Online Multi-Person Tracking Using Feature-Less Location Measurements

by

Emad William Farag

B.S., Massachusetts Institute of Technology (2015)

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of Master of Engineering in Computer Science and Engineering at the MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 2016 [September 2016]

© Massachusetts Institute of Technology 2016. All rights reserved.

Signature redacted

Author ............

Department of Electrical Engineering and Computer Science

August 16, 2016

Signature redacted

Certified by ............... Dina Katabi

Professor of Electrical Engineering and Computer Science Thesis Supervisor

Signature redacted

Accepted by ............... Christopher J. Terman

Chairman, Master of Engineering Thesis Committee
EDITED BY

Robert W. Bement

ASSISTED BY

Mae K. Andrade

MIT Libraries

77 Massachusetts Avenue
Cambridge, MA 02139
http://libraries.mit.edu/ask

DISCLAIMER NOTICE

Due to the condition of the original material, there are unavoidable flaws in this reproduction. We have made every effort possible to provide you with the best copy available.

Thank you.

The images contained in this document are of the best quality available.
Online Multi-Person Tracking Using Feature-Less Location Measurements

by

Emad William Farag

Submitted to the Department of Electrical Engineering and Computer Science on August 16, 2016, in partial fulfillment of the requirements for the degree of Master of Engineering in Computer Science and Engineering

Abstract

This thesis presents a scalable real-time multi-object tracking system based on feature-less location measurements. The thesis introduces a two-stage object tracking algorithm along with a server infrastructure that allows users to view the tracking results live, replay old frames, or compute long-term analytics based on the tracking results. In the first tracking stage, consecutive measurements are connected to form short tracklets using an algorithm based on MHT. In the second stage, the tracklets are connected to form longer tracks in an algorithm that reduces the tracking problem to a minimum-cost flow problem. The system infrastructure allows for a large number of connected devices or sensors while reducing the possible points of failure. The tracking algorithms are evaluated in a controlled environment and in a daylong experiment in a real setting. In the latter, the number of people detected by the tracking algorithms was correct 83% of the time when tracking was done using noisy motion-based measurements.

Thesis Supervisor: Dina Katabi
Title: Professor of Electrical Engineering and Computer Science
Disclaimer

The work in this project was done in collaboration other students in our lab: Chen-Yu Hsu, Rumen Hristov, Thomas Zhang, and John Mikhail. While I was mostly responsible for designing and building the tracking algorithms and the server infrastructure, we did work on some aspects of this project together, sharing the same codebase, discussing ideas, and reviewing each others’ code.
Acknowledgments

I am tremendously grateful for having Dina as my supervisor. Her dedication to her work and to her students has been an inspiration to me. While she gave me autonomy to pursue the ideas I had, her leadership, mentorship, and her attention to details were essential to guide me through this project.

I have been very fortunate to work with an amazing group of people that have taught me a lot and helped me grow as a researcher, as an engineer, and as a person. I am grateful to have spent the last year working with Chen-Yu, Rumen, Zach, and Tommy.

I am grateful to my group mates, especially Fadel, Omid, Mingmin, Ezz, and Dinesh for allowing me to install a camera in their office to evaluate my project. I would also like to give special thanks to Fadel for introducing me to Dina and to the lab and for his mentorship at the start my program. I am also grateful to Driss for his help in setting up and running experiments.

I would like to give special thanks to John for reviewing this thesis and suggesting edits, and more importantly, for being a supporting brother and a great friend.

I would also like to thank my friends at MIT for their support and for making my years at the Institute some of the best years of my life. I’m especially grateful to my roommates Narek and Hayk, for putting up with me arriving home very late after long nights in the lab.

Finally, I must express profound gratitude to my family and thank them for always supporting me through the ups and downs and for encouraging me to aim high and reach where I am today.
# Contents

1 Introduction .................................................. 15  
   1.1 Overview .................................................. 15  
   1.2 Results .................................................... 16  

2 Related Work ................................................. 19  
   2.1 Related Work on Indoor Localization ...................... 19  
   2.2 Related Work on Multi-Object Tracking ................... 20  

3 Overview .................................................... 21  
   3.1 Tracking Algorithms ........................................ 21  
   3.2 Server Infrastructure ....................................... 23  

4 Short-term Tracking .......................................... 25  
   4.1 Overview And Definitions ................................... 26  
   4.2 Extending Tracks ........................................... 27  
   4.3 Kalman Filter ............................................... 28  
   4.4 Scoring ..................................................... 29  
   4.5 Pruning ..................................................... 31  
      4.5.1 Branch Elimination .................................... 31  
      4.5.2 Clustering ............................................... 33
7 Evaluation

7.1 Experiment Setup ............................................. 59
7.2 Evaluation of Long-Term Tracking .......................... 60
7.3 Evaluation Of A Real Deployment ........................... 62
List of Figures

3-1 System Pipeline Overview ........................................ 22
4-1 Pruning By Branch Elimination ................................. 32
5-1 Example of Tracklets and Tracks ............................... 38
5-2 Graph Representation Of A Tracking Problem ............. 45
6-1 Server Architecture .................................................. 52
7-1 Percentage of correct tracklet linking in long-term tracking as a function of
    the mean of the simulated off time between tracklets. ....... 61
7-2 Number of people detected by the long-term tracking algorithm as a func-
    tion of the mean of the simulated off time between tracklets. .... 61
7-3 Tracking Results From A Real Deployment ................... 63
Chapter 1

Introduction

1.1 Overview

In this thesis, we present a scalable real-time tracking system that is capable of tracking multiple people using feature-less location measurements. In our experiments, we used the WiTrack[3] device which generates measurements indicating the location of moving objects in the environment, but the algorithms and the system were designed to be general and could be used with other types of sensors or devices. The data from these devices are used by the tracking algorithms to provide both a real-time view of the tracking results and an enhanced but delayed version of them. The system is capable of handling many simultaneous long-running connections and it provides analytics on top of the tracking results.

A real-time indoor tracking system has useful applications in many domains including security, healthcare, smart homes, and space utilization. For instance, a tracking system for a WiTrack device deployed in a home could be used to uncover many patterns with medical significance especially for the elderly such as the number of hours that a person sleeps, when they go to bed, the number of times they visit the bathroom, and their general activity
level. It could also be used to detect falls[2] and automatically report them to the caregiver. This system could also be used as a layer in a smart-home system that would automatically turn the lights on and off or adjust the heat based on where people are. For space utilization, it could be used by large retail stores to identify high-traffic and low-traffic areas for optimal merchandise placement.

Tracking multiple people using motion-based location measurements is challenging for multiple reasons. First of all, there are no features associated with the measurements and that renders the association problem harder and makes identity switches among tracked objects more common. Second, if the existence of location measurements is conditional on the movement of the target, there could potentially be large gaps in time without any measurements generated by the device for that target if they remain largely static. We introduce a two-stage approach to the tracking problem. In the first stage, measurements are connected to form short tracklets, and in the second stage tracklets are connected to form longer tracks.

In addition to the tracking algorithms, we designed and implemented a server infrastructure that is reliable and that scales with the number of connected devices. The objective of this project was not to develop offline tracking algorithms, but to build a complete tracking system including long-running tracking algorithms and the infrastructure to support them.

1.2 Results

We implemented the tracking algorithms and we built and deployed the server infrastructure along with a website that allows users to see the results live, replay old results, or view long-term analytics based on the tracking data. We have already deployed multiple devices
that are connected to our system and some of them have been continuously sending data to
our servers for a few months.

We tested our algorithms both in a controlled environment with ground-truth tracking
results and in a real-life setting. We did experiments in a room equipped with the VICON
tracking system and compared the results to those of our algorithm. We also deployed a
WiTrack device in an office for 24 hours and evaluated the results by looking at the video
stream of a web cam mounted inside the room. The tracking algorithms correctly predicted
the number of people in the room 83% of the time even when measurements were not
continuous due to people being static for large periods of time.
Chapter 2

Related Work

This project is related to previous work in mainly two fields: indoor localization using radio-frequency waves and multiple object tracking. In this section, we will discuss the relevant works and how they are related to this project.

2.1 Related Work on Indoor Localization

Recent advances in RF-based localization techniques have allowed for accurate indoor localization of both wireless devices and people. To localize devices, some approaches used RSSI[7], RFID backscatter[26], and antenna arrays[14]. On the other hand, Wi-Vi[4] and WiTrack[3] show how RF waves could be used to localize moving people behind walls and occlusions even if those people do not have any wireless devices mounted on them. RF-Capture[1] can even capture an image of the human figure and use it to extract features that are used to train a classifier to distinguish between different people. In this project, we use WiTrack as black box that provides feature-less location measurements which we use as input to our tracking algorithms.
2.2 Related Work on Multi-Object Tracking

The problem of multi-target tracking is well-studied in both the radar and the computer vision communities. In computer vision, many of the algorithms are decomposed into two stages: detection and tracking[27]. In the detection stage, potential objects are detected in the image and their features are extracted. An appearance model is used to determine if the features of two detections belong to the same object[20]. The extracted features could be based on SIFT[21] (Scale-Invariant Feature Transform), a deformable part model[11] based on HOG features, or convolutional neural networks[28]. Some trackers, such as TLD[15], incorporate an online learning model that uses features from the detections and the background to continuously learn a classifier for the object. Most of these methods differ fundamentally from our approach because we assume that we do not have access to features from the underlying images and the only input to the algorithm are feature-less location measurements.

For the tracking stage, multiple approaches have been used in the literature. GNN (Global Nearest Neighbor) greedily matches observations to objects and is evaluated in [18]. An approach based on representing the problem as a bipartite graph to link measurements between consecutive frames is presented in [23]. Both of the aforementioned approaches solve the association problem on a frame-by-frame basis not over a larger period of time. MHT (Multiple-Hypothesis Tracking)[24], an influential work by Reid, presents a more optimal albeit intractable solution to the problem where multiple association hypotheses are maintained. JPDA (Joint Probabilistic Data Association)[13] has also been used to solve the association problem. An approach described in [29] represents the problem as a flow graph and solves it using a minimum-cost maximum-flow solver. In our project, we use a two-stage tracking system based on both MHT and the flow graph representation of the association problem.
Chapter 3

Overview

This project has two distinct components: the first consists of the algorithms used for tracking and the second component is the infrastructure that allows tracking and its applications to be done at scale. Figure 3-1 shows a general overview of the data pipeline in the system from the sensor to the end user.

3.1 Tracking Algorithms

The tracking problem consists of mapping the measurements generated by the device in each frame to trajectories that correspond to the people’s movement in the space even in the presence of missed detections (false negative) or incorrect detections (false positive). In our experiments, we used the WiTrack device which finds the location of moving objects in the environment. In contrast to multiple-object tracking as done the computer vision community, we assume that the measurements do not contain any features and that we are likely to stop receiving any measurements if the targets are motionless.

In our project, we divided the tracking algorithm into two separate stages. The first is short-term tracking that is run on the device while the second stage uses the tracking output
Sensor generates Location Measurements

Short-term Tracking

Live Tracking Webpage

Stream to Server

Longer-term Tracking

Replay Webpage

Analytics & Applications

Figure 3-1: System Pipeline Overview
of the first stage to perform longer-term tracking and is run on the server. The separation of the two tracking stages into one that runs on the device and one on the server is meant to reduce the amount of data transferred on the network which is especially expensive if the device is using mobile broadband. In principle, however, they could both be run on the device or on the server. Tracking results from the first stage only have a delay of a few seconds which allows users to view the results live. The second stage connects the tracklets from the first stage to give better results in exchange for a larger delay. The more accurate results from the second stage could then be used when running analytics applications on the tracking results or when replaying frames from the past.

The difference in the time scale between tracking measurements frame-to-frame versus connecting longer tracks makes a fundamental difference in the choice of the algorithms. We use multiple hypotheses tracking with a Kalman filter for the first stage and represent the problem as a flow graph in the second stage.

3.2 Server Infrastructure

We designed a scalable server architecture that processes tracking data streamed from the devices and allows users to view the tracking results in real-time or replay frames from any given point in time. The long-term tracking algorithm is run on the servers and its results could be used when replaying frames from a previous point in time to provide more accurate tracking. The system also supports a framework to generate long-term analytics on the tracking data. For instance, the framework allows users to quickly view analytics such as room usage or an overview of the number of people tracked over a period ranging from a few hours, weeks, or months. The system was designed and built with the goals of being scalable, reliable, and modular.
Chapter 4

Short-term Tracking

The tracking problem consists of correctly assigning the location measurements generated by the sensor to trajectories corresponding to the paths of the multiple people in the environment. The problem is challenging for multiple reasons: false-positive measurements, false negatives where a person doesn’t generate a measurement at their location, people blocking each other, secondary reflections from objects in the environment. In addition, if two people are close enough, it is not immediately obvious which peak corresponds to which person because the measurements are feature-less.

We use the Multiple-Hypothesis Tracking (MHT) framework which allows us to generate different hypotheses as to where people are when it is ambiguous to determine that definitively. The decision is thus delayed until a time in the future, usually a few seconds later, when more evidence is gathered.
4.1 Overview And Definitions

Every tracked object is represented as a set of tracks, where each track is a possible trajectory for the object:

\[ O = \{ T_i \} \]

Each track could be represented as an ordered sequence of sensor measurements, \( M_t \), and missed observations, \( \chi_t \) (false negative):

\[ T_i = (M_{t_1}, M_{t_2}, ..., \chi_{t_n}, ...) \]

Two tracks from two different objects are considered "in conflict" if they shared a measurement at any point in time. A hypothesis, \( H \), is a set containing exactly one track from each tracked object such that no two tracks in the set are in conflict, i.e. all the measurements in all tracks of a valid hypothesis are distinct.

\[ H = \{ T \mid \forall T_i, T_j \in H. i \neq j. \exists M_t. M_t \in T_i \land M_t \in T_j \} \]

At any given frame, the algorithm finds the hypothesis with the maximum score and the tracks in the best hypothesis are used to determine the position of the tracked object thus far. We define the score of a hypothesis to be:

\[ \text{score}(H) = \log \prod_{T_i \in H} P(T_i) = \sum_{T_i \in H} \log P(T_i) \]

Since each hypothesis contains exactly one track from each tracked object, defining the score of a hypothesis in terms of independent track probabilities allows us to treat every tracked object separately from other objects in the implementation. Not only does this allow us to have a cleaner and faster implementation without worrying about the joint probabilities of tracks, but it also allows us to approximate the solution using a linear-program.
solver with Lagrangian relaxation[8].

In the following sections, we will refer to the tracks in the best hypothesis at each frame as the "best-estimate tracks". Note that the best-estimate track for a specific tracked object might not necessarily be the track with the highest probability; it is the track chosen for that object in the hypothesis with the highest total score.

Every frame, new measurements are used to extend current tracks and update their probabilities, or create new objects with new tracks. Then, the optimizer finds the best hypothesis across tracked objects. Afterwards, the pruning stage is run to keep the total number of tracks from growing too large.

4.2 Extending Tracks

When a new set of measurements from the device is available, the current tracks from all tracked objects are extended using the new measurements. The algorithm does not attempt to make a binding decision by matching each measurement to its closest track. Instead, every measurement is matched to every track if the distance between them is below a certain "gating" threshold.

Every track is split into multiple copies and every copy is extended with each of the measurements, if they are within the gating threshold. Furthermore, one of the copies does not get assigned any measurements to account for the case where all the measurements are either generated by another person or are false positives. For example, if two new measurements $M^1_{in}$ and $M^2_{in}$ are within the gating threshold of a certain track $T$, then $T$ is
extended from the following representation in the current frame:

\[ T = (M_{t_1}, \ldots, M_{t_{n-1}}) \]

To the following tracks in the subsequent frame:

\[ T' = (M_{t_1}, \ldots, M_{t_{n-1}}, M_{t_n}^1) \]

\[ T'' = (M_{t_1}, \ldots, M_{t_{n-1}}, M_{t_n}^2) \]

\[ T''' = (M_{t_1}, \ldots, M_{t_{n-1}}, \chi_{t_n}) \]

Any measurement that is not within the gating threshold of any of the existing objects is used to initialize a new object.

To avoid the exponential growth of the number of tracks, some heuristics are used. For example, if one of the measurements assigned to a particular track is very close to the estimated position of that track but not close to any other track from other objects, then there is not need to split the track and create a copy with the false-negative \( \chi_t \) since that measurement most likely belongs to that object.

4.3 Kalman Filter

The Kalman filter[16] is an optimal estimator for linear models with Gaussian noise. It could be used to estimate the true state of a system based on noisy observations. We use a Kalman filter to estimate the state of the tracks for multiple reasons. First and foremost, as the name implies, it is used to filter the measurements that are received from the device to reduce the noise. Filtering is useful when viewing the tracking results so that the trajectories are smooth and not too shaky. We also use the Kalman filter to predict the position
of tracks in the upcoming frame based on the state model. We incorporate the predicted positions of tracks in the probability calculations as will be explained in section 4.4.

We use a variant of the Kalman filter known as the Interacting Multiple Model (IMM)[17]. It allows us to model the person’s motion using multiple models and estimate the person’s position using a weighted sum of the states from all models. In our case, we use two models to capture the person’s motion: the first is a constant velocity model and the second is a constant acceleration model. The IMM Kalman filter combines both models allowing us to estimate the people’s trajectories accurately when they are static, walking in straight lines, or when they are turning. Having the velocity and the acceleration as part of the state model allows for predicting the position in upcoming frame.

### 4.4 Scoring

As mentioned in section 4.1, in every iteration, we find the hypothesis with the highest score where the score of a hypothesis is defined as the logarithm of the product of its tracks’ probabilities. The probability of a track indicates how likely that track would have been generated by a person in the environment. For example, a track that has a sequence of measurements forming a smooth path will have a higher probability compared to a track that has a sequence of measurements without a clear structure.

If a track $T = (M_{t_1}, M_{t_2}, ..., M_{t_{n-1}}, M_{t_n})$, then the joint probability could be expressed as a product of conditional probabilities:

$$P(T) = P(M_{t_n}|M_{t_1}, ..., M_{t_{n-1}})P(M_{t_{n-1}}|M_{t_1}, ..., M_{t_{n-2}})\ldots P(M_{t_1})$$

Before defining the probability of a measurement given all the previous measurements,
we will first define the measurement-to-prediction distance, $r$. Let $r$ be the distance between the new measurement and the position predicted for the track by the Kalman filter given its previous state based on the previous measurements. If the model is completely accurate and the measurements are consistently devoid of noise, then $r$ would always be zero. In reality, we should expect the values of $r$ to be very small for a sequence of measurements that correctly represent a single person’s path.

We can now define the probability based on the measurement-to-prediction distance, $r$, and two constant parameters $a$ and $b$:

$$P(M_{t_n}|M_{t_1}, ..., M_{t_{n-1}}) = e^{-\frac{r+a}{b}}$$

For tracks where instead of a sensor measurement we have a false-negative assignment, $\chi_t$, we use the same probability function but with a constant parameter, $\delta$, in place of $r$. We call $\delta$ the equivalence distance because we set the probability of observing a false negative to be equivalent to observing a measurement at distance $\delta$ from the predicted position of the track.

We set the gating threshold to be larger than the equivalence distance $\delta$. If a single irregular measurement was at a distance greater than $\delta$ from a given track, then it will not be chosen by the optimizer to be part of that object. However, if that measurement was followed by a series of other measurements in that location, then the Kalman filter would eventually converge and the track with the measurements included will have a higher probability that the track with multiple false negative assignments.

If we take the logarithm of the product of probabilities over $n$ measurements, we would get the following equation (using $\delta$ in place of $r$ where appropriate):
\[ \log P(T) = \frac{a \cdot n - \sum_i^n r_i}{b} \] (4.1)

Since all the tracks of a given object have originated from a single measurement at a specific point in time, they are all going to have the same size, \( n \). Because of that, we could eliminate the \( a \) and \( b \) terms from equation 4.1 without altering the result of the optimizer. In essence, the stripped-down version of \( \log P(T) \) is a negative summation of the \( r \)’s and \( \delta \)’s resulting from every additional measurement in the track.

### 4.5 Pruning

If many measurements fall within the gating threshold of multiple tracks from multiple objects, the number of tracks will grow exponentially to account for all the possible combinations of measurement-track assignments. If we let the number of tracks grow exponentially, it will render the problem intractable. Therefore, pruning is essential to keep the total number of tracks below a certain threshold. In this section, we discuss the pruning methods we use. Some of the pruning methods are local, which means they are applied on the tracks of a single tracked object at a time. Others are global methods that work on the set of tracks from all objects.

#### 4.5.1 Branch Elimination

If we consider a track as a sequence of measurements, we can define the root of a track to be the measurement made \( N \) frames ago, where \( N \) is a predefined number of frames. In this pruning method, for each object, all tracks whose root measurement is not the same as the root of the best-estimate track of that object are eliminated. Any track that has diverged from the current best estimate for more than \( N \) frames is deemed no longer acceptable. Essentially, \( N \) is the maximum number of frames where uncertainty is allowed. Figure 4-1
Figure 4-1: **Pruning By Branch Elimination**: A track tree where a branch that had diverged more than $N$ frames ago is eliminated. In this example, $N = 2$. Each level of the tree represents a frame, every node represents a measurement, and every path represents a possible track. The branch with dotted lines is eliminated because it diverged for more than $N$ frames from the best-estimate track, $T_2^*$. 

shows an example of a track tree where one branch of the tree is eliminated.

When this pruning method is applied, it is guaranteed that for frames older than $N$ frames ago, every tracked object only has one acceptable "history" as all diverging hypotheses were eliminated. Thus, at each frame, the output of the algorithm is the delayed tracks from $N$ frames ago. We do not use the current positions of best-estimate tracks directly without delay because it is possible that, in the following frame, the best estimate track would change and the detected position of the person would "jump" to that of the other track. However, when the output is delayed by $N$ frames, we can be assured that track switching would not happen. In fact, branch elimination is essential to guaranteeing a stable and consistent output.
4.5.2 Clustering

Even if we prune tracks by eliminating diverging branches, the number of tracks could still grow exponentially in terms of the number of frames in the delay window, $N$. If $N$ is not very small, this could make the problem intractable. This is why other methods of pruning are needed. For every object, we use K-means to cluster the tracks based on their Kalman-filter-estimated positions and we only keep the track with the highest probability in each cluster. We also make sure that the best-estimate track of each object is not pruned in this step even if it does not have the highest probability because of a conflict with a track from another object. We use the K-means++[6] variant of algorithm because it leads to better cluster initialization.

4.5.3 Conflict-Based Pruning

This pruning method removes tracks whose descendants are guaranteed not to be selected as the best-estimate track of their object. Since the optimizer always selects one track for each tracked object in the best hypothesis, if a track is in conflict with all tracks of another object, then its descendants are also always going to be in conflict with all the descendants of those tracks from the other object. Hence, descendants of that track will never be selected as the best-estimate and it could safely be deleted.

4.6 Initialization & Termination

So far, we have not discussed how objects are initialized and how they terminated. Objects are to be initialized when a new person enters the coverage area of the device, or when a person is "lost" and in then re-detected again later. In chapter 5, we will discuss how to connect such objects that were likely generated by the same person. New objects are
initialized based on the log probability of their best-estimate track, but are deleted under a different criteria based on the number of frames for which the track was not assigned any measurements.

4.6.1 Validation

A tracked object is considered valid as soon as the log probability of its best-estimate track crosses a specific threshold, the minimum validation threshold. Once an object is validated, it is considered to be initialized and its position is included in the results. Objects that are not yet validated have not accumulated enough evidence and they are still considered as possibly an accumulation of false-positive measurements. Non-validated objects are treated differently from validated objects. For example, they are quickly discarded if their best-estimate track does get assigned to measurements for a couple of consecutive frames. Additionally, the optimizer does not apply the constraint of choosing exactly one track per object for non-validated objects. The optimizer is allowed not to assign any tracks to them in the best hypothesis if that leads to a higher score, which adds a bias in favour of the already-validated objects in case of a conflict.

4.6.2 Termination

Unlike initialization, probabilities are not directly used for deletion. A track that has not been assigned new measurements for a large period of time should still be deleted even if it had accumulated a high log probability because after a large time gap between measurements, the continuity of the track can no longer be assumed. One reason that an object would stop getting assigned measurements is that if the person was momentarily blocked while moving and is later considered as a new object to be initialized. In this case, it is essential not to keep the old object lingering for too long so that the long-term tracking algorithm described in the next chapter would be successful at reconnecting those shorts
tracks. Instead of using the track probability, we keep count of the number of frames that the track has not received new measurements. The object is deleted when that count reaches a few seconds for its best-estimate track.
Chapter 5

Long-term Tracking

The short-term tracking approach based on multiple-hypothesis tracking works by connecting measurements detected in successive frames, but is limited in nature because it has a delay window of only a few seconds and thus has to make binding decisions without the ability to wait for more measurements to be made. Additionally, increasing the delay will not necessarily lead to improved performance due to the need to prune the exponentially-growing number of tracks.

The problem is aggravated due to the lack of features in the measurements. For example, consider the case where a moving person is detected by the algorithm before coming to a halt. A minute later, the device detects a person that starts moving from around the same location where the previous person had stopped. It is not immediately obvious whether the two detected tracks belong to the same person or belong to two people sitting in close proximity. Having a global perspective on the scene by taking a long window in time helps make the distinction between the two cases.

The long-term tracking algorithm attempts to connect the shorter tracklets generated by the MHT algorithm on the device to create long tracks where each long track corresponds
Figure 5-1: **Example of Tracklets and Tracks**: The plot contains 4 tracklets from a short experiment with three people. Tracklets of the same color have been assigned to the same track. The two tracklets in green are assigned to the same track (person).

to the path of one person in the entire period of time. Figure 5-1 illustrates an example of two tracklets joined together to form one longer track.

### 5.1 Overview

The long-term tracking problem takes as input the tracklets generated by the previous tracking stage and its output is a smaller set of longer tracks where each track is a sequence of shorter tracklets. For each track, the person’s position is interpolated throughout the entire track period.

The problem is defined in terms of maximizing a posterior probability function. To make the problem tractable, we define the total probability as a product of probabilities that are functions of either one or two tracklets. This allows us to reduce the problem to a minimum-cost flow problem that has a polynomial-time solution in the number of tracklets.

We will initially assume that the whole set of tracklets is given as input, then in section 5.5, we will discuss how to slightly modify the algorithm to make it work in real time by having a continuous input and generating a continuous output with a fixed delay window.
5.2 Defining The Posterior

We are going to define the problem in terms of a posterior probability function that we will then maximize.

5.2.1 Definitions

- $S = \{ S_i \}$ is the set of tracklets generated in the previous tracking stage where each element $S_i$ is a single tracklet. A tracklet, $S_i$, is itself a sequence of time-tagged positions.
- $T_j = (S_i)$: a track, which is an ordered sequence of tracklets.
- $T = \{ T_j \}$ is the set of tracks, the output of the algorithm. A set of tracks, $T$, is considered valid if and only if each tracklet $S_i$ appears at most once in all of the tracks in $T$.

5.2.2 The Posterior

The goal is to find the set of tracks $T^*$ that maximizes the posterior:

$$ P(T|S) $$

By Bayes’ rule, we get:

$$ T^* = \arg \max_T P(T|S) = \arg \max_T P(S|T)P(T) $$

We now have to define both terms, then solve for the set of tracks that maximizes the posterior. If we further assume that the probabilities for tracklets and tracks are independent, we get:
\[ P(S|T)P(T) = \prod_i P(S_i|T) \prod_j P(T_j) \]

We can define the probability of certain track \( T_j \) to consist of the probability that the track starts and ends at the respective locations in addition to the likelihood of transition from one tracklet to another:

\[ P(T_j) = P((S_1, S_2, \ldots, S_n)) = P_{\text{start}}(S_1)P_{\text{link}}(S_2|S_1)P_{\text{link}}(S_3|S_2)\ldots P_{\text{link}}(S_n|S_{n-1})P_{\text{end}}(S_n) \]

The likelihood that a tracklet \( S_i \) is generated, given the set of tracks \( T \), could be defined in terms of the false positive rate \( \beta \):

\[ P(S_i|T) = \begin{cases} 
1 - \beta, & \text{if } S_i \in T \\
\beta, & \text{if } S_i \notin T 
\end{cases} \]

### 5.3 Defining The Probabilities

We have defined \( P(S_i|T) \) in terms of the false positive rate, \( \beta \). We now have to define the probability functions \( P_{\text{start}} \), \( P_{\text{end}} \), and \( P_{\text{link}} \).

#### 5.3.1 Link Probability

\( P_{\text{link}}(S_j|S_i) \) is the probability that two tracklets, \( S_i \) and \( S_j \), are connected in that order in one track, i.e. the probability that the same person generated the tracklet \( S_i \) then \( S_j \) in that order. We assign the probability to be zero if the \( S_j \) tracklet starts before \( S_i \) ends.

The reason that the same person could generate two consecutive tracklets instead of one long track could be that the person was temporarily blocked by another person while
moving or that the person was static for an extended period of time without being detected by the device.

We define the link probability to fall on a two-dimensional Gaussian distribution where one axis represents the time difference between the two tracklets and the other axis represents the distance between the end point of the first and the starting point of the second. Furthermore, we use a negative covariance between the two axes to further penalize linking tracklets that are far apart in both distance and time.

This approach is meant to be general and does not take the geometry of the space into account. If data on the boundaries of the rooms in the space is available, it could be factored in to make the probabilities more realistic: instead of using the distance between the two endpoints directly, we could first calculate the shortest path between these two endpoints given the room boundaries, and use that distance instead.

5.3.2 Start & End Probabilities

How do we make the algorithm more biased towards longer tracks? After all, if we are tracking people in a house or a small office, it is natural to assume that we would have a few persistent tracks representing a handful of people instead of a hundred shorter tracks belonging to different people. To encode that bias in the posterior, we could directly incorporate the length of the track in the probability $P(T_i)$. This approach, however, will not work because the problem will not be solvable in polynomial time anymore as it cannot be reduced to a min-cost flow problem. To bypass this problem, we will utilize both $P_{\text{start}}(S_i)$ and $P_{\text{end}}(S_i)$ to indirectly capture the length of the track.

Both of $P_{\text{start}}(S_i)$ and $P_{\text{end}}(S_i)$ have two components: one component that depends on
location and another that depends on the time of tracklet (starting point for $P_{\text{start}}$ and ending point for $P_{\text{end}}$). If the geometry of the space is available, then the location component of the probability is high when the tracklet starts or ends near an external door or near the edge of the coverage area. On the other hand, it would be lower if it’s in the center of the room. If the geometry is not available, then only the time component of the probability is used.

If we want to use the start and end probabilities to indirectly calculate a probability of the total length of the track, we need to find a function that relates the length of a given track to the product of the start probability of its first tracklet and the end probability of its last tracklet. In particular, we need to find a function $f(k)$ where $k$ is the total length of the track, $k = t_{\text{end}}(S_n) - t_{\text{start}}(S_1)$ such that:

$$P_{\text{start}}(S_1) P_{\text{end}}(S_n) = f(k)$$

We also require $f(k)$ to be increasing with $k$ such that longer tracks have a higher probability than shorter tracks. Hence, we define $P_{\text{start}}$ and $P_{\text{end}}$ as follows:

$$P_{\text{start}}(S_i) = e^{-\frac{t_{\text{start}}(S_i) + a}{c}}$$

$$P_{\text{end}}(S_i) = e^{\frac{t_{\text{end}}(S_i) + b}{c}}$$

The multiplication of $P_{\text{start}}(S_1)$ and $P_{\text{end}}(S_n)$ yields the following function in terms of the total length of the track, $k$:

$$f(k) = e^{-\frac{b + k + a + b}{c}}$$

If we know the minimum and maximum times for the start and end of tracklets, we can choose the values $a$, $b$, and $c$ so that $P_{\text{start}}$ and $P_{\text{end}}$ fall within well-defined bounds of our
choice, for example to guarantee that they are valid probability values between zero and one.

Furthermore, we could utilize the relationship between $f(k)$ and $\beta$, the false positive rate, to define a minimum track length. Given the probabilities defined in the previous section, a single tracklet, $S_i$, would be considered a false positive measurement instead of forming its own track if the following inequality holds:

$$\beta > (1 - \beta) f(k(S_i))$$

Therefore, we could set the minimum track length by choosing the values $a$, $b$, and $c$ such that they are under the following constraint:

$$\log f(k_{\min}) = \frac{k_{\min} + a + b}{c} = \log \frac{\beta}{1 - \beta}$$

### 5.4 Graph Representation

Now that we have defined the probabilities, we have to find the assignments of tracklets to tracks that maximizes the total posterior probability. We are going to define a graph such that each edge has an associated cost that is the negative logarithm of some probability function. In addition, each edge in the graph can have a flow of 0 or 1. We will use the flow constraints of the graph to guarantee that each tracklet would be assigned to at most one track. The total cost of the graph is:

$$\text{Total cost} = \sum_{i=1}^{\text{#edges}} \text{flow}_i \times \text{cost}_i$$
It is worth noting that if the cost of each edge is the negative logarithm of a probability value, then the problem of minimizing the total cost is equivalent to the problem of maximizing the product of those probabilities, or maximizing the total probability if they are assumed to be independent.

5.4.1 Nodes

The graph has two essential nodes: the source and the sink. In addition, each tracklet is represented by two nodes: an in node and an out node. All edges connecting to a tracklet will connect to its in node, and all edges leaving the tracklet originate from its out node.

5.4.2 Edges

- **Start & End:** the source node has an edge that connects to the in node of every tracklet with a cost of $-\log P_{\text{start}}(S_i)$. Similarly, The out node of every tracklet is connected to the sink with a cost of $-\log P_{\text{end}}(S_i)$.

- **Link:** the out node of every tracklet, $S_i$, is connected to the in for every other tracklet, $S_j$ that comes after itself. The cost of the edge is $-\log P_{\text{link}}(S_j|S_i)$.

- **Self:** the in and out nodes of each tracklet are connected by one edge with a cost of $-\log \frac{1-\beta}{\beta}$. Having only one edge between the in and out of each tracklet guarantees that the maximum total flow is one. Hence, each tracklet is guaranteed to be included in at most one track. If the tracklet has a flow of 1, the cost is $-\log \frac{1-\beta}{\beta}$. Otherwise, the cost is 0. This is equivalent to setting to cost to $-\log(1 - \beta)$ when the tracklet is used and $-\log\beta$ when it isn’t.

An example of such graph with two tracklets is presented in figure 5-2.
Figure 5-2: **Graph Representation Of A Tracking Problem** with two tracklets $S_1$ and $S_2$. All edges have a maximum flow capacity $= 1$. The goal is to assign each edge a flow (0 or 1) such that the total cost of all used edges is minimized while adhering to the flow constraints.
5.4.3 Solving The Minimum-Cost Flow Problem

The minimum-cost flow problem could be represented as a linear program with the following linear constraints.

1. The minimum capacity of each edge: \( \forall u, \forall v, f_{u,v} \geq 0 \)

2. The maximum capacity of each edge: \( \forall u, \forall v, f_{u,v} \leq 1 \)

3. The net flow of each node other than the source and sink: \( \forall w, \sum_u f_{u,w} = \sum_v f_{w,v} \)

Even though the flows must be integers, not real values, the min-cost flow problem is guaranteed to have an optimal integer solution even if that constraint is not imposed[5]. Thus, we can use any real-valued LP solver. There are also other algorithms to solve this problem optimally such as the network simplex[10] and the K-shortest paths algorithm[22].

To generate the set of tracks \( T \) from the solution, we can follow every positive-flow path from source to sink. All tracklets falling on the same path from source to sink are considered to be in the same track. Because of the maximum capacity constraint on the edges between the \( in \) and \( out \) nodes of every tracklet, we can guarantee that each tracklet will be in at most one track.

5.5 Window-Based Tracking

We have demonstrated how we can get an optimal set of tracks given a set of tracklets as input by defining all the probabilities involved, generating a flow graph, and solving it to retrieve the solution. However, this solution is not complete if we want to have a long-running tracking system since we do not have the whole set of tracklets as input at once; it is being generated in real time as new data comes from the device. Additionally, the algorithm must produce results within a maximum delay threshold. By using a fixed delay window,
other services get to know when the results are finalized and are ready to be processed for other applications. Thus, we have to slightly modify the algorithm to take that into account.

We are going to employ a sliding window approach. The window has a fixed time length, $W$. The tracklets under consideration are those that appeared within the time range $[t_{\text{now}} - W, \ t_{\text{now}}]$ where $t_{\text{now}}$ is the current time. New data that comes from the sensor at time $t_{\text{now}}$ is used to update the current set of tracklets under consideration. The other side of the window is the hard decision line. Any tracklet that appeared before $t_{\text{now}} - W$ has been already assigned to a track in a decision that can no longer be overturned.

As the window slides, incoming frames are used to update current set of tracklets. Whenever $t_{\text{now}} - W$ coincides with the beginning or end of a tracklet in the set under consideration, we convert the tracklets into a flow graph, run the optimization algorithm, and use the result to make a binding decision based on the following criteria:

- **If the tracklet begins at $t_{\text{now}} - W$:**
  - **If the tracklet was the beginning of a new track in the solution:**
    We make that decision binding in the future by setting the minimum capacity from the source to the tracklet’s $in$ node to 1. This will guarantee that any future solution must pass flow through that edge and make that tracklet the start of a separate track.
  - **If it was a continuation of a previous track:**
    We similarly make that decision binding by setting the minimum capacity of the edge between the $out$ and $in$ of the two tracklets to 1.

- **If the tracklet ends at $t_{\text{now}} - W$:**
  - **If the tracklet was the ending of a track in the solution:**
We remove that tracklet and all the tracklets that precede it in its track from the current set under consideration. We also mark that track as finished.

- **If it is in the middle of a track:**

  We set the minimum capacity of the corresponding edge to one to make the assignment permanent and we keep this tracklet in the set under consideration until its track ends.

In section 5.3.2, we argued in favor of defining the product \( P_{\text{start}}(S_l) \ P_{\text{end}}(S_n) \) in terms of a function of the total length of the track \( f(k) \). In fact, this is essential in making the sliding-window approach stable. It guarantees that as the window shifts, the probability of any given track remains constant, as long as its tracklets are not extended by new incoming frames.

### 5.6 Comparing Short-Term and Long-Term Tracking

In this chapter, we presented an approach to tracking that connects tracklets to generate longer tracks using an algorithm that is polynomial in the number of tracklets. We do not use this same representation and polynomial-time min-cost flow algorithm for tracking in the first stage because there is a fundamental difference between the input data in both stages. While it is reasonable to define the link probability of two tracklets as a function of only those two tracklets, the same argument cannot be made about the link probability of two device measurements. The individual measurements from the sensor are noisy which requires us to use measurements from multiple consecutive frames, not only two, when computing the score or the probabilities. We use a Kalman filter to estimate the noise-free position of a person from a sequence of measurements, taking velocity and acceleration into account, and use the Kalman-predicted position in the score calculations. However, in the long-term tracking stage, tracklets are already filtered and are long enough that we
can define the link probability between each pair independently which allows us to use the algorithm presented in this chapter.
Chapter 6

Server Infrastructure

We designed and built a server infrastructure that supports a deployment of a large number of devices which allows users to view the tracking results live, replay frames from the past, and view long-term analytics and statistics based on those tracking results. We implemented the different infrastructure services and deployed them on the cloud using Amazon Web Services. We used multiple programming languages and frameworks for the different services including Python, Go, C++, and JavaScript.

Figure 6 outlines the different infrastructure components and how they interact with each other. They are described in details in the following sections.

6.1 Design Goals

When designing the system, we focused on three main goals:

- **Scalability**: we should be able grow the number of connected devices without the need to redesign the server architecture and with minimal intervention from the system administrators.

- **Reliability**: it is very important to some of the applications that we envision our sys-
Figure 6-1: Server Architecture
tem will be used for that the system be reliable. For example, in security applications, users would expect the system to monitor their properties with minimal downtime. Similarly, for health applications, if tracking is conjoint with fall detection as shown in [3], care givers would expect the system to operate reliably. A few minutes of downtime during which a fall occurs could lead to serious implications if it goes undetected. Thus, in our design we attempt to minimize the points of failure so that the system would still function properly even if one service or one server crashes.

- **Modularity:** It should be relatively easy for system developers to add new features or applications without impacting the core services. For this reason, we avoided the monolith approach to server architecture and instead divided our services into independent modular components that could potentially be written in different programming languages and run independently. The components interact with each other through either a inter-server publish-subscribe queueing service or a REST API.

### 6.2 Load Balancer

A load balancer is a server that acts as a single access point receiving public traffic from the Internet and distributing the incoming traffic to multiple backend servers. It allows for balancing the incoming traffic so that if one server is busy processing older requests, new requests would be redirected to other backend servers that have more idle resources. The public DNS record for the system would only contain the IP of the load balancer instead of all the IPs of the multiple backend servers. This setup allows us to easily increase and decrease the number of backend servers in response to varying demand. To mitigate the fact that the load balancer has now become a single point of failure, we can instead use two identical load balancers and put both of their public IPs in the DNS record of the domain name. If one of them is down, the client (e.g. web browser or WiTrack device) would then
attempt to connect to the other working server.

### 6.3 Data Storage Layer

We use two database systems to store our data: Postgres[25] and Cassandra[19]. Postgres is a relational database with strong consistency guarantees and support for transactions. Cassandra, on the other hand, does not have a relational model and provides less consistency and serializability guarantees, but is highly scalable and has no single point of failure.

The Postgres instance is used to store relational data such as user accounts, user permissions, deployed devices, and so on. One large instance could hold tens of millions of rows, so we do not need to implement sharding for this database. However, we could setup master-slave replication so that if the master server crashes or becomes unreachable, the replica would provide an accessible read-only version of the database until the master is brought back up.

Cassandra is used for high-volume data where transactional guarantees are less important. More specifically, we use the Cassandra instance to store the tracking data that we receive from the devices at a rate of 25 frames per second per device. We also use it to store pre-computed analytics results that are cached for fast access when requested by the user. Cassandra provides linear scalability [9] which means that the total throughput is a linear function of the number of nodes.

### 6.4 Device Handler

The device handler service listens to connections from deployed devices and handles the incoming data by adding them to the inter-server pub-sub queue for other services to con-
sume and also stores them in Cassandra for later retrieval. The devices send tracking results to the servers in real time over long-running TLS-encrypted TCP connections. Some of the deployed devices use WiFi connections while others transmit data over mobile broadband. Thus, it was essential to use a compact data format to reduce the total amount of data transmitted per device. Each device accumulates frames for one second before sending them to reduce the TCP packet overhead.

Another service allows users to see the tracking results live by providing a WebSockets [12] interface. Compared to the standard HTTP protocol, WebSockets allows the browser (or a mobile app) to keep an open full-duplex connection with the server instead of polling the server a few times per second for new frames. It puts less strain on the server and provides more timely updates as new data become available. This service must be disjoint from the device handler service described earlier since it is not guaranteed that the user and the device are going to be connected to the same server behind the load balancer. Therefore, this service consumes the relevant frames from the queue, and sends them to the connected browser or mobile app.

6.5 Worker Services

Worker servers perform long-term processes that run in the background and do not depend on immediate user action. There are currently two types of worker services in the system: one is the long-term tracking service and the other is the analytics framework that pre-computes analytics and store the results for swift user access.

The long-term tracking service subscribes to the inter-server queue to receive new frames from the devices as they become available and use them to produce better, but delayed, tracking results. The new tracking results are stored in Cassandra for access by
users when they request to replay frames from a particular device at a particular point of
time in the past. They are also used by the analytics framework that reads them from the
database.

6.6 Analytics Framework

The analytics framework provides extensible modules based on the functional map/reduce
paradigm. A developer could add a new analytics module by defining a data model along
with the corresponding map and reduce functions. The map function maps a list of con-
tinuous frames to an instance of the defined data model, and the reduce functions takes a
sequence of those data objects and returns one object containing the results for the module
at hand. For example, to generate a heatmap that shows space usage over a specific period
of time, the map would generate a heatmap based on a list of frames and the reduce func-
tion would calculate a weighted average of a list of heatmaps to generate one final heatmap
representing the entire period.

At the end of every hour, for all devices, the analytics service gets an hour-worth of
frames from the database and runs the map function for each of the analytics modules. It
then stores the results, which are much smaller in size, in the database. Later on, if
a user requests to see the analytics results for a module for a given time period, the service
would run the corresponding reduce function on the cached results. This provides the
user with a much better experience because there would be little delay in generating the
results as most of the heavy lifting has been done earlier in the background by the map
functions.
6.7 Web Servers

Our web servers have two purposes. First, they serve the pages for the website along with static files such as images, CSS, and JavaScript files. Second, they provide a public REST API that is currently used by the web pages, but could also be used by future mobile apps or third-party developers. The API provides a single interface for user authentication, user operations, and accessing tracking and analytics data.
Chapter 7

Evaluation

We have implemented a real-time tracking system and deployed it conjunction with the WiTrack device in multiple locations. In this chapter, we present two sets of results. Initially, we will evaluate the two stages of the tracking algorithms independently in a controlled environment with ground-truth tracking data. Afterwards, we will evaluate tracking in a longer experiment from a real deployment in a student office in the MIT Stata building. For this deployment, we installed a web cam to capture a video of the scene in the office for manual evaluation.

7.1 Experiment Setup

To get ground-truth tracking data that could be compared to the output from our algorithms, we use the VICON tracking system. VICON is a tracking system that uses well-calibrated infrared cameras to accurately localize objects that are tagged with special infrared-reflective markers. The markers on each object have a unique geometry which allows the VICON system to distinguish them apart and localize the object with sub-centimeter accuracy. The infrared cameras in the VICON room that we used are able to localize tagged objects that fall within an area of $6m \times 5m$. 
We did multiple experiments in the room where each subject was instructed to wear a helmet mounted with infrared markers. During the experiment, test subjects were asked to sit, walk around, and interact with each others. We performed a total of three experiments, each lasting around 10 minutes for a total of 30 minutes.

### 7.2 Evaluation of Long-Term Tracking

The long-term tracking algorithm is used to link tracklets generated in the previous stage into longer tracks. In order to evaluate this algorithm independently of the performance of the previous stage, we will use the ground-truth tracks where each track is the correct path of one person in the environment. We will simulate "cuts" in those tracks by cutting the tracks in random places and use them as input to the algorithm. We then compare the result of the algorithm to the original tracks.

To cut the tracks, we will assume that every track could be considered as a series of on and off segments. The on parts are included in the input for the algorithm, and the off parts are excluded. The length of each of these on and off segments is a Gaussian variable. We will fix the mean of the on variable to be 60 seconds with a standard deviation of 60 seconds. We will try different values for the mean of the off variable. For all our experiments, we set the standard deviation to be equal to the mean used in that experiment. For each value, we calculated the total number of tracks detected by the algorithm (i.e. number of people) and we also calculated the percentage of tracks there were correctly linked by the algorithm. We ran each experiment 100 times for each value for the mean of the off and averaged the calculated results. The longer the length of the off, the harder the problem becomes, and the more likely the algorithm will make linking mistakes.
Figure 7-1: Percentage of correct tracklet linking in long-term tracking as a function of the mean of the simulated off time between tracklets.

Figure 7-2: Number of people detected by the long-term tracking algorithm as a function of the mean of the simulated off time between tracklets.
7.3 Evaluation Of A Real Deployment

We deployed a WiTrack device in a $7m \times 5m$ office and also installed a camera so that we can evaluate the tracking results in a real setting. The room has 5 desks and one couch. We ran the device with our tracking algorithms along with the camera for 24 hours. Since the device is able to detect people through walls, we limit our analysis to tracklets that correspond to movement within the room boundary. We cannot evaluate the tracking results outside of the room since no cameras were installed there.

Running the algorithms without enforced room boundaries resulted in:

- short-term tracking: 429 tracklets
- long-term tracking: 105 tracks

When the algorithms are run with enforced room boundaries, we get:

- short-term tracking: 201 tracklets
- long-term tracking: 34 tracks

We evaluated those tracks manually by looking at the images from the camera at fixed intervals of 15 minutes and comparing them to the results from the algorithms. Namely, we compared the number people in the scene to the number of tracks detected by the algorithm at the respective points in time. The number of people detected by the tracking algorithms was correct in 83% of the cases. We looked at the cases where the number of people was incorrect and identified three main points of failure and we suggest below how they could be circumvented:

- **Inanimate Objects:** Since the WiTrack device generates measurements by detecting movement in the environment, some of the detected tracklets were coming from
Figure 7-3: **Tracking Results From A Real Deployment**: 90 minutes of tracking results. Each segment is a tracklet, and segments of the same color are grouped together by the algorithm in the same track. In this subset of the results, there are three tracks. The tracks in red and blue were generated by the same person, but were not grouped together because the gap between them was too large. The single tracklet in dotted black is marked as a false positive because it could not be linked to any other tracklet to form a longer track.
moving but non-living objects. For example, a spinning fan or a wall-mounted clock would be detected by the device, especially if there is no signal interference from other moving people.

- **Leaving And Quickly Re-entering**: If a person leaves the room and re-enters after a short period of time, generating two tracklets, both of those tracklets would sometimes be connected together in one long track. The algorithm assumes that the person remained static near the door but was undetected throughout that period instead of assuming the person had left and re-entered. This would have been a valid assumption if it had happened in the middle of the room, but not near the door. This could be mitigated by taking the geometry of the room into account by increasing $P_{\text{start}}$ and $P_{\text{end}}$ near entry and exit areas.

- **Large Gap**: If a person remains static, generating two tracklets that are separated by a large period of time (e.g. 20 minutes or more), the tracklets might not get connected by the algorithm. Instead, it would generate two separate tracks. While this could potentially be mitigated by increasing the time variance in the $P_{\text{link}}$ function, doing this could lead to other problems especially if the room geometry is not used.
Bibliography


