiresolution merging. *IEEE Transactions on Image Processing*, 18(10):2275–2288, 2009. ISSN 10577149. doi: 10.1109/TIP.2009.2025555.

[21] S. Garcia-Villalonga and A. Perez-Navarro. Influence of human absorption of Wi-Fi signal in indoor positioning with Wi-Fi fingerprinting. In *2015 International Conference on Indoor Positioning and Indoor Navigation, IPIN 2015*, 2015. ISBN 9781467384025. doi: 10.1109/IPIN.2015.7346778.

[22] R. C. Gonzalez and R. E. Woods. *Digital Image Processing (3rd Edition)*. 2007. ISBN 013168728X. URL `http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&amp;path=ASIN/013168728X`.

[23] D. Helbing and P. Molnár. Social force model for pedestrian dynamics. *Physical Review E*, 51(5):4282–4286, 1995. ISSN 1063651X. doi: 10.1103/PhysRevE.51.4282.

[24] M. Hofmann, P. Tiefenbacher, and G. Rigoll. Background segmentation with feedback: The pixel-based adaptive segmenter. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 38–43, 2012. ISBN 9781467316118. doi: 10.1109/CVPRW.2012.6238925.

[25] M. Hu, S. Ali, and M. Shah. Detecting global motion patterns in complex videos. In *2008 19th International Conference on Pattern Recognition*, pages 1–5, 2008. ISBN 978-1-4244-2174-9. doi: 10.1109/ICPR.2008.4760950. URL `http://ieeexplore.ieee.org/document/4760950/`.

[26] H. Izadinia, I. Saleemi, W. Li, and M. Shah. (MP) 2T: Multiple people multiple parts tracker. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 7577 LNCS, pages 100–114, 2012. ISBN 9783642337826. doi: 10.1007/978-3-642-33783-3{_}8.

[27] L. Juan and O. Gwun. A comparison of sift, pca-sift and surf. *International Journal of Image Processing (IJIP)*, 3(4):143–152, 2009. ISSN 00496979. doi: 10.1007/s11270-006-2859-8.

[28] R. Kasturi, D. Goldgof, P. Soundararajan, V. Manohar, J. Garofolo, R. Bowers, M. Boonstra, V. Korzhova, and J. Zhang. Framework for performance evaluation of face, text, and vehicle detection and tracking in video: Data, metrics, and protocol. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2009. ISSN 01628828. doi: 10.1109/TPAMI.2008.57.

[29] D. Koller, J. Weber, and J. Malik. Robust Multiple Car Tracking with Occlusion Reasoning. *Computer*, 1(January):189–196, 1994. ISSN 1098-6596. doi: 10.1007/3-540-57956-7{_}22. URL `http://www.springerlink.com/index/F766437V25K87854.pdf`.

[30] L. Kratz and K. Nishino. Tracking pedestrians using local spatio-temporal motion patterns in extremely crowded scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2012. ISSN 01628828. doi: 10.1109/TPAMI.2011.173.

[31] M. Kunt, A. Ikonomopoulos, and M. Kocher. Second-Generation Image-Coding Techniques. *Proceedings of the IEEE*, 73(4):549–574, 1985. ISSN 15582256. doi: 10.1109/PROC.1985.13184.

[32] C. H. Kuo and R. Nevatia. How does person identity recognition help multi-person tracking? In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1217–1224, 2011. ISBN 9781457703942. doi: 10.1109/CVPR.2011.5995384.

[33] Y. Li, S. Wang, Q. Tian, and X. Ding. A survey of recent advances in visual feature detection. *Neurocomputing*, 149(PB):736–751, 2015. ISSN 18728286. doi: 10.1016/j.neucom.2014.08.003.

[34] W. C. Liu, S. Z. Lin, M. H. Yang, and C. R. Huang. Real-time binary descriptor based background modeling. In *Proceedings - 2nd IAPR Asian Conference on Pattern Recognition, ACPR 2013*, pages 722–726, 2013. ISBN 9781479921904. doi: 10.1109/ACPR.2013.125.

[35] J. Luo and X. Zhan. Characterization of smart phone received signal strength indication for WLAN indoor positioning accuracy improvement. *Journal of Networks*, 9(3):739–746, 2014. ISSN 17962056. doi: 10.4304/jnw.9.3.739-746.

[36] W. Luo, T. K. Kim, B. Stenger, X. Zhao, and R. Cipolla. Bi-label propagation for generic multiple object tracking. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1290–1297, 2014.

ISBN 9781479951178. doi: 10.1109/CVPR.2014.168.

[37] W. Luo, J. Xing, X. Zhang, X. Zhao, and T.-K. Kim. Multiple Object Tracking: A Literature Review. *arXiv, 1-39. Retrieved from http://arxiv.org/abs/1409.7618*, V(212), 2014. doi: 10.1145/0000000.0000000. URL `http://arxiv.org/abs/1409.7618`.

[38] W. Luo, B. Stenger, X. Zhao, and T.-K. Kim. Automatic Topic Discovery for Multi-Object Tracking. In *Proceedings of the Twenty-Ninth AAAI COnference on Artificial Intelligence*, number 1, pages 3820–3826, 2015. ISBN 9781577357032.

[39] R. Maini and H. Aggarwal. Study and comparison of various image edge detection techniques. *International Journal of Image Processing*, 3(1):1–11, 2009. ISSN 16113349. doi: http://www.doaj.org/doaj?func=openurl{\&}genre=article{\&}issn=19852304{\&}date=2009{\&}volume=3{\&}issue=1{\&}spage=1.

[40] J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. In *Image and Vision Computing*, volume 22, pages 761–767, 2004. ISBN 1-901725-19-7. doi: 10.1016/j.imavis.2004.02.006.

[41] A. Milan, K. Schindler, and S. Roth. Detection- and trajectory-level exclusion in multiple object tracking. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3682–3689, 2013. ISBN 1063-6919 VO -. doi: 10.1109/CVPR.2013.472.

[42] A. Milan, S. Roth, and K. Schindler. Continuous energy minimization for multitarget tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36 (1):58–72, 2014. ISSN 01628828. doi: 10.1109/TPAMI.2013.103.

[43] D. Mitzel, E. Horbert, A. Ess, and B. Leibe. Multi-person tracking with sparse detection and continuous segmentation. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 6311 LNCS, pages 397–410, 2010. ISBN 3642155480. doi: 10.1007/978-3-642-15549-9{_}29.

[44] Y. Nonaka, A. Shimada, H. Nagahara, and R.-i. Taniguchi. Evaluation report of integrated background modeling based on spatio-temporal features. *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 9–14, 2012. ISSN 978-1-4673-1612-5. doi: 10.1109/CVPRW.2012.6238920. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6238920`.

[45] P. B. Padmanabhan and V. N. RADAR: An in-building RF based user location and tracking system. *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and*

Communications Societies (Cat. No.00CH37064), 2(c):775–784, 2000. ISSN 0743-166X. doi: 10.1109/INFCOM.2000.832252. URL `http://research.microsoft.com/en-us/groups/sn-res/infocom2000.pdf`.

[46] N. R. Pal and S. K. Pal. A review on image segmentation techniques. *Pattern Recognition*, 26(9):1277–1294, 1993. ISSN 00313203. doi: 10.1016/0031-3203(93)90135-J.

[47] S. Pellegrini, A. Ess, K. Schindler, and L. Van Gool. You'll never walk alone: Modeling social behavior for multi-target tracking. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 261–268, 2009. ISBN 9781424444205. doi: 10.1109/ICCV.2009.5459260.

[48] D. L. Pham, C. Xu, and J. L. Prince. Current Methods in Medical Image Segmentation. *Annual Review of Biomedical Engineering*, 2(1):315–337, 2000. ISSN 1523-9829. doi: 10.1146/annurev.bioeng.2.1.315. URL `http://www.annualreviews.org/doi/10.1146/annurev.bioeng.2.1.315`.

[49] C. M. Ramya, M. Shanmugaraj, and R. Prabakaran. Study on ZigBee technology. In *ICECT 2011 - 2011 3rd International Conference on Electronics Computer Technology*, volume 6, pages 297–301, 2011. ISBN 9781424486779. doi: 10.1109/ICECTECH.2011.5942102.

[50] D. Reid. An algorithm for tracking multiple targets. *IEEE Transactions on Automatic Control*, 24(6):843–854, 1979. ISSN 0018-9286. doi: 10.1109/TAC.1979.1102177. URL `http://ieeexplore.ieee.org/document/1102177/`.

[51] S. Ridler, T.W. Calvard. Picture Thresholding Using an Iterative Slection Method. *IEEE Transactions on Systems, Man and Cybernetics*, 8(8):630–632, 1978. ISSN 00189472. doi: 10.1109/TSMC.1978.4310039.

[52] B. Ristic, Ba-Ngu Vo, D. Clark, and Ba-Tuong Vo. A Metric for Performance Evaluation of Multi-Target Tracking Algorithms. *IEEE Transactions on Signal Processing*, 2011. ISSN 1053-587X. doi: 10.1109/TSP.2011.2140111.

[53] P. L. Rosin. Unimodal thresholding. *Pattern Recognition*, 34(11):2083–2096, 2001. ISSN 00313203. doi: 10.1016/S0031-3203(00)00136-9.

[54] E. Rosten and T. Drummond. Machine Learning for High Speed Corner Detection. *Computer Vision – ECCV 2006*, 1:430–443, 2006. ISSN 03029743. doi: 10.1007/11744023{_}34.

[55] D. Russell and S. Gong. A highly efficient block-based dynamic background model. In *IEEE International Conference on Advanced Video and Signal Based Surveillance*

- *Proceedings of AVSS 2005*, volume 2005, pages 417–422, 2005. ISBN 0780393856. doi: 10.1109/AVSS.2005.1577305.

[56] E. Salahat and M. Qasaimeh. Recent advances in features extraction and description algorithms: A comprehensive survey. In *Proceedings of the IEEE International Conference on Industrial Technology*, pages 1059–1063, 2017. ISBN 9781509053209. doi: 10.1109/ICIT.2017.7915508. URL `https://arxiv.org/pdf/1703.06376.pdf`.

[57] N. Senthilkumaran and R. Rajesh. Edge detection techniques for image segmentation–a survey of soft computing approaches. *International Journal of Recent Trends in Engineering and Technology*, 1(2):250–254, 2009. doi: 10.1109/ARTCom.2009.219. URL `http://searchdl.org/public/journals/2009/IJRTET/1/2/1474.pdf`.

[58] M. Sezgin and B. Sankur. Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic Imaging*, 13(1):146–165, 2004. ISSN 10179909. doi: 10.1117/1.1631316.

[59] L. Shapiro and G. Stockman. Computer Vision. *October*, 2004(October):608, 2001. ISSN 15562646. doi: 10.1525/jer.2008.3.1.toc. URL `http://www.amazon.com/Computer-Vision-Linda-G-Shapiro/dp/0130307963`.

[60] K. K. Singh and A. Singh. A study of image segmentation algorithms for different types of images. *International Journal of Computer Science*, 7(5):414–417, 2010. ISSN 1694-0814.

[61] B. Song, T. Y. Jeng, E. Staudt, and A. K. Roy-Chowdhury. A stochastic graph evolution framework for robust multi-target tracking. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 6311 LNCS, pages 605–619, 2010. ISBN 3642155480. doi: 10.1007/978-3-642-15549-9{_}44.

[62] J. Song, S. Hur, Y. Park, and J. Choi. An improved RSSI of geomagnetic field-based indoor positioning method involving efficient database generation by building materials. In *2016 International Conference on Indoor Positioning and Indoor Navigation, IPIN 2016*, 2016. ISBN 9781509024254. doi: 10.1109/IPIN.2016.7743605.

[63] C. Stauffer and W. E. L. Grimson. Adaptive background mixture models for real-time tracking. *Proceedings 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Cat No PR00149*, 2(c):246–252, 1999. ISSN 10636919. doi: 10.1109/CVPR.1999.784637. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=784637`.

[64] D. Sugimura, K. M. Kitani, T. Okabe, Y. Sato, and A. Sugimoto. Using individ-

uality to track individuals: Clustering individual trajectories in crowds using local appearance and frequency trait. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1467–1474, 2009. ISBN 9781424444205. doi: 10.1109/ICCV.2009.5459286.

[65] N. Thi, L. Anh, F. M. Khan, F. Negin, and F. Bremond. Multi-Object tracking using Multi-Channel Part Appearance Representation. URL `http://www-sop.inria.fr/members/Francois.Bremond/Postscript/LanAnhAVSS2017.pdf`.

[66] K. Toyama, J. Krumm, B. Brumitt, and B. Meyers. Wallflower: principles and practice of background maintenance. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, pages 255–261, 1999. ISBN 0-7695-0164-8. doi: 10.1109/ICCV.1999.791228. URL `http://ieeexplore.ieee.org/document/791228/`.

[67] T. Tuytelaars and K. Mikolajczyk. Local Invariant Feature Detectors: A Survey. *Foundations and Trends® in Computer Graphics and Vision*, 3(3):177–280, 2007. ISSN 1572-2740. doi: 10.1561/0600000017. URL `http://www.nowpublishers.com/article/Details/CGV-017`.

[68] H. Voorhees and T. Poggio. Detecting Textons and texture Boundaries in Natural Image. In *Proceedings of the First International Conference on Computer Vision London*, pages 250–258, 1987. ISBN 081860777X.

[69] C. Wu, Z. Yang, and Y. Liu. Smartphones based crowdsourcing for indoor localization. *IEEE Transactions on Mobile Computing*, 14(2):444–457, 2015. ISSN 15361233. doi: 10.1109/TMC.2014.2320254.

[70] S. Xia, Y. Liu, G. Yuan, M. Zhu, and Z. Wang. Indoor Fingerprint Positioning Based on Wi-Fi: An Overview. *ISPRS International Journal of Geo-Information*, 6(5):135, 4 2017. ISSN 2220-9964. doi: 10.3390/ijgi6050135. URL `http://www.mdpi.com/2220-9964/6/5/135`.

[71] Y. Xiang, A. Alahi, and S. Savarese. Learning to track: Online multi-object tracking by decision making. In *Proceedings of the IEEE International Conference on Computer Vision*, 2015. ISBN 9781467383912. doi: 10.1109/ICCV.2015.534.

[72] J. Xing, H. Ai, and S. Lao. Multi-object tracking through occlusions by local tracklets filtering and global tracklets association with detection responses. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2009*, pages 1200–1207, 2009. ISBN 9781424439935. doi: 10.1109/CVPRW.2009.5206745.

[73] J. Xing, H. Ai, L. Liu, and S. Lao. Multiple player tracking in sports video: A dual-

mode two-way Bayesian inference approach with progressive observation modeling. *IEEE Transactions on Image Processing*, 20(6):1652–1667, 2011. ISSN 10577149. doi: 10.1109/TIP.2010.2102045.

[74] Y. Xu, J. Dong, B. Zhang, and D. Xu. Background modeling methods in video analysis: A review and comparative evaluation. *CAAI Transactions on Intelligence Technology*, 1(1):43–60, 2016. ISSN 24682322. doi: 10.1016/j.trit.2016.03.005. URL `http://linkinghub.elsevier.com/retrieve/pii/S2468232216000068`.

[75] K. Yamaguchi, A. C. Berg, L. E. Ortiz, and T. L. Berg. Who are you with and where are you going? In *CVPR 2011*, 2011. ISBN 978-1-4577-0394-2. doi: 10.1109/CVPR.2011.5995468.

[76] B. Yang, C. Huang, and R. Nevatia. Learning affinities and dependencies for multi-target tracking using a CRF model. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1233–1240, 2011. ISBN 9781457703942. doi: 10.1109/CVPR.2011.5995587.

[77] A. Yassin, Y. Nasser, M. Awad, A. Al-Dubai, R. Liu, C. Yuen, R. Raulefs, and E. Aboutanios. Recent Advances in Indoor Localization: A Survey on Theoretical Approaches and Applications. *IEEE Communications Surveys & Tutorials*, 19(2): 1327–1346, 2017. ISSN 1553-877X. doi: 10.1109/COMST.2016.2632427. URL `http://ieeexplore.ieee.org/document/7762095/`.

[78] J. H. Yoon, C.-R. Lee, M.-H. Yang, and K.-J. Yoon. Online Multi-object Tracking via Structural Constraint Event Aggregation. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1392–1400. IEEE, 6 2016. ISBN 978-1-4673-8851-1. doi: 10.1109/CVPR.2016.155. URL `http://ieeexplore.ieee.org/document/7780524/`.

[79] M. Youssef and A. Agrawala. The Horus WLAN location determination system. In *Proceedings of the 3rd international conference on Mobile systems, applications, and services - MobiSys '05*, page 205, 2005. ISBN 1931971315. doi: 10.1145/1067170.1067193. URL `http://portal.acm.org/citation.cfm?doid=1067170.1067193`.

[80] L. Zhang and L. Van Der Maaten. Structure preserving object tracking. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1838–1845, 2013. ISBN 978-0-7695-4989-7. doi: 10.1109/CVPR.2013.240.

[81] L. Zhang, Y. Li, and R. Nevatia. Global data association for multi-object tracking using network flows. *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008. ISSN 1063-6919. doi: 10.1109/CVPR.2008.4587584. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.`

`htm?arnumber=4587584`.

[82] Y. Zhang and L. Wu. Optimal multi-level thresholding based on maximum Tsallis entropy via an artificial bee colony approach. *Entropy*, 13(4):841–859, 2011. ISSN 10994300. doi: 10.3390/e13040841.

Glossary

Bhattacharyya distance A measure of the similarity of two probability distributions. 28, 123

blob A region of an image in which some properties are constant or approximately constant; all the points in a blob can be considered in some sense to be similar to each other. 20, 123

centroid The arithmetic mean position of all the points in a shape. 63, 81, 87, 123

contour A curve joining all the continuous points (along the boundary), having same color or intensity. 62, 63, 123

Euclidean distance The straight line distance between two points. 81, 123

image moment A certain particular weighted average (moment) of the image pixels' intensities, or a function of such moments, usually chosen to have some attractive property or interpretation. 63, 123

multimodal A continuous probability distribution with two or more modes (peaks). 15, 123

symbol A state of the communication channel that persists for a fixed period of time. 39, 66, 123

symbol rate The number of symbol changes, waveform changes, or signaling events, across the transmission medium per time unit using a digitally modulated signal or a line code. Also known as baud rate and modulation rate. 38, 123

tracklet A fragment of the track followed by a moving object, as constructed by an image recognition system. 26, 123

unimodal A continuous probability distribution with exactly one mode (peak). 15, 123

unit interval The minimum time interval between condition changes of a data transmission signal. Also known as pulse time and symbol duration time. 38, 39, 123

Acronyms

AGMM Adaptive Gaussian Mixture Model. 19, 123

API Application Programming Interface. 68, 123

BT Bluetooth. 1, 123

CLI Command Line Interface. 7, 51, 69, 123, 135

CLRF Carriage Return Line Feed. 51, 123

CRF Conditional Random Field. 34, 123

CSV Comma Separated Values. 52, 123

DBT Detection-Based Tracking. ix, x, 24, 25, 47, 123

DFT Detection-Free Tracking. ix, x, 24–26, 47, 123

DNCC Deep Convolutional Neural Network. 18, 123

ECC Error Correction Codes. 40, 123

FAF False Alarms per Frame. 78, 123

FAST Features from Accelerated Segment Test. 22, 123

FEC Forward Error Correction. 40, 123

FN False Negative. 94, 123

FP False Positive. 94, 123

FPS Frames Per Second. 9, 38–41, 53, 68, 123

GA Genetic Algorithm. 14, 123

GMM Gaussian Mixture Model. 19, 123

GPS Global Positioning System. 1, 54, 123

GT Ground Truth. 123

IP Internet Protocol. 67, 123

IPS Indoor Positioning System. 1, 2, 7, 123

JPDAF Joint Probabilistic Data Association. 123

KDE Kernel Density Estimation. 123

LOG Laplace of Gaussian. 16, 123

MAP Maximal a Posteriori. 24, 32, 33, 123

MODA Multiple Object Detection Accuracy. 78, 123

MODP Multiple Object Detection Precision. 78, 123

MOT Multiple Object Tracking. x, 23–27, 31, 33, 34, 49, 77, 123

MOTA Multiple Object Tracking Accuracy. 79, 88, 91, 123

MOTP Multiple Object Tracking Precision. 79, 88, 91, 123

MSER Maximally Stable Extremal Region. 20, 22, 123

MTT Multiple Target Tracking. 34, 123

MWIS Maximum-weight Independent Set. 34, 123

NCC Normalized Cross Correlation. 123

NN Neural Network. 14, 123

OSPA Optimal Subpattern Assignment. 79, 123

PBAS Pixel Based Adaptive Segmenter. 19, 123

PDAF Probabilistic Data Association. 123

PDE Partial Differential Equation. 21, 123

POM Probabilistic Occupancy Map. 28, 123

RFID Radio Frequency Identification. 1, 123

RGB Red Green Blue color model. 53, 123

RQ Research Question. 3, 123

SIFT Scale-Invariant Feature Transform. 20–22, 123

SOTA State of the Art. 11, 123

SURF Speeded Up Robust Features. 20–22, 123

TCP Transport Connection Protocol. 51, 123

TDE Tracking Distance Error. 79, 123

TP True Positive. 94, 123

UI User Interface. 68, 123

UWB Ultra Wide Band. 1, 123

A. Test Plan

Test Plan for Crowdstream Server

Purpose

This document describes the plan for testing the prototype of the Crowdstream system. This test plan supports the following objectives:

- Identify existing project information and the software that should be tested
- List the recommended test requirements
- Describe the testing methods to be employed
- Identify the required resources and provide an estimate of the test efforts
- List the deliverable elements of the test activities

Scope

The test plan describes the integration, usability and system tests that will be conducted on the crowdstream prototype. It it assumed that unit testing has already been provided through unit tests and simulated tests.
The interfaces between the following components will be tested:

- Crowdstream server
- Crowdstream mobile app
- Crowdstream admin CLI

The most critical performance measures to test are:

- Response time for admin CLI
- Response time for device connections and disconnects
- Network delay during streaming
- App performance during streaming on older devices

Tests

Device positioning

The system should be able to position 90% of participating devices under normal circumstances

Objectives

Determine how well the system can detect, track and position devices under varying conditions

Cases

Stationary devices
Low random movement
Medium random movement
High random movement
Partial short-term (0.2s) obstruction of device
Complete short-term (0.2s) obstruction of device
Partial medium-term(2s) obstruction of device
Complete medium-term(2s) obstruction of device
Low light conditions
Medium light conditions
High light conditions
Short distance from camera, < 5m
Medium distance from camera, 5-10m
Long distance from camera, > 10m

Method

A number of real devices will be placed at the floor or held by assistants in one half of a hall. A camera will be placed near the roof at the other end of the hall, which will be used by the system to record the devices.

Analysis

The system will be evaluated based on it’s detection performance during testing. The input from the camera as well as frame timing data will be stored so that it can be used for further evaluation, analysis and development.

System streaming performance

The system can successfully handle 2500 devices in simulated tests. Due to practical constraints it is not possible to test with 2500 real devices.

Objective

Determine how well the system performs during streaming with real devices.

Cases

Stream from video
Stream from single image
Stream from webcam

Method

The system will be set up with roughly 10 devices. The system will then be used to stream different content to all devices.

Analysis

The output from the devices during streaming will be recorded and analysed. Should also look at number of dropped frames, latency and other metrics.

Usability

The system should be simple to set up and use for the client

Objective

Determine if the system can be used by someone with general technical knowledge but no system specific knowledge, using a written guide.

Method

The client will be provided with a written explanation of how to set up and use the system

Analysis

Feedback from the client will be reviewed.

B. Results from experiments

Movement	Light	Distance	GT	TP	FP	FN	Recall	Precision
Stationary	Dark	Close	2	2	0	0	100%	100%
		Medium	2	2	0	0	100%	100%
		Far	3	3	0	0	100%	100%
	Medium	Close	2	2	0	0	100%	100%
		Medium	2	2	2	0	100%	50%
		Far	3	3	3	0	100%	50%
	Bright	Close	2	2	3	0	100%	40%
		Medium	2	2	11	0	100%	15.4%
		Far	3	3	16	0	100%	15.8%
Low	Dark	Close	2	2	0	0	100%	100%
		Medium	2	2	0	0	100%	100%
		Far	2	2	0	0	100%	100%
	Medium	Close	2	2	1	0	100%	66.7%
		Medium	2	2	3	0	100%	40%
		Far	2	2	5	0	100%	28.6%
	Bright	Close	2	1	4	1	50%	20%
		Medium	2	2	11	0	100%	15.4%
		Far	2	2	14	0	100%	12.5%
Moderate	Dark	Close	2	2	0	0	100%	100%
		Medium	2	2	0	0	100%	100%
		Far	2	2	0	0	100%	100%
	Medium	Close	2	1	2	1	50%	33.3%
		Medium	2	2	3	0	100%	40%
		Far	2	2	5	0	100%	28.6%
	Bright	Close	2	1	3	1	50%	25%
		Medium	2	1	9	1	50%	10%
		Far	2	0	14	2	0%	0%
High	Dark	Close	2	0	0	2	0%	0%
		Medium	2	0	0	2	0%	0%
		Far	2	0	0	2	0%	0%
	Medium	Close	2	0	3	2	0%	0%
		Medium	2	0	4	2	0%	0%
		Far	2	0	7	2	0%	0%
	Bright	Close	2	0	4	2	0%	0%
		Medium	2	0	11	2	0%	0%
		Far	2	0	17	2	0%	0%

Table B.1.: Experiment results

C. Setup

C.1. Installation

The system offers simple installation through python pip[1] or easy install[2]. It requires an existing Python 3.6 environment and an internet connection, but any additional dependencies will be downloaded and installed automatically. Using either installation method will result in two new tools being available on the command-line:

cws This tool is the actual server, and can be used to configure and start an instance of the server.

cws-cli This tool is the administrator CLI, which provides access to and control of an already running server instance.

Both tools provide documentation and help regarding available commands and their arguments.

C.2. Deployment

C.2.1. Local machine

In order to run the system on a local machine, simply install and run ‘cws‘ from the command-line, optionally specifying a port and administrator password.

The server can then be manged by executing commands with ‘cws-cli‘ or the CLI client.

[1]https://pypi.org/project/pip/
[2]http://setuptools.readthedocs.io/en/latest/easy_install.html

C.2.2. Docker

The server comes with a complete Dockerfile, which can be used to run the server in a Docker[3] container. It it based on an existing image[4] to provide OpenCV support, but has no other dependencies.

C.2.3. Packaging

The server can be run locally from a terminal or remotely in a docker container, which makes it easy to launch the server at remote locations closer to the venue it is intended for.

C.3. Development

The source code for the server, test framework and admin clients can be downloaded from:

`https://github.com/ezet/crowdstream-server`

The source code for the mobile application can be downloaded from:

`https://github.com/ezet/crowdstream-client`

[3]https://www.docker.com/
[4]https://hub.docker.com/r/jjanzic/docker-python3-opencv/

D. Configuration

The server offers several configurable parameters, some of which are described in the sections below.

D.1. Server Configuration

Sequence length The length of the sequence, excluding the prefix. Must be long enough to assign each connected device a unique sequence, following the rules described in subsection 4.1.2.

Symbol frequency The symbol frequency, symbols per second, that devices will be asked to broadcast at. Must be $minimumcapturefps/3$ or less.

Camera Dimensions The resolution of the camera used for recording

Extend Capture Seconds How many additional seconds to extend to the recording duration, to compensate for network delay

Sample rectangle size The size of each rectangle, a device, used to map the source image onto the target devices. The average RGB value of all pixels within each rectangle will be sent to each device.

Sequence prefix The sequence prefix that is prepended to all sequences.

Max Init Rounds The maximum number of times to run the init sequence in an attempt to successfully map all connected devices

Warmup frames Number of frames used to warm up the camera. These frames will not be included in the stored recording.

Map Visible First When true, the algorithm will attempt to associate all currently visible devices with their tracks first, before attempting to associate previously occluded devices. Recommended setting is ‘True‘.

Max Symbol Streak The maximum number of times a given symbol can appear in succession in a sequence. Used to avoid long streaks of ‘0‘ or ‘1‘. A lower setting is preferable, but drastically reduces the entropy of any given sequence length. Recommended setting is ‘2‘ or ‘3‘.

Detection Threshold The global threshold value for object detection. Should be tuned specifically for the environment and lightning conditions. A lower value will detect more devices, but may also produce more false positives. A higher setting will produce less false positives, but will also increase the amount of false negatives. A very dark arena can use values closer to 200, where a bright venue should use values close to 250. Recommended setting is 200-250.

password The administrator password

D.1.1. Location settings

position acceptance radius The radius, in gps coordinates, for which to restrict device participation.

restrict device location Whether devices should be restricted by their location or not.

auto detect location Whether the server should attempt to auto detect its own location or not.

longitude Manually override longitude for server position

latitude Manually override latitude for server position

Detection parameters, detection.py

Tracking parameters, tracking.py