Abstract
This thesis presents a system for long-term tracking of multiple people in indoor environments using a min cost flow algorithm to optimize the association of short-term tracklets. This system is built on top of an existing RF-based indoor localization system called WiTrack, which is able to track a number of people without requiring them to hold or wear any special devices. However, WiTrack relies primarily on being able to detect motion from a person to track him, and has its effectiveness limited by other factors such as obstructions and multipath effects. Consequently, WiTrack’s strength lies in tracking over shorter time intervals.

The system presented in this thesis utilizes these short-term tracklets produced by the WiTrack system and performs optimizations to try and account for missed detections either from lack of motion or from occlusions. This system is designed to utilize information about the indoor environment in which WiTrack is deployed in to make more informed decisions during the tracklet association process. To this end, an accompanying iOS application is built to aid in mapping the room layout during a deployment, and streamline the process for creating models for the indoor environment. In a two-week deployment in two separate environments, the system was able to reach precision and recall rates of 89% in predicting tracklet assignments, and a less than 3% rate for identity switching.

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

Some of the work in this project was done in collaboration with my group mates. In particular, using network flow to perform tracking was an idea I discussed with Emad Farag, another MEng student in the group. The work in this thesis expands upon some of these ideas and also focuses on taking into account information about the environment to improve tracking results.
Acknowledgments

I am extremely thankful to my advisor Dina for all of her advice, support, and help during my time working on this project. She gave me a tremendous opportunity and pushed for the best in me during my time here.

I would also like to give special thanks to all of my group mates and coworkers - Chen-yu, Zach, Rumen, Shichao, Emad, Mingmin, and Fadel - who have all at some point worked on the WiTrack project with me, for the inspired discussions and work environment.

Finally, I would like to thank my parents for their support throughout this time period.
# Contents

1 Introduction .................................................. 15  
1.1 Overview of WiTrack ........................................ 15  
1.2 Towards Long-Term Tracking ................................ 16

2 Related Work .................................................. 19  
2.1 Indoor Localization ......................................... 19  
2.2 Multiple Object Tracking (MOT) ............................ 19  
2.3 Approaches to the MOT Problem ............................ 20

3 Tracklet Stitching ............................................... 23  
3.1 MAP Formulation ............................................. 23  
3.2 Network Flow Formulation ................................... 25  
3.3 Edge Cost Estimation ....................................... 26  
3.3.1 Estimating Transition Costs ............................ 27  
3.3.2 Estimating Entry and Exit Costs ....................... 29

4 Room Model .................................................... 33  
4.1 Definitions .................................................. 33  
4.2 iOS Application ............................................. 34  
4.2.1 Streaming Location Information ....................... 34  
4.2.2 Information Format ..................................... 35  
4.2.3 Usability Features ..................................... 36  
4.3 Algorithms Involving the Room Model .................... 37
4.3.1 Shortest Path .................................................. 38
4.3.2 Boundary Construction ...................................... 40
4.3.3 Point in Polygon .............................................. 41

5 Evaluation .......................................................... 43
  5.1 Experiment Setup ................................................. 43
  5.2 Evaluation Metrics .............................................. 43
    5.2.1 Improvements .............................................. 46
List of Figures

4-1 Screenshots of the application running on an iPhone 7 Plus. . . . . . . . . 35
4-2 The issue with the intuitive shortest path algorithm. . . . . . . . . . . . . . 39
4-3 Visualization of the corner representation. . . . . . . . . . . . . . . . . . . 40
4-4 The issue with using the convex hull as the boundary. . . . . . . . . . . . . 41
4-5 Adjacent corners in the boundary. . . . . . . . . . . . . . . . . . . . . . . . 42

5-1 The room layouts for the two WiTrack deployments. . . . . . . . . . . . . . 44
5-2 Example tracklet stitching results for different scenarios. . . . . . . . . . . . 45
List of Tables

5.1 Classification of flow variables . . . . . . . . . . . . . . . . . . . . . . . . . 44
5.2 Classification of entry and exit flow variables . . . . . . . . . . . . . . 45
Chapter 1

Introduction

1.1 Overview of WiTrack

RF-based indoor localization systems have a number of potential applications ranging from home security to fall detection to augmented reality systems. WiTrack [1, 2] is one such system that is able to track a number of people to sub-meter accuracy without requiring them to hold or wear anything. It works across walls, spans a radius of 30 to 40 feet, and can concurrently locate up to 4 moving people. The lack of any sort of transmitter/receiver on the user’s end allows for far greater convenience and flexibility in tracking and enables many application possibilities, but at the same time places greater reliance on being able to differentiate the user from the environment.

WiTrack tracks the trajectories of individuals using RF reflections off of their bodies. It transmits a low-power signal (1000 times lower than WiFi), and then analyzes the signal it receives back after reflection off of the various surfaces in a room in order to estimate the locations of these reflectors. Reflections off of people are assumed to change over time as the person moves, unlike reflections off of other surfaces.

However, this reliance on motion in order to detect a person means that the tracking system that WiTrack uses cannot always produce confident results. In particular, if a person stops moving for an extended period of time, or becomes obstructed by a metallic reflector (such as a computer or TV screen) that blocks the signal reaching
the person, then the person’s location is essentially lost. This can occur often as a user moves around in the environment, or enters/exits the coverage area, causing the tracks outputted by WiTrack to often be short and intermittent.

1.2 Towards Long-Term Tracking

In order to provide long-term tracking, we build a system that uses the tracking results provided by WiTrack as input observations. We define a tracklet to be the position of a person over a single continuous period of time without being lost by WiTrack. A trajectory is a sequence of distinct tracklets that represent the position of a single person across a longer period of time, with possible periods of occlusion or untracked intervals in between tracklets. Based on the strengths and limitations of the WiTrack system as discussed previously, each tracklet tends to correspond to a period of motion from the person and is assumed to be reliable over the time periods that it spans.

Our approach for combining a set of tracklets into trajectories is to maximize the a posteriori probability of observing the set of tracklets conditioned upon a hypothesized set of trajectories. This formulation can be represented as a network flow optimization problem, where individual tracklets are mapped to vertices, and probabilities are mapped to edge costs, and then optimized using a min-cost flow algorithm.

Similar techniques have been used in computer vision to track moving objects and people across frames in a video [6, 7]. However, in an image or video, features of a detected object such as its color and size can be used as input and taken into consideration when determining edge costs, whereas the tracklets obtained from WiTrack contain only location data.

In order to make more informed decisions about tracklet associations in our scenario, we make use of information about the room layout. Thus, in our formulation, while edge costs are still largely dependent on the distance and time between tracklets, other information such as the position of the tracklet endpoints relative to doors
and the device’s operating range, or the locations of “static” regions in the room such as couches or beds is also incorporated.

The rest of the paper is structured as follows. In Chapter 2, we review previous work related to multi-object tracking. In Chapter 3, we describe the framework and implementation for the min-cost flow formulation for tracklet association, as well as how we estimate the network flow cost parameters. In Chapter 4, we discuss how the model of the indoor environment is created using the iOS application, as well as the implementation of algorithms related to the room model that are used by the network flow formulation. Finally, in Chapter 5, we go over the results of a two-week deployment in two different environments and evaluate the performance of the tracklet stitching algorithm using a number of different metrics.
Chapter 2

Related Work

2.1 Indoor Localization

There is a rich literature on indoor localization using radio signals. A large number of these systems localize users using signals emitted from their cell phones or other personal devices [12, 13, 14, 15]. Others use a large network of deploys sensors to localize users based on the received signal strength between different nodes [16, 17]. Unlike these systems, the equipment that WiTrack uses is limited to a single device, relying instead on RF signal reflections of the body to perform localization.

Regardless of the underlying implementation, the goal of connecting the individual localization observations produced by such a system over an extended period of time can be formulated as a multiple object tracking (MOT) problem.

2.2 Multiple Object Tracking (MOT)

Multiple object tracking (MOT), also known as multiple target tracking (MTT), is an important problem with a wide range of applications, ranging from surveillance to human behavior analysis. It has gained traction in recent years, especially in the field of computer vision. The goal of MOT is to be able to identify objects and track their positions over time, commonly called trajectories. The problem presents a number of key challenges, including dealing with sensor inaccuracies and missed
detections, having robust solutions to account for possible occlusions, and preventing “identity switching” between tracked objects, where one object becomes mistaken for another [9,11].

Some of the earliest applications of MOT were motivated by solving problems related to military surveillance and defense [11]. Object detection in these scenarios relied primarily on measurements from mechanical sensors, such as radar or sonar. In recent years, with advancements in computer vision, the focus has shifted more towards identifying objects in videos or image streams, where object detection relies on appearance models based on the shapes, sizes, and colors in an image [6,7,11].

In the context of RF-based indoor localization, the difference in difficulty between localization and tracking depends on the reliability and consistency of the localization results. For example, a system which uses signals from carried cell phones to perform localization has no issues with identity switching, but is still susceptible to occlusions and missed detections.

### 2.3 Approaches to the MOT Problem

One recent improvement towards the MOT problem is to separate the tracking task into two distinct stages: a short-term tracking stage which relies primarily on sensor inputs to produce tracklets, and a tracklet stitching stage which combines tracklets to form long-term tracks [9]. This tracklet stitching stage is often implemented as a network flow optimization problem to estimate the maximum a posteriori (MAP) probability of the data associations given a set of all observations [10]. The network representation means that the optimal network flow can be found in polynomial time [8,9,10]; however, this means that the input data must be available all at once, so that results are not necessarily obtainable in real-time [10]. The primary challenges of this network flow data association (NFDA) approach towards tracklet stitching come in the form of being able to estimate the parameters used in the flow network, typically edge costs.

The system presented in this paper uses the tracklets produced by WiTrack’s
tracking algorithm as input to a tracklet stitching problem formulated as a network flow problem. Much of the work done on the NFDA approach towards tracklet stitching has been in the context of computer vision [6,7,8]. One primary difference between the NFDA approach in computer vision applications and in the proposed system is that the only identifying features of tracklets produced by WiTrack are position over time, whereas tracklets detected from a video source can make use of additional features such as color and size. This information is important in determining transition costs and preventing identity switching; for example, if the pixels from one tracklet location are primarily red and another are primarily blue, then these tracklets should not be associated as they likely represent different objects.

Our system is designed with WiTrack’s use cases in mind, specifically for indoor tracking in settings such as houses or offices, but is applicable to any indoor localization system that can produce location estimates in the form of tracklets. This means that we are able to use information about the environment, obtained from a constructed model of the room layout, in the tracklet stitching algorithm to improve the quality of the tracklet associations.
Chapter 3

Tracklet Stitching

In this chapter we discuss the design of the tracklet stitching algorithm using the network flow data association (NFDA) approach. This is divided into three main sections: formulating the MAP probability of the data associations as a network flow problem, constructing the flow network, and estimating the edge cost parameters in the flow network.

3.1 MAP Formulation

Let $Z = \{z_i\}$ be the set of all tracklets generated by WiTrack over some time interval, where a tracklet $z_i$ is defined as a sequence $\{(x_t, y_t), \ldots, (x_{t+n}, y_{t+n})\}$ of $(x, y)$ positions of a detected object over some time interval $t$ to $t+n$. A trajectory hypothesis is an ordered sequence of tracklets $T_k = \{z_{k_1}, z_{k_2}, \ldots, z_{k_{l_k}}\}$, where the length $l_k \geq 1$ represents the number of distinct tracklets in the trajectory. A set of trajectory hypotheses $T = \{T_k\}$ is called an association hypothesis.

Given a set of input tracklets $Z$, our goal is to find the optimal association hypothesis $T^*$. Here, we define the “optimal” association hypothesis $T^*$ to be the hypothesis
with the maximum a posteriori (MAP) probability of $T^*$ conditioned on observing $Z$.

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

$$= \arg\max_T P(Z|T)P(T)$$

$$= \arg\max_T \left[ \prod_i P(z_i|T) \right] P(T) \quad (3.1)$$

where we assume that the likelihood terms $P(z_i|T)$ in (3.1) are conditionally independent.

The second term, $P(T)$, is fairly intractable as written; however, we can decompose this further if we make two additional assumptions:

1. Each tracklet is a part of at most one trajectory, i.e. each tracklet observation is due to a single person or object:

   $$T_k \cap T_l = \emptyset, \forall k \neq l \quad (3.2)$$

2. The motions of each trajectory are independent, so we can decompose $P(T)$ as:

   $$P(T) = \prod_{T_k \in T} P(T_k) \quad (3.3)$$

Using (3.2) and (3.3), we can rewrite (3.1) as:

$$T^* = \arg\max_T \prod_i P(z_i|T) \prod_{T_k \in T} P(T_k) \quad (3.4)$$

s.t. $T_k \cap T_l = \emptyset, \forall k \neq l$

The likelihood terms $P(z_i|T)$ correlate to the reliability of the observation generator. Namely, $1 - P(z_i|T)$ represents the probability that an observed tracklet is a false positive and hence should not be part of any trajectory. Since WiTrack’s tracking algorithm filters raw sensor observations to detect people primarily during periods of motion, we assume that $P(z_i|T) \approx 1$ in our model.
For the $P(T_k)$ terms, recall that each trajectory $T_k$ is comprised of a sequence of tracklet observations $\{z_{k_1}, z_{k_2}, \ldots, z_{k_{l_k}}\}$. We can model the transitions between these tracklets in the sequence as a Markov chain, with entry probability $P_{\text{entry}}(z_{k_1})$, exit probability $P_{\text{exit}}(z_{k_{l_k}})$, and intermediate transition probabilities $P_t(z_{k_{i+1}}|z_k)$ for $1 \leq i \leq l_k - 1$. We can substitute this into (3.4) to obtain our final expression for the data association using the MAP formulation:

$$
T^* \approx \prod_{T_k \in T} P(T_k) = \prod_{T_k \in T} \left[ P_{\text{entry}}(z_{k_1}) \left( \prod_{i=1}^{l_k-1} P_t(z_{k_{i+1}}|z_k) \right) P_{\text{exit}}(z_{k_{l_k}}) \right] \quad (3.5)
$$

### 3.2 Network Flow Formulation

In order to solve the MAP formulation in (3.5), we make use of the network flow data association (NFDA) approach of translating the MAP problem into a network flow problem.

Define a network flow $G$ with source $s$ and sink $t$. For each tracklet $z_i \in Z$, we create a corresponding vertex $v_i$, with an edge $(s, v_i)$ from the source with a cost $c_{si}$ and an edge $(v_i, t)$ to the sink with a cost $c_{it}$. For every pair of tracklets $z_i \neq z_j \in Z$, we add an edge $(v_i, v_j)$ with cost $c_{ij}$.

Given this graph construction, any unit of flow that passes from $s$ to $t$ necessarily passes through some sequence of distinct vertices $\{v_{i_1}, v_{i_2}, \ldots, v_{i_l}\}$. Thus, the path that this unit of flow takes corresponds to a trajectory containing the corresponding tracklets $\{z_{i_1}, z_{i_2}, \ldots, z_{i_l}\}$. In order to enforce the non-overlap assumption in (3.2), we can set the capacity of each edge to be 1, meaning that each vertex can admit at most one unit of flow from $s$ to $t$.

The cost incurred by a unit of flow passing through vertices $\{v_{i_1}, v_{i_2}, \ldots, v_{i_l}\}$ is

$$
c_{s,i_1} + \sum_{i=1}^{l-1} c_{i,i+1} + c_{i_l,t}
$$
We can relate this to the probability of a trajectory by taking the negative logarithm of the Markov expansion of $P(T_k)$ in 3.5. Thus the costs are defined as:

$$
c_{si} = -\log P_{\text{entry}}(z_i)
$$

$$
c_{it} = -\log P_{\text{exit}}(z_i)
$$

$$
c_{ij} = -\log P_i(z_j|z_i)
$$

It follows that to maximize the expression in (3.5), we need to minimize the total cost of the flow across the network, given by:

$$
T^* = \arg\min_T \left( \sum_i c_{si} f_{si} + \sum_{i \neq j} c_{ij} f_{ij} + \sum_i c_{it} f_{it} \right)
$$

(3.6)

where $f_{si}$, $f_{ij}$, and $f_{it}$ are indicator (0-1) variables representing whether the corresponding edge contains flow. To respect flow conservation constraints and the non-overlap assumption in (3.2), we enforce that

$$
f_{si} + \sum_j f_{ji} = f_{it} + \sum_i f_{ij} \leq 1
$$

### 3.3 Edge Cost Estimation

The problem of estimating the entry, exit, and transition edge costs in the flow network $G$ is important to the performance of the tracklet stitching algorithm. In the field of computer vision, a previously explored technique is to train a supervised learning model based on a large set of training videos [6, 7, 8]. While the design of the learning model varies between different papers, these models can all make use of image features such as shape, size, and color in addition to the location and time of detected tracklets. In contrast, the data obtained from WiTrack contains only the latter location and time information, meaning that any model trained on the input data alone will have less capability in distinguishing between tracklets of different sources.
This limitation can be somewhat alleviated by making use of information about the tracked area (henceforth also interchangeably referred to as the room) in which the WiTrack device is deployed. For instance, a tracklet that begins near the boundary of the tracked area is likely to be the starting point of a new tracked person, whereas a tracklet that begins in the middle of the tracked area is likely to be the continuation of a previously tracked and lost person. Implementing a supervised learning model using these room features is much more difficult, however, as the model would have to be trained separately for each deployed location, and objects and reflectors in the room such as furniture or television and computer screens that can affect the model and are not necessarily permanent fixtures.

Our approach to estimating the edge costs uses a heuristic-based approach using both the location and time information of tracklets as well as features from the room model constructed using the iOS application (discussed in more detail in the following chapter). This method avoids some of the difficulties in training individual models for each WiTrack deployment, instead allowing for a deployment-independent model at the cost of some potential accuracy.

3.3.1 Estimating Transition Costs

Transition edge costs between tracklets are modeled using a time-dependent Gaussian distribution based on the distance between the endpoint of one track and the startpoint of the other. Given a tracklet \( z_i \) with endpoint \((x_{t_1}, y_{t_1})\) and tracklet \( z_j \) with startpoint \(( x_{t_2}, y_{t_2})\), with \( t_1 < t_2 \), the distance \( d_{z_i,z_j} \) represents the shortest path distance between \((x_{t_1}, y_{t_1})\) and \((x_{t_2}, y_{t_2})\) with respect to the tracked region. This shortest path distance accounts for known obstructions such as walls, and is equal to the Euclidean distance between \((x_{t_1}, y_{t_1})\) and \((x_{t_2}, y_{t_2})\) if and only if there is a direct line of sight between the two endpoints. The method by which this distance \( d_{z_i,z_j} \) is computed is discussed in the following chapter.

We consider only pairs of tracklets where \( d_{z_i,z_j} \) is bounded by some gating threshold \( d_{gating} \). Recall our assumption that WiTrack is adept at detecting a person’s motion in the absence of metallic reflectors that obstruct the signal reaching the person.
The implementation of a gating threshold enforces the assumption that a person cannot be lost over an extended distance, as otherwise the person should have been detected by WiTrack. The choice of the gating threshold determines the ability of the tracklet association algorithm to deal with obstructions: a larger gating threshold allows for larger obstructions to be accounted for, but also increases the potential for erroneous tracklet associations. In practice, we find a gating threshold between 1.5 and 2 meters performs well.

For pairs of tracklets \( z_i \) and \( z_j \) bounded by the gating threshold, we make the assumption that if \( z_j \) is indeed the tracklet continuation of \( z_i \), then the set of observed shortest path distances \( d_{z_i,z_j} \) takes the form of a normal distribution centered at 0. Thus we compute the probability density at \( d_{z_i,z_j} \) as

\[
f(d_{z_i,z_j} | \sigma^2_{z_i,z_j}) = \frac{1}{\sqrt{2\pi \sigma^2_{z_i,z_j}}} e^{-\frac{(d_{z_i,z_j})^2}{2\sigma^2_{z_i,z_j}}}
\]

Here, \( \sigma^2_{z_i,z_j} \) is the variance of the Gaussian distribution, and is a function of the time difference \( \Delta t = t_2 - t_1 \) between the end of tracklet \( z_i \) and the start of tracklet \( z_j \). The reasoning for this is to capture the relationship between the uncertainty in a person’s location and the amount of time which has elapsed since the person was last detected.

If \( \Delta t \) is small, then we expect that the person was temporarily lost by WiTrack due to being static or being momentarily obstructed by a reflector. The uncertainty in the person’s position in this case is proportional to the possible distance that person could have covered in the time he was not tracked, so we expect the standard deviation \( \sigma_{z_i,z_j} \) to be proportional to \( \Delta t \). In particular, assuming that the average person has a typical indoor walking speed \( v_w \), then the distance he could have covered in the interval \( \Delta t \) is \( v_w \cdot \Delta t \). We choose a 2-standard deviation threshold to correspond to this uncertainty in distance, meaning that

\[
\sigma_{z_i,z_j} = \frac{v_w \cdot \Delta t}{2}
\]
The indoor walking speed \( v_w \) varies depending on person and environment, but we find that a value \( v_w = 1 \) meter per second works well in practice.

If \( \Delta t \) is large, then we expect the person was almost certainly lost by WiTrack due to being static. It is possible that the person’s location during this static period drifted slightly from his original ending location, but since this motion was not large or consistent enough, it was not detected as a new tracklet by WiTrack tracking algorithm. We again choose a 2-standard deviation threshold to correspond to the uncertainty in distance; in this case, we upper bound the maximum allowable drift by a parameter \( d_{\text{max}} \), so

\[
\sigma_{z_i,z_j} = \frac{d_{\text{max}}}{2}
\]

To enforce the piecewise continuity of \( \sigma_{z_i,z_j} \), the \( \Delta t \) threshold for switching between these two cases is given by \( \Delta t = d_{\text{max}} / v_w \), so we can express the standard deviation concisely as

\[
\sigma_{z_i,z_j} = \begin{cases} 
\frac{v_w \Delta t}{2}, & \Delta t \leq \frac{d_{\text{max}}}{v_w} \\
\frac{d_{\text{max}}}{2}, & \Delta t > \frac{d_{\text{max}}}{v_w}
\end{cases}
\]  

(3.7)

The probability densities \( f(d_{z_i,z_j} | \sigma_{z_i,z_j}^2) \) are then normalized across the incoming/outgoing edges for each tracklet to obtain \( \tilde{f}(d_{z_i,z_j} | \sigma_{z_i,z_j}^2) \), and so the transition edge costs \( c_{z_i,z_j} \) are computed as

\[
c_{z_i,z_j} = -\log \tilde{f}(d_{z_i,z_j} | \sigma_{z_i,z_j}^2)
\]

### 3.3.2 Estimating Entry and Exit Costs

Entry and exit costs are estimated based on a model of the tracked area as well as the technical limitations of the WiTrack device. To determine the location of an object relative to the device, WiTrack uses a combination of frequency-modulated continuous-wave (FMCW) radar to determine distance and antenna array steering to determine angle.
Detection Range Limitations

The detection range $d_{\text{range}}$ of the device is limited primarily by the signal to noise ratio of the received signal after reflecting off the object relative to noise from background radiation and antenna interference. The angle of the object is determined by combining the signals received by each antenna in a linear array after applying appropriate phase compensations. Due to the directionality of the antenna beam-form, angles can only reliably be interpolated between some $\theta_{\text{min}}$ and $\theta_{\text{max}}$, where $0 < \theta_{\text{min}} < \theta_{\text{max}} < \frac{\pi}{2}$. This means that the detectable region due to technical limitations is a circular sector with radius $d_{\text{range}}$ and central angle $\theta_{\text{max}} - \theta_{\text{min}}$. Typically, $d_{\text{range}}$ is on the order of 10 meters and the angle range parameters we choose are $\theta_{\text{min}} = \frac{10\pi}{180}$ and $\theta_{\text{max}} = \frac{170\pi}{180}$.

Room Model

The model for the tracked area consists of two main features that are used in determining entry and exit costs. These are the boundary for the entire tracked area, i.e. the region of interest, and the static regions such as couches, desks, or beds within the tracked area. The methods in which these are determined are described in the following chapter.

The boundary of the tracked area determines the possible locations in which a person could have entered or left the regions of interest. The intersection of this boundary with the circular sector that is the detectable region for the device determines the possible locations in which a person could have been first picked up/finally lost by WiTrack’s tracking algorithm.

A tracklet that starts at the boundary of this region likely corresponds to a new person entering the tracking area, either due to entering the room or entering the device’s detectable region. Conversely, a tracklet that starts in the middle of the tracked area likely corresponds to a person who was previously lost by WiTrack, since the person would most likely have been detected by walking from the boundary that position originally. We can draw analogous conclusions for tracklet endpoints.
Static regions are defined as areas where it is assumed that a person is likely to spend extended periods of time while exhibiting little motion. These can consist of locations such as couches, desks, or beds.

When compared to other regions in the interior of the tracked area, a tracklet that starts in a static region is more likely to be the continuation of a previous tracklet from a previously lost person, rather than the start of a new person’s trajectory. This reasoning takes precedence over the boundary heuristic, as a static region can be located near the boundary of the tracked area.

**Decision Tree**

Here we formalize the decision process made in determining the entry and exit costs. Let $D$ denote the circular sector corresponding to the detection area of the WiTrack device. Let $B$ denote the boundary of the room, with $S_1, S_2, \ldots, S_k$ denoting the static regions in $B$.

We can express the probability $P_s(i)$ that a tracklet $z_i$ corresponds to a new person being detected for the first time in terms of the following parameters arranged as a decision tree:

$$P_s(i) = \begin{cases} 
  p_S, & \text{entering at the location of a static region} \\
  p_D, & \text{entering at the limits of the detection range} \\
  p_B, & \text{entering at the boundary of the room} \\
  p, & \text{otherwise} 
\end{cases}$$

The entry cost $c_{si}$ is then taken to be the negative logarithm of this quantity, $-\log P_s(i)$. An analogous decision tree is constructed to determine the exit cost for each tracklet.
Chapter 4

Room Model

In this chapter, we describe the design and implementation of the iOS application that is used to streamline the process for creating the room model during a deployment. In addition, we cover the implementation of various algorithms used during the tracklet stitching process that involve the room model, including shortest path, boundary construction, and point in polygon.

4.1 Definitions

The tracked area in the room model can be thought of as a graph \( R = (V, E) \) superimposed upon the Euclidean plane consisting of a set of vertices \( V \) and edges \( E \). Vertices represent the significant points such as corners, and edges correspond to walls.

The boundary \( B \) of the tracked area is the polygon of minimal area whose vertices are a subset of \( V \) and encloses all vertices in \( V \).

In addition, the room model keeps track of a set of static regions \( \{S_1, S_2, \ldots, S_k\} \), which correspond to locations where a person is likely to remain relatively static for extended periods of time, such as couches, desks, or beds. Each of these is a polygon that is completely enclosed within the boundary \( B \).
4.2 iOS Application

The main goal of the iOS application is to provide a streamlined way to collect the location data necessary to create the room model during a WiTrack deployment. This involves recording the \((x, y)\) locations of the vertices in \(V\) and pairwise wall connections in \(E\).

Previously, this information was collected by repeated laser pointer measurements during deployment, a method which is both tedious and prone to human error. A key point during the design stage of the application was to maintain correctness of the location measurements while allowing for ease of use.

We cover here three main components of the design and implementation of the application: streaming location information from the WiTrack device itself, specifying and displaying in a more user friendly format, and usability features to allow for ease of use and error corrections.

4.2.1 Streaming Location Information

Our method of recording the locations of the vertices in the room model is to make use of WiTrack’s existing real-time localization and tracking algorithm to allow the user to walk around the room and record key locations in the application as he moves. To do this, the application needs to be able to interface with WiTrack and access this data in real-time.

While running, the WiTrack device streams the localized positions of the people it is tracking over a socket to a secure web server. The web server exposes an API by which an authenticated user can access the real-time information for a given device provided that the user supplies an appropriate access token.

In our application, the user is required to login with appropriate credentials for the device. The application then opens a socket to the web server using the returned access token, after which the location information can be streamed. A callback is provided to plot the \((x, y)\) data on the screen as the data is received. The user can then mark his current position as a point of interest via a button (Figure 4-1a).
4.2.2 Information Format

While the final room model is stored as a set of vertices $V$ and set of edges $E$ between those vertices, this format is not the most user friendly to input. In particular, denoting the location of each individual wall is tedious. Instead, the application allows the user to specify rooms by connecting a set of vertices, which then includes all of the corresponding walls. Entrances and exits to the rooms, i.e. doorways, can then be specified to denote that the corresponding walls should be split to account for the doorway.

Each marked location is stored as a Point object, and a set of points can be connected to form a Polygon object by drag-selecting the corresponding points in order (Figure 4-1b). These objects cross-reference each other so that a Point object can be dragged and moved and have all of its containing Polygons updated to reflect the move, and vice versa.

Each Polygon can be named and have other metadata added to it through an edit tab below the plotting area. In particular, each Polygon is denoted as either a room
or a region of interest (i.e. a static region).

Door objects are created by selecting a pair of Points and specifying the start and endpoints of the doorway in terms of fractions of the segment. The Door reference is added to any Polygon that has an edge containing the doorway.

Converting this representation into the data format used by the final room model is relatively straightforward. The set of all edges in all room Polygons is joined together to form a preliminary set of walls. Any Door object divides its containing edge into two separate edges with the doorway removed. Finally, any duplicate walls generated by this process are removed to give the set $E$ of edges in the room model. The set of vertices $V$ is simply the set of all endpoints of edges in $E$, and the set of static regions is simply the set of Polygons that have been denoted as such.

4.2.3 Usability Features

As screen space is a factor to consider in the design of the application and a limiting factor to a certain extent (depending on the iOS device), usability is a key consideration in the application design. To this end, we aimed for the core features to be accessible via simple gestures, and designed a way for users to accurately modify the locations of various objects without worrying about alignment issues or exact positioning.

Both Points and Polygons are selected and highlighted via a simple tap, and can be moved by holding until the color fades and then dragging to a new location. Polygons are created by dragging across a selection of Points without holding. The UI panel at the bottom of the application contains various tabs for adding, editing, or deleting objects.

Snapping Algorithm

While the primary tool for positioning Points is the real-time tracking data of WiTrack, this data may not always be exact, or it may be difficult or infeasible for the user to reach the desired position. At the same time, dragging an object on
the screen is inaccurate, especially since precision is needed to ensure points line up correctly.

We implemented an algorithm to snap Points and Polygons into place if they are dragged close enough. To do this, the algorithm computes an inferred location based on the relative positions of other objects on the screen, and snaps the moved object to the inferred location if the distance between them is below some threshold (Figure 4-1c).

We make the assumption that rooms tend to have multiple collinear vertices and walls tend to be at right angles to each other. Let \( P = \{p_1, p_2, \ldots, p_l\} \) denote the \((x, y)\) positions of the Points on screen at some point in time, let \( q \) denote the current location of Point that is being moved.

We say that a point is close to a line \( l \) if the distance from \( q \) to \( l \) (i.e. from \( q \) to its projection on \( l \)) is less than some threshold \( d \). If this is true, then we say that \( l \) is a possible constraint on the position of \( q \). We compile a set of constraint lines that could be imposed on \( q \) to snap it to a location:

1. For any \( p \in P \), consider the set of lines through \( p \) that form an angle to the Cartesian axes that is a multiple of \( 45^\circ \). If \( q \) is close to any of these lines, then we take the corresponding line as a constraint.

2. For any \( p_i, p_j \in P \), if \( q \) close to the line \( \overline{p_i p_j} \), then \( \overline{p_i p_j} \) is a constraint.

If there is exactly one unique constraint line \( l \), then we snap \( q \) to its projection on \( l \). If there are two unique constraint lines \( l \) and \( m \), then we snap \( q \) to the intersection of \( l \) and \( m \). Otherwise, we leave \( q \) untouched.

Constraints on Polygons are implemented in much the same way, except we additionally check for existing points that are collinear to any of the edges in the Polygon.

4.3 Algorithms Involving the Room Model

The tracklet stitching algorithm described in the previous chapter makes use of the room model to determine the edge costs of the flow network. In particular, shortest
path with respect to the room model is used to compute transition edge costs, and
distance to the boundary and static regions are both used in computing entry and
exit costs. In this section, we describe the implementation of the algorithms used to
answer these queries.

4.3.1 Shortest Path

The shortest path algorithm must take into account line of sight blockages and impass-
able paths that are caused by the positions of walls and corners. Since the shortest
path between two points in line of sight is a straight line, it can be shown that in
general the shortest path between points $p$ and $q$ passes through a subset of vertices
$v \in V$ such that each pair of consecutive points has line of sight.

Thus, the intuitive approach would be to augment the graph $(V, E)$ with points
$p$ and $q$ and add edges between any pairs of vertices with line of sight, and then find
the shortest path on the augmented graph using a standard shortest path algorithm.
This method does not always work, however, since it does not account for how corners
affect the reachability of vertices in the graph.

For example, in Figure 4-2a, the shortest path between the two points with respect
to the wall is to take the bottom path. However, if we add an intersecting wall at the
bottom so that it forms a corner, as in Figure 4-2b, then the shortest path between
the two points now must take the top path.

Corner Representation

Our solution to this problem is to create an expanded graph representation of the
room model where the vertices no longer represent single points, but instead represent
corners. A corner $(a, b, c)$ is defined by a center point $a$, and two endpoints $b$ and
c. Together, the corner represents the region of the plane spanned counterclockwise
from vector $\overrightarrow{ab}$ to $\overrightarrow{ac}$. Note that the $b$ and $c$ are not commutative; in fact if $b$ and $c$
are switched, then the resulting corner $(a, c, b)$ spans the remaining part of the plane
not spanned by $(a, b, c)$. 
Using this notation, we can define corner adjacency in the graph representation as follows. An edge exists between corner \((a, b, c)\) and corner \((d, e, f)\) if one of the following is true:

1. \(a = f\) and \(b = d\). This means that the leading edge \(\overrightarrow{ab}\) of corner \((a, b, c)\) is the same as the trailing edge \(\overrightarrow{df}\) of corner \((d, e, f)\).

2. \(a = e\) and \(c = d\). This means that the trailing edge \(\overrightarrow{ac}\) of corner \((a, b, c)\) is the same as the leading edge \(\overrightarrow{de}\) of corner \((d, e, f)\).

A point \(p\) is in line of sight of corner \((a, b, c)\) if the vector \(\overrightarrow{ap}\) lies in between vectors \(\overrightarrow{ab}\) and \(\overrightarrow{ac}\) and there is no wall that blocks line of sight between \(a\) and \(p\). This same rule can be extended to determine whether two corners have line of sight of each other.

Figure 4-3 exaggerates the display of the room layout in Figure 4-2 to highlight the corners in the corner representation. In general, if a vertex \(v\) has \(n\) incoming edges, then \(v\) is expanded into \(n\) corners, since each edge beyond the first subdivides the plane into an additional region. In Figure 4-3, there are a total of 6 corners, 3 at the original intersection of the two walls, and 1 at each of the 3 endpoints.

To implement the final shortest path algorithm points \(p\) and \(q\), we construct the expanded graph using corners as vertices and the corner adjacency rule to determine
Figure 4-3: Visualization of the corner representation of the room layout from Figure 4-2b. The exact positions of vertices have been modified/exaggerated for clarity.

draw new edges. We then augment the graph with points $p$ and $q$, and add edges between any points or corners in line of sight of each other. Finally, we can run any standard shortest path algorithm such as Dijkstra’s on this expanded graph.

4.3.2 Boundary Construction

As defined at the start of the chapter, the boundary $B$ of a room $R$ is the polygon of minimal area whose vertices are a subset of $V$ and encloses all vertices in $V$. In most cases, when the room layout is convex, the boundary is simply the convex hull of $V$.

However, when the room layout is not convex, we do not want to include the portion of the convex hull outside of the room layout as part of the boundary, as we do not want to consider tracklets that appear in that region during tracklet stitching.

Our solution to this problem again uses the corner representation of the room. If two corners are adjacent, then they necessarily share an edge, meaning that they span adjacent regions of the plane. We can use this idea to modify the convex hull to remove edges outside of the boundary.

We first compute the convex hull $H$ of the vertex set $V$. We assume that there exists at least one edge $e^* \in H$ such that $e^* \in E$ as well. Intuitively, this means that the room layout is at least “partially” convex, i.e. there is some portion of the boundary which forms a convex set, which is a reasonable assumption given how
(a) The layout of a room with a non-convex boundary.

(b) The convex hull of the vertex set (red) includes a region outside of the boundary.

Figure 4-4: The issue with using the convex hull as the boundary.

rooms are almost always laid out with orthogonal walls.

We iterate through the edges of $H$ in clockwise order beginning from $e^*$. If for some pair of consecutive edges $e$ and $e'$, we have $e \in E$ but $e' \notin E$, then we know that $e'$ is in the convex hull but not does not belong on the boundary. Since we know $e \in E$, there exists exactly one corner with leading edge $e$ and one corner with trailing edge $e$ that span adjacent regions in the plane. Because we are iterating through clockwise, this means that the corner with trailing edge $e$ follows directly after edge $e$, so we add that corner’s leading edge to the boundary following edge $e$. This process is repeated until we reach the original edge $e^*$, at which point the boundary will have been traversed fully.

One step of this algorithm is illustrated in Figure 4-5. Here, the corners (highlighted in red) are adjacent and span adjacent regions of the plane outside of the room boundary. When traversing clockwise, we reach an edge in the convex hull not on the boundary as in Figure 4-4b, so we select the edge based on the leading edge of the upper corner.

4.3.3 Point in Polygon

Determining whether a point $q$ lies within a polygon $P$ is used to determine whether a originates/terminates within the boundary of the room, or alternatively within the
Figure 4-5: Adjacent corners (highlighted in red) in the boundary that span the region outside of the boundary.

boundary of a known static region.

We use a standard ray casting/crossing number algorithm to count the number of times that a ray from $q$ to a point at infinity crosses the edges of $P$. Each crossing inverts the inside/outside status of a point, and since a point at infinity is definitively outside of $P$, this means that if the number of crossings is odd, then $q$ is inside $P$, and if the number of crossings is even, then $q$ is outside of $P$. 
Chapter 5

Evaluation

In this chapter, we present the results of running our system on data from real-world deployments of the WiTrack device. The trajectories outputted by the tracklet stitching algorithm on this data are compared with manually annotated trajectories and evaluated based on number of identity switches and tracklet compression. Furthermore, we compare the flow networks that correspond to each of these sets of trajectories and evaluate their similarity in terms of precision and recall rates.

5.1 Experiment Setup

We evaluated our system on data from two real-world WiTrack deployments. The room layouts for these deployments are shown in Figure 5-1. In total, we ran the tracklet stitching algorithm on approximately 20 hours worth of data, spread out in 1-hour length intervals across a 2-week time period. These hour intervals were chosen to be during periods of “high activity” relative to the norm, meaning that the number of tracklets was higher than average during these periods.

5.2 Evaluation Metrics

We manually annotated the tracklet assignments for approximately 20 hours of data in order to evaluate the performance of the tracklet stitching algorithm. The actual
In the first, we convert the trajectories (both the computed and manually annotated) back into their flow network representation. Since the flow network representation is dependent only on the tracklets and not on the tracklet assignments, the only difference between the computed and annotated flow networks are the binary flow variables (i.e. whether or not each edge contains flow). In this representation, we treat the flow variables as unknowns that our model attempts to predict, then measure the precision and recall rates. The aggregated numbers of true/false positives/negatives over all of the evaluated experiments are summarized in table 5.1. Based on these counts, we get a precision 0.887 and a recall of 0.891.

\[
\begin{array}{ccc}
\text{actual flow} = 1 & \text{predicted flow} = 1 & 2150 \text{ (true positive)} \\
\text{actual flow} = 0 & \text{predicted flow} = 1 & 264 \text{ (false negative)} \\
\text{actual flow} = 0 & \text{predicted flow} = 0 & 273 \text{ (false positive)} \\
\text{actual flow} = 1 & \text{predicted flow} = 0 & 184063 \text{ (true negative)} \\
\end{array}
\]

Table 5.1: Classification of flow variables.

It is important to note that these rates can be slightly misleading as the number of edges \(|E|\) in the flow network representation is on the order of \(|V|^2\), where \(|V|\) is the number of tracklets. The vast majority of these edges have zero flow, since any assignment of tracklets to trajectories sets \(O(|V|)\) edges to have unit flow, due to the non-overlap assumption (see 3.2).

To account for these, we evaluate the results based on another metric as well,
(a) Tracklet stitching run over an 8 hour interval while the person in red is asleep. The extraneous trajectories are from short interval tracklets in the living room area.

(b) Trajectories for simultaneous tracking of 2 people. The blue trajectory is a person sitting at a desk, while the red/green trajectories correspond to different parts of the body on a single person.

Figure 5.2: Example tracklet stitching results for different scenarios. In 5.2a, 133 tracklets are combined into 4 trajectories, while in 5.2b, 119 tracklets are combined into 3 trajectories.

namely the number of identity switches. An identity switch can occur in a number of ways; for example, a trajectory that belongs to a single person may be split into two, or two people may have their trajectories swapped at some point in time.

This can be difficult to account for all of the different cases, so we approximate the number of identity switches by the number of incorrect predictions among the entry and exit flow edges. The reasoning for this is that an identity switch either creates or deletes a false person, except in the case where two people have their trajectories switched. However, this case should occur far less commonly, as at the very least it requires two people to be in close proximity at some point in time.

<table>
<thead>
<tr>
<th>predicted flow = 1</th>
<th>predicted flow = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>actual flow = 1</td>
<td>78 (true positive)</td>
</tr>
<tr>
<td>actual flow = 0</td>
<td>32 (false positive)</td>
</tr>
</tbody>
</table>

Table 5.2: Classification of entry and exit flow variables

Out of a total of 2368 tracklets across all experiments, only 66 had an incorrectly predicted entry or exit cost, which is a ratio of 0.028.

Finally, we examined the average compression ratio of the tracklet stitching algo-
rithm. We define this to be the ratio of the number of predicted trajectories to the number of input tracklets. Each experiment had an average of 113 tracklets, with an average compression ratio of 0.061. This varies based on factors such as the complexity or the number of people in the experiment, but is a good estimate for the number of tracklet associations that our algorithm can make.

5.2.1 Improvements

The current main point of failure for our tracklet stitching algorithm occurs when two tracklets end up at approximately the same location. This does not occur often, but can occur in situations when two people cross paths, or as a result of noisy tracklets or inanimate moving objects that are picked up by WiTrack’s tracking system.

However, in the event this occurs, it becomes difficult to make correct tracklet associations after this point in time, since we have metrics or features to prefer one tracklet over another. The result is that there are often a large number of identity switches (the case we ignored previously).

The possible solutions to this involve filtering out the tracklets in question early, or in the case of two people crossing paths, hoping that their tracks will diverge at some point. If the locations of inanimate moving objects can be identified, then it would be possible to recognize tracklets originating from that source and ignore them for tracklet associations. If two people cross paths, then the assumption is that at some point in the future they will diverge, and up until that point, identity switching is acceptable since there is no other way to distinguish between the two given the accuracy of the location measurements.
Bibliography


