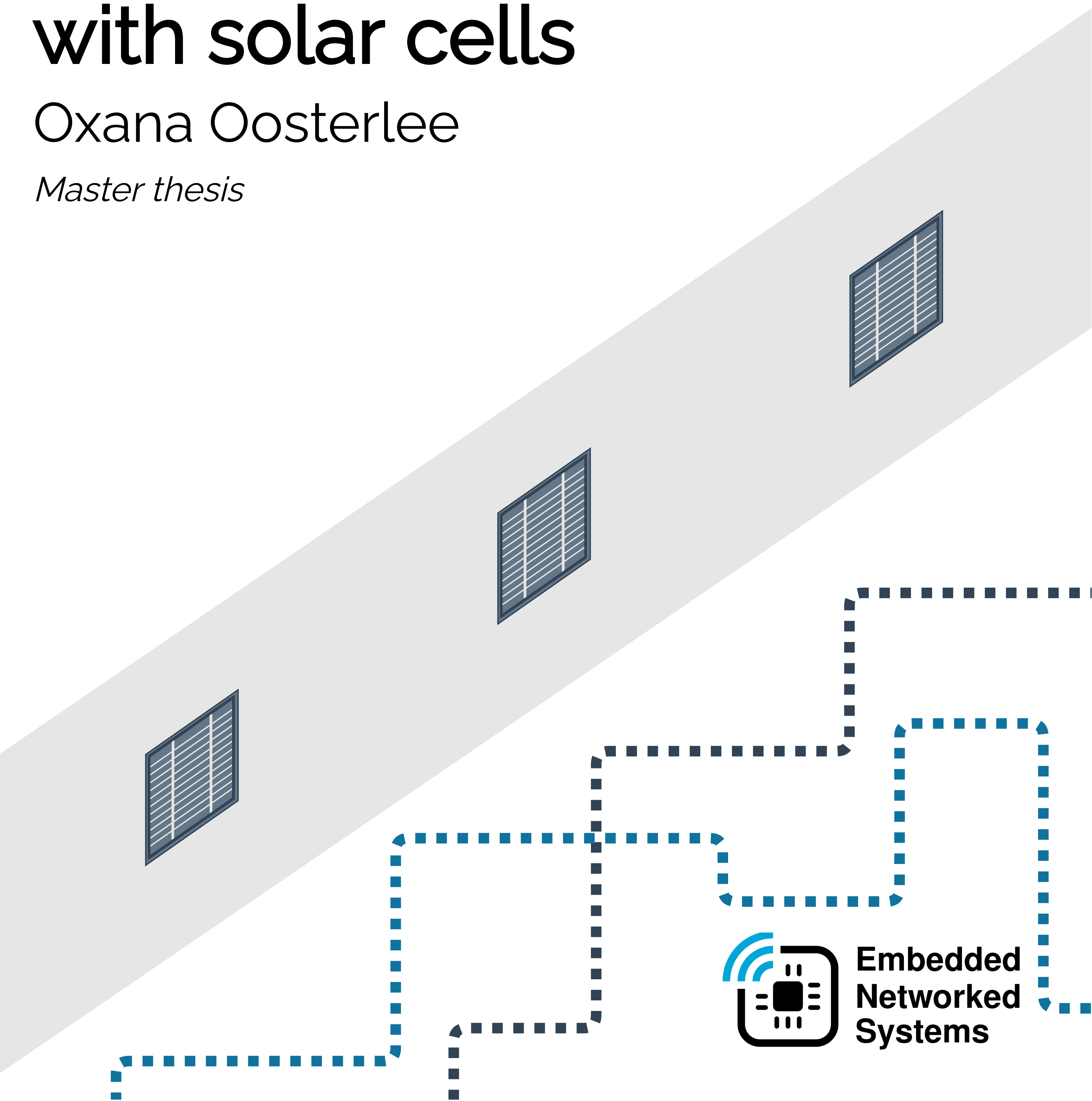
Delft University of Technology

Indoor multi-target tracking with solar cells

Oxana Oosterlee

Master thesis



Embedded Networked Systems

Indoor Multi-Target Tracking with Solar Cells

Master of Science Thesis in Embedded Systems

Embedded and Networked Systems Group
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands

Oxana Oosterlee

June 29th, 2021

Author
Oxana Oosterlee (o.oosterlee@student.tudelft.nl, oxanaoosterlee@gmail.com)
Student number
4374428
Title
Indoor Multi-Target Tracking with Solar Cells
Supervised by

ir. T. Xu	Delft University of Technology
dr. M. A. Zúñiga Zamalloa	Delft University of Technology

MSc Presentation Date
July 6th, 2021
Graduation Committee

dr. M. A. Zúñiga Zamalloa (chair)	Delft University of Technology
dr. C. Lofi	Delft University of Technology

Abstract

Solar cells are usually used as power source, but can be used for sensing as well. We propose a novel indoor tracking system that tracks people by using the change in output caused by their shadows. First, we develop a simulator to determine how the solar cells should be positioned in the tracking environment to get the best detection rate. This applies ray tracing in a model of the environment, and uses the standard deviation of solar cell output to compare different positions. Next, we apply changepoint detection methods to convert the solar cell output to a binary signal. One approach uses Bayesian online changepoint detection and another uses the change of gradient in the signal. Finally, the binary output from multiple solar cells is fused to track multiple targets in a real indoor environment in different scenarios. For this, we have implemented a particle filter based on the probability hypothesis density filter. We compare this with a tracking algorithm that uses a hidden Markov model. We have combined everything to show that it is possible to track up to two people in an indoor environment in different lighting conditions using a network of multiple solar cells.

Preface

This master thesis marks the end of my seven-year journey as a university student. First and foremost, I want to express my gratitude to Talia and Marco for their guidance throughout this process. Our weekly meetings truly helped me get through it, and I really valued your insights and feedback each time I was stuck in my own train of thought again.

The last year has definitely not been easy, having to work on a thesis from home during a pandemic. Luckily, I could count on all my 'colleagues' for support. I want to give a big thank you to all of my friends that have used my living room as an office. Because of you I have been able to keep my motivation, as well as my overall sanity. I do need to apologize for being a bad host, because I think that every single one of you has offered me tea in my own home. And not to forget, a very specific thank you to Emma and Eline for walking through my apartment repeatedly so I could do my experiments.

Last but not least I want to thank my family for everything. I am really lucky to always have your full support for literally anything I do.

Oxana Oosterlee

Delft, The Netherlands
June 28, 2021

Contents

Chapter 1

Introduction

1.1 Motivation

Indoor tracking is a broad topic with numerous applications. The basic principle of indoor tracking is that by using a network of sensors, the position of one or more human targets can be estimated as they move throughout an indoor space. This information can be used in smart environments. For example, it can be used in offices to regulate lighting and HVAC systems based on the number of people present to reduce energy consumption [20]. In smart homes for elderly, it can provide information about the health of the residents [54]. In stores, it can provide retailers with insights into consumer behavior [33].

Indoor tracking can be achieved by many different methods using various types of sensors. The design of an indoor tracking system is influenced by three design considerations: user participation, privacy and deployment infrastructure. Within these considerations, there is a growing emphasis placed on tracking systems that do not require user participation, preserve people's privacy, and are simple to deploy and maintain over large areas.

User participation in tracking means that people in the tracked environment are required to carry a device or tag that aids in the tracking. This can be a sensing device, an ID tag or a smartphone. However, in many cases this requirement is difficult to meet in practice since people may be unwilling to do so, or may simply forget. For *passive tracking*, people do not need to carry any devices. Sensors placed in the environment do all of the sensing. This can be, among others, a camera, infrared or ultrasound sensors [29]. In most circumstances, such as in stores and workplaces, passive tracking is preferable.

Due to privacy laws, tracking data is often required to be anonymous. Thus, a camera cannot be employed for many scenarios, even if it is a good approach to high-precision tracking. Many studies are therefore centered around the use of anonymous passive sensing methods. These methods often use low-cost binary sensors that only output whether or not a person is present inside their sensing range. Examples of binary sensors include passive infrared motion sensors [56, 66, 73], break beam sensors and contact switches for doors [76]. This type of sensor cannot be used to identify people individually, and can therefore be used for privacy-preserving tracking.

When deploying an indoor tracking system in a large space, a large number of devices is required to cover the entire area. As a result, the ease of deployment and maintenance, as well as the associated costs, must be carefully considered. The sensors' power supply is often a bottleneck. Connecting them to a building's existing electrical infrastructure is usually difficult to achieve, and battery-powered devices need costly replacements. The sensors should ideally be deployed with minimal modification to the existing infrastructure needed, and they should not require regular maintenance.

Considering the requirements listed above, we propose a novel indoor tracking method based on small solar cells. The solar cells are installed in an indoor environment. People walking by will cast shadows on the cells, causing a change of output. This data is used to track people walking through the environment. This solution is passive and preserves privacy because people cannot be identified by their shadows. Furthermore, given the primary role of solar cells as energy harvesters, the devices will be free of maintenance and infrastructure.

1.1.1 Goal

The goal of this thesis is to investigate whether solar cells can be used to develop a working indoor multi-target tracking system by designing a prototype. Additionally, we want to provide insights on the challenges that need to be overcome to reach this goal.

1.1.2 Contributions

This work has the following contributions:

- Design of a simulation framework with a graphical user interface to optimize solar cell configuration for people sensing in terms of placement height and angle.
- Identification and analysis of robust changepoint detection methods in order to extract a binary signal from a solar cell when a person walks by.
- Identification and analysis of two multi-target tracking algorithms for tracking more than one person in an indoor environment when noise is present in the sensor data.
- An experimental platform with a graphical user interface that allows for testing of the tracking system.

1.1.3 Organization

We start with discussing the state of art in Chapter 2. Then, the thesis is subdivided into three main topics. First, Chapter 3 describes the implementation of a simulation framework for determining optimal solar cell position for sensing purposes. Chapter 4 describes the implementation of changepoint detection algorithms for extracting a binary output from solar cells. Chapter 5 describes how the output from solar cells is combined to track multiple targets. As a proof of concept, the system is built and tested in a real-life scenario, which is explained in Chapter 6. Finally, conclusions and recommendations for future work are given in Chapter 7.

Chapter 2

State of the Art

In this chapter we discuss the state of the art in regards to tracking and using solar cells for sensing. First, we illustrate the different approaches to tracking in general. Then, we specifically look at tracking methods that use visual light sensing, which is what our system uses as well. Finally, we look at recent work on indoor solar cells and using them as sensors, which is an unique feature of our system.

2.1 Tracking methods

Tracking builds on localization, for which sensor outputs are used to determine the position of one or more targets. This data is used by tracking algorithms to determine the sequence of locations, or tracks, of moving targets. Different sensors can be used for passive, anonymous localization. Examples include ultrasound sensors and laser rangefinders, which also provide information about the distance of the target. Doorjamb [29] uses a system of ultrasonic sensors mounted in doorways to measure people's height to provide room-level localization and tracking. Sensors with binary output are also a popular option because they are typically inexpensive. They only provide information on whether or not they are detecting something and can therefore be used in situations where maintaining anonymity is required. Pressure mats, contact switches (for doors), and active infrared sensors are all examples of sensors that fall in this category. The passive infrared (PIR) sensor is by far the most commonly used type of binary sensor [11, 21, 67, 75, 76].

Single-target tracking algorithms form tracks by applying a filter on localization data to compensate for noisy or incorrect input. This can be accomplished by a variety of methods, such as by using a variation of the Kalman filter or particle filters. Multi-target tracking is an extension of single-target tracking that requires the use of more sophisticated algorithms or a wider range of deployed sensors. The data association problem is the main challenge of multi-target tracking. To create accurate tracks, new disjoint sensor events at time t must be matched to the disjoint sensor events occurring at $t-1$. In many cases this is solved by equipping the targets with ID sensors or using additional data [41, 64, 76].

Different algorithms exist for multi-target tracking. Classical approaches are multiple hypothesis tracking [63] and the joint probabilistic data association filter (JPDA) [7]. These are often not usable in large sensor networks because of exponential time and space complexities. Newer approaches often employ particle filters using methods such as the joint multi-target probability density (JMPD) [40] and the probability hypothesis density (PHD) [52]. There are also approaches based on Markov chains [42, 64].

Many examples can be found that use these algorithms, as well as variations or combinations thereof, to do multi-target tracking with binary sensors. For example, in [75] they propose an algorithm for multi-resident tracking in a smart home using a large number of deployed motion sensors throughout the house. It is based on a sequential Bayes estimation problem and does not need any prior information on sensor adjacency, but determines the relations between sensor directly from the input data. In [12], binary sensors are used in an office environment and a hidden Markov model is used to estimate the trajectories of multiple targets. In [55], multiple hypothesis tracking is used where a new hypothesis is generated using Bayesian filters every time a binary sensor is triggered to be able to track up to two targets in a residential home. TransTrack [38] uses multi-hypothesis tracking on zone transition data. FindingHuMo [12] uses a hidden Markov model to correct an estimated path based on probable state transitions. In [17] they apply two types of particle filters, an auxiliary particle filter and a cost-reference particle filter to do multi-target tracking with binary sensors.

In short, for multi-target tracking there is no definite best method. Many different ways exist to achieve the approximate same result. The key differences are in time and space complexity of the algorithms, whether the number of targets needs to be known beforehand, and the assumptions that need to be made about the underlying model. The challenge lies in adapting the chosen method to the needs of the specific application.

2.2 Visual light sensing

Visual light sensing (VLS) focuses on sensing objects by exploiting the influence of the object on visual light within the environment [14]. Visual light positioning (VLP) is a type of VLS concerned with finding the positions of targets in an area. VLP systems with no active users make use of shadowing. The system detects the presence of targets based on the shadows they cast on a light-receiving sensor. This has different applications such as reconstructing body postures using active LED panels and photodiodes such as in StarLight [46], LiSense [45], and Li-Tect [2]. In VLS, this sensing device can infer the position of a target from the received light. This can be a device that leverages the angle of arrival [79], RSS [27, 79, 81], Time Distance of Arrival (TDOA) [74] [36] and line of sight (LOS) blockage [31].

Many methods require the use of *active sources*. An active source is a light source that provides illumination, but has data modulated in the light as well by an adapted LED driver. The data can for example be an ID, such that the receiver is able to distinguish different light sources and use information about which light has been blocked. This can be done at a frequency invisible to the human eye, so no flickering is observed. In [34], the different LEDs can be identified using a TDMA-like scheme where the LED blinks in its unique time slot in a synchronized input signal. In [46], each LED blinks at its own unique frequency. In [58] photosensors are integrated into light bulbs to detect changes in the amount of reflected light. The light bulbs emit pulses according to a time division signalling scheme so that the detection system is able to distinguish different light sources. In [82] photodiode nodes are mounted in the floor. The LED transmitters each send a unique ID that can be detected by each node. It localizes a target based on the information on which node is shadowed by what transmitter.

Fewer examples exist for passive tracking, for which no modifications to the existing lighting infrastructure have to be made. One approach to this is *fingerprinting*. Fingerprinting requires an offline learning process, where the outputs of different receiver signal strengths are stored for different situations. When deployed, it uses the knowledge of these values to estimate the state of the current situation. In FieldLight [39], an array of wall-mounted photodiodes is used to track people. The system needs to be trained offline, where for each photodiode it generates histograms of signal strengths with a user present (at any location) and without a user present. In the online phase, it compares the received signal strength with the histograms to extract the position of one target walking in the area. Watchers on the Wall [22] uses a similar setup, and performs better. This is because in the offline phase for each node it learns histograms for specific target positions, which consequently requires more work. CeilingSee [80] determines the number of occupants in a room by modifying a ceiling

Name	Target	Source	Multi-target	Sensor type	Application	Remarks
Watchers on the Wall [22]	Passive	Passive	No	PD	Tracking	Requires fingerprinting
Fieldlight [39]	Passive	Passive	No	PD	Tracking	Requires fingerprinting
Eyelight [58]	Passive	Active	Yes	PD	Room occupancy	
Indoor intruder tracking [3]	Passive	Passive	No	PD	Tracking	Simulation only
Activity sensing [34]	Passive	Active	No	PD	Activity sensing	
Indoor position tracking [81]	Active	Active	No	PD	Localization	
Device-free localization [82]	Passive	Passive	Yes	Light intensity sensor	Localization	Requires grid of optical receivers on floor
Pedestrian detection [31]	Passive	Active	No	Undefined	Localization	Simulation only

Table 2.1: Overview of localization/tracking methods using visual light.

LED to use its photoelectric effect to detect reflections caused by occupants. It determines the number of users in a room by storing output voltages of different (known) occupancy levels and apply regression to infer occupancy for new situations. A disadvantage of fingerprinting is that it requires a lot of work to implement for each new environment. Furthermore, it is not very robust in situations where the lighting situation is not constant as the signal strength is not only influenced by the presence of targets, but by environmental light as well.

Table 2.1 gives an overview of previous works regarding tracking with visual light sensing. The majority only do localization (detecting where a target is), however these approaches could in theory be extended to tracking by applying tracking algorithms. Some works have only been tested in simulation, as is the case with [3]. This describes a single-user tracking algorithm using passive ceiling light, and multiple receivers in the floor. The algorithm uses a minimax filter that gives better results than a Kalman filter. The results are based on simulation and can therefore not be verified to work in real applications as well. From Table 2.1, it is clear that there is a lack of fully-passive tracking methods that are easy to implement. Furthermore, no tracking or localization method exist using solar cells as sensing device.

2.3 Solar cells as sensors

Solar cells are similar to photodiode (PD) type light sensors, as both are PN junctions. However, there are some differences as light sensors are optimized for light detection, speed, and certain wavelengths. For this reason, PDs often output a very small voltage that needs amplification before it can be used. Solar cells are optimized for producing power and not for sensing. This means their output is noisy and non-linear. Furthermore, solar cells are usually designed to work optimally for natural light wavelengths. However, in recent years there has been an increasing interest in improving solar cells for artificial light for indoor use [13, 49]. This includes a growing interest in organic solar cells for indoor use such as those described in [24, 43], which introduce new material compositions for organic solar cells optimized for artificial light that enable powering electronics indoors.

The ability to employ solar cells as a sensor has the advantage of providing simultaneous power generation and sensing. When the solar cell is appropriately sized, it will be able to supply sufficient power to charge the battery of the operating device. This is advantageous for mass-deployed systems since it avoids the need for battery replacements. There are few examples of solar cells being used as sensor. Examples include SolarGest [50], where a transparent solar panel is used to distinguish a number of hand gestures. Solar cells can also be used to receive uses solar cells to receive LiFi (light fidelity) communication signals, and leverage the harvested energy to improve the performance of an RF backscatter module [26]. In [71] a solar cell is combined with a comparator circuit to generate a a binary output without the use of an ADC. Events that cast a shadow on the panel, such as someone walking by or a gesture, are communicated as '0' and periods that do not have an event are communicated as '1'. This combined with RF backscatter results in a battery-free device. In uncontrolled conditions, however, the basic thresholding circuit used for generating the binary output is likely to be susceptible to false positives.

There has been a growing interest in designing solar cells for indoor use. However, very few examples exist that utilize indoor solar cells for sensing as well. The existing works show promising results, and provide motivation for exploring this topic further.

Chapter 3

Optimal configuration of solar cells

This chapter describes a simulator for determining the ideal configuration of a solar cell to detect people. The reason for designing the simulator is that in order to make an accurate tracking system, we need a high detection rate. When a target passes a solar cell, we should be able to reliably detect the target within a distance range required for the application. Therefore, we need to determine what configurations leads to a high detection rate.

We make the assumption that the solar cells will be positioned on walls, as this is often the most easy to install. This leaves two important configuration parameters to consider: height and pitch angle. What height and angle result in the best detection rate, depends on the layout of the environment. To make it possible to easily compare many different configurations with each other, we have created a simulator. The simulator enables comparing many different configurations by simulating the environment. For different scenarios and different solar cell configurations, it can determine the expected output by applying a ray tracing algorithm. This output is converted to a single score metric, which makes it possible to compare different solar cell configurations and select the one that is expected to perform best.

3.1 Ray tracing

Ray tracing models the behaviour of light rays and their interactions with objects in an environment. How a light ray interacts with an object, depends on its shape and material. Using these physical properties, it is possible to calculate the intensity of the light reflected by different areas of an object. For ray tracing, these properties are used to follow all rays for a given scenario to be able to calculate at what intensity they are perceived by an observer, such as a camera. Ray tracing is most notably used for rendering 3D models and environments for applications such as animation, game design and CAD product development [61].

There are two main approaches to ray tracing: deterministic methods and Monte Carlo methods. Using the deterministic method, all rays and their reflections in the environment are traced. Monte Carlo based approaches trace a random subset of the rays and use Monte Carlo integration to return an estimation of the reflections in the environment. Different approaches to the Monte Carlo method exist [28,48]. These methods decrease the amount of calculations required compared to the deterministic method at the cost of reduced accuracy. In our case, we use the deterministic method. It provides a simple and intuitive implementation of ray tracing. Previous work applying the deterministic model for simulating cast shadows can be found in [2, 3, 19, 50, 53].

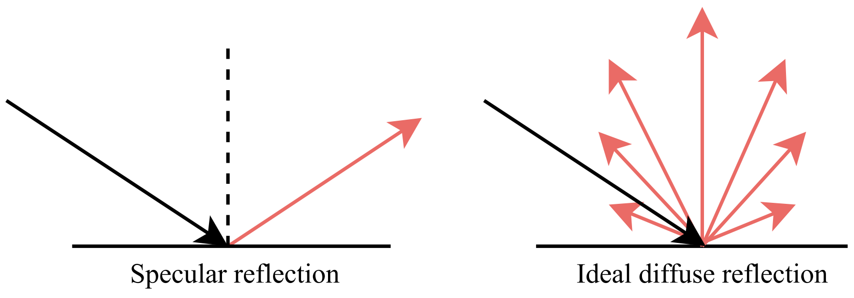


Figure 3.1: Types of reflection.

3.1.1 Reflection types

To be able to calculate the reflections of an incident ray, the reflection properties of light need to be considered. When a light ray hits a surface, part of the ray's power will be absorbed. Furthermore, depending on the surface's material properties, part of the power will also be reflected, transmitted (pass through the object), or both. For the simulator only reflectance is considered, as this is the most common property observed for large indoor surfaces and makes up the largest portion of the received light intensity by an observer. Following this logic, other phenomena such as transmission, diffraction and dispersion [15] are omitted as well to simplify the problem.

Two types of surface reflections can be distinguished: specular reflection and diffuse reflection. Figure 3.1 illustrates the different types of reflection. *Specular* reflection occurs when light hits a smooth surface such as steel or mirrored glass. According to Snell's law, the angle of incidence is equal to the angle of reflection. *Diffuse* reflection occurs on non-smooth surfaces such as a plaster wall, textile or paper. Due to the surface irregularities, light will be dispersed in all different directions. An ideal diffuse reflector will reflect light in a pattern described by Lambert's cosine law: $I = I_0 \cos(\theta)$, where I_0 is the incident ray's intensity, and θ is the angle between the surface normal and the exitant angle of the reflection. The law states that the reflected intensity is proportional to the cosine of the angle of reflection, independent of the angle of incidence.

3.1.2 Path loss and power calculations

In simulation, we distinguish three types of objects: sources (e.g., lamps), reflectors (e.g., walls, floors, humans and other obstacles) and receivers (solar cells). A ray originates at a source and travels to a receiver directly or via $n \geq 1$ reflectors. When traveling from one object to another, a ray 'loses' part of the original source power due to absorption and path loss. Below, the calculations for path loss L and received power P_r are described for all three object types, adapted from the work by [44]. For the calculations the following assumptions are made:

- All sources are ideal Lambertian sources.
- All reflectors are perfect diffuse reflectors and there are no significantly large specular surfaces.
- All receivers absorb 100% of the incident power and do not have any reflections.

Figure 3.2 show reflections and angles used in the equations.

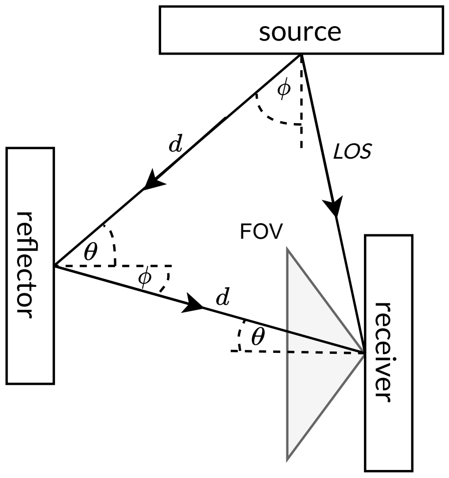


Figure 3.2: Light rays from source to a receiver. Both LOS and reflections.

Source

Given that the light source is Lambertian, the path loss from the source to a reflector or receiver in the scene is given by:

$$L_0 = \frac{(m+1)\cdot\cos(\phi)^m\cdot\cos\theta}{2\cdot\pi\cdot d^2} \tag{3.1}$$

With θ the angle of incidence (the angle between the target cell normal and the incident ray). ϕ is the angle of irradiance (the angle between the emitted ray and the normal of the source). The mode number m can be calculated using (3.2). $\phi_{1/2}$ is the semi-angle at half-power, which is a property of the light source. The value for $\phi_{1/2}$ is equal to half the viewing angle and describes the source's beam directivity.

$$m = \frac{-1}{\log_2(\cos(\phi_{1/2})} \tag{3.2}$$

The received power P_r from a ray originating at a source is calculated using:

$$P_r = P_s \cdot L_0 \cdot A_r \tag{3.3}$$

Where P_s is the power emitted by the source and A_r the area of the ray's target in m^2. The ray going directly from the source to a receiver is also called the line of sight (LOS) component.

Reflectors

Each reflector is assumed to be diffuse. Rays originating from a reflection of a diffuse reflector have the following path loss:

$$L_n = \frac{\cos(\phi)\cdot\cos\theta}{\pi\cdot d^2} \tag{3.4}$$

With θ the angle of incidence and ϕ is the angle of irradiance. Power P_r received by a target r from a ray originating from a reflector o can then be calculated using:

$$P_r = P_o \cdot L_n \cdot \rho_o \cdot A_r \tag{3.5}$$

Where P_o is the total power of the incident ray at its origin, P_r is the total power of the reflected ray, A_r is the surface area of the target and ρ_o is the reflective coefficient of the ray origin object. Note that in some cases $\rho_j(\lambda)$ can be used to specify a reflection coefficient based on the light wavelength. For simplification this implementation uses a constant value. The reflective coefficient of a surface depends on its material, texture and color.

Receivers

The receiver is an object in the scene where rays 'end'. The assumption is made that the receiver absorbs the power of all incident rays and there are no reflections. Only the power of rays of which the angle of incidence is smaller than the receiver's field of view (FOV) are absorbed. The total amount of power received by a receiver is the sum of all n incident rays that adhere to the FOV condition.

$$P_r = \sum_n \{P_r^n | \theta^n \leq FOV\} \tag{3.6}$$

Solar cell

The solar cell is a specific type of receiver with specific properties. The output voltage of a solar cell increases logarithmically with the incident light intensity (assuming equal temperature), whereas the current increases linearly [62]. The output of the simulation for solar cells is retrieved by taking the logarithm of P_r. Note that the simulation does not output the estimated current and/or voltage by the solar cell. This is a property of the type of solar cell used. For the metric used for determining an optimal configuration, we only need to know the relative change in output. Therefore, the actual solar cell characteristics apart from its surface area (as it influences the received power), are not taken into account.

3.1.3 Implementation

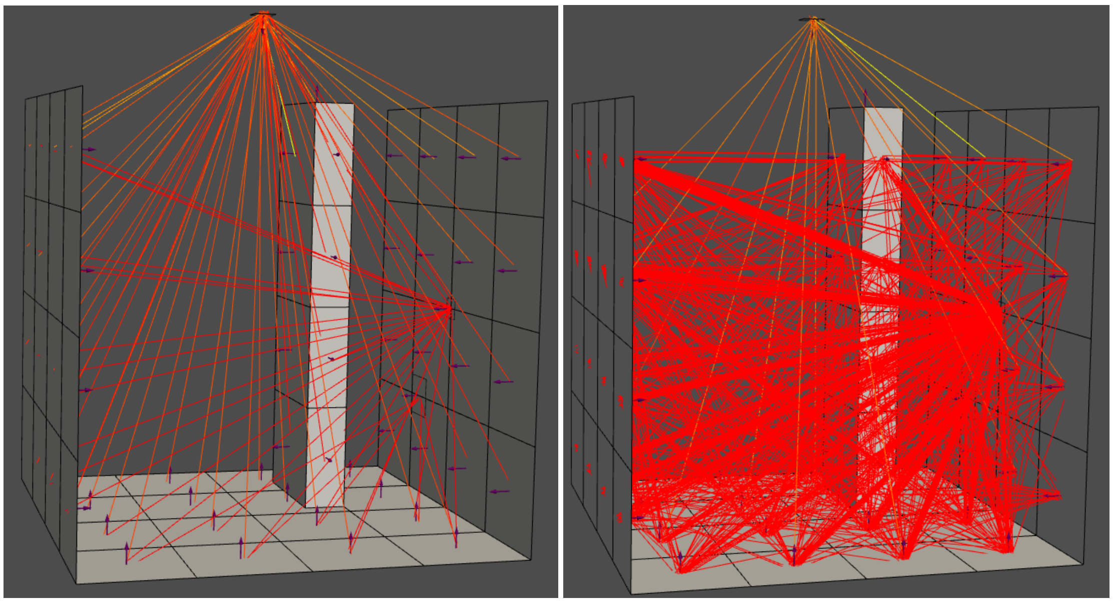
The above equations have been implemented in a simulator that traces rays recursively. All rays start a a source and are traced to all reflectors and receivers. The rays that target reflectors are traced again to all reflectors and receivers, and so on. The resulting traced rays for an example environment are shown in Figure 3.3. The full ray tracing algorithm can be found in Algorithm 1. To make it computationally tractable, a number of optimizations have been implemented.

First, all objects and surfaces are divided into cells. This is known as *meshing*. Smaller cell sizes give larger accuracy, but also increases the computational complexity n^2, as each ray and its reflections need to be traced to an increasing number of cells. To further reduce the number of computations, each ray is then traced to each receiver and to each 'reachable' cell of all reflectors in the scene. 'Reachable' means that (a) the target cell faces the originating cell, and (b) the ray does not pass through another obstacle in the scene to reach the target e.g., a ray cannot travel from a wall to an opposite wall, when an obstacle stands in between the two points.

The rays ending in a receiver are added to the total received power of the receiver. The rays targeting reflectors are traced further to all reachable reflectors and receivers until a predetermined number $N_{\max}$ of max to-be-traced reflections is reached. Furthermore, a value of maximum attenuation, L_{max} can be set. When the combined attenuation of a reflected ray exceeds this threshold, the contribution of this ray and its reflections to the final output is deemed marginal and thus will not be traced further. Finally, to keep the number of expensive vector calculations down for calculating the angles of incidence and irradiance, the results for these calculations are stored in memory. Because many rays follow the same path, this significantly reduces the number of calculations.

Algorithm 1: Ray tracing algorithm

```
Input:
    Set of sources $S$, set of reflectors $O$, set of receivers $\mathscr{R}$
    Maximum number of reflections $N_{\max}$
Output:
    The received output power for each $o$ in $O$
1:  $r.P \leftarrow 0 \, \forall r \in \mathscr{R}$          ▷ Initialize the received power of each receiver to 0
2:  R ←DetermineSourceRays($S$)          ▷ Initialize the first set of rays to be traced
3:  while R$\neg\emptyset$ do
4:      R ← TraceRay(ray ∈ R) ∪R
5:      R ← R \ ray
6:
7:  function DetermineSourceRays($S$)
8:      for every source ∈ $S$ do
9:          for every cell in $\mathscr{R} \cup O$ do          ▷ Loop through all reflector and receiver cells
10:             if not reachable then
11:                 continue
12:             $L_0 \leftarrow L_0(\phi, \theta, m, d)$          ▷ Calculate path loss using (3.1)
13:             ray.$P \leftarrow P(P_s, L_0,$ cell.area)          ▷ Calculate received power using (3.3)
14:             if cell ∈ $\mathscr{R}$ then
15:                 $r.P \leftarrow r.P +$ ray.$P_r$          ▷ Add ray power to targeted receiver sum
16:             else
17:                 ray.$n \leftarrow 0$          ▷ Initialize number of reflections
18:                 R ← R ∪ ray          ▷ Add new ray to to-be-traced because it is reflected.
19:     return R
20:
21: function TraceRay(ray, $O$, $\mathscr{R}$)
22:     R ← $\emptyset$          ▷ Initialize return to-be-traced rays
23:     for every cell in $\mathscr{R} \cup O$ do          ▷ Loop through all reflector and receiver cells
24:         if not reachable then
25:             continue
26:         ray$_{ref}$ ← new reflected ray
27:         $L \leftarrow L(\phi, \theta, d)$          ▷ Calculate using (3.1)
28:         ray$_{ref}$.$P_r \leftarrow P($ray.$P$,ray.target.$\rho, L,$ cell)          ▷ Calculate using (3.5)
29:         ray$_{ref}$.target ← cell,    ray$_{ref}$.$n$ ← ray.$n + 1$
30:         if cell ∈ $\mathscr{R}$ then
31:             $r.P \leftarrow r.P +$ ray$_{ref}$.$P_r$          ▷ Add ray power to targeted receiver sum
32:             continue          ▷ Don't trace ray any further
33:         else if ray$_{ref}$.$n \leq N_{max}$ or ray$_{ref}$.$P_r \leq P_{threshold}$ then
34:             continue          ▷ Don't trace ray any further
35:         else
36:             R ← R ∪ ray          ▷ Add new ray to to-be-traced
37:     return R          ▷ Return newly traced rays
```



(a) $n = 1$ **(b)** $n = 2$

Figure 3.3: Screenshot of the ray tracing simulation for a simple scene with one source, two walls, two receivers and one obstacle. Rays with high power are colored yellow, rays with low power are colored red. The simulation was limited for each ray to have a maximum of n reflections. A maximum of 1000 rays is visualized for performance reasons.

3.2 Procedure for selecting the optimal configuration

We can use the output data of the ray tracing simulator to determine the best configuration for solar cells in a specific environment. This is done by executing ray tracing iteratively as a modelled human obstacle passes by the solar cells to determine the amount of influence human shadowing causes on the output of different configurations. An overview of the procedure is given in Figure 3.4. It consists out of the following steps:

1. A 3D model of the environment is made. This model includes walls, floors, light sources and a human target modelled as an obstacle (see Figure 3.6b).
2. In the 3D environment, solar cells are added at a number of different heights and angles that are of interest (the numbered items in Figure 3.6b).
3. The human target is simulated in the environment as well, and moves through the environment along a predefined path. A common path would be to walk from one end to the other, at different distances from the solar cells. At each iteration dt, the target moves forward and ray tracing is executed. This calculates the expected received power for each solar cell.
4. After the target has finished moving along the path, the simulation run is finished and we know the expected power for each solar cell configuration for different target positions. Depending on the solar cell's configuration, the shade or extra reflections from the passing target causes different changes in the solar cell's received power. From this data, we give each solar cell configuration a different score based on a metric.

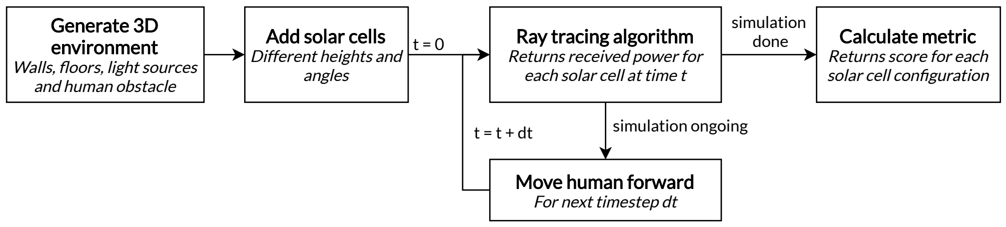

Generate 3D environment
Walls, floors, light sources and human obstacle
Add solar cells
Different heights and angles
t = 0
Ray tracing algorithm
Returns received power for each solar cell at time t
simulation done
Calculate metric
Returns score for each solar cell configuration
simulation ongoing
t = t + dt
Move human forward
For next timestep dt

Figure 3.4: Step-by-step overview of the simulation process for determining the best solar cell configuration.

3.2.1 Metric

Based on the simulation's output, we calculate a metric used to predict which solar cell configuration will have a greater detection rate than others. Different metrics can be calculated from the simulated solar cell received power, such as the variance, standard deviation and the median absolute deviation (MAD). The standard deviation was selected to give the best indication of a high detection rate. The reason for this is that when using the data to determine when a target is detected, this is usually done using methods that are based on detecting a change of distribution in the signal (this is further explained in Chapter 4). A larger standard deviation means that it is easier to detect changes in the signal. Standard deviation was selected over variance because it does not include the square term.

The standard deviation is calculated over the full length output of a 'run' for each different solar cell configuration. A run means that the target passes the solar cell from left to right, at a specific distance. We can execute the same run path for different distances between the solar cell and the target. This gives us information about the detection range of the solar cell as well. By finding which configurations have a relatively high value for standard deviation when the target passes at a large distance, we can maximize the detection range.

The metric value enables us to compare different solar cell configurations. Figure 3.5 shows an example of heatmaps that describe how the standard deviation metric changes for different positions, angles and target detection distances. To make it easy to compare different configurations with each other, we normalized the calculated metric value with respect to all calculated metric values (including the ones not shown in the figure). The normalized metric for each solar cell height is plotted. Both top plots show the results for solar cells placed at 0°, and the bottom plots show the results for solar cells placed at −45° (facing towards the floor). The difference between the left and right plot is the distance at which the target passed the solar cell for which the metric value was calculated.

It can be seen that for our specific test case a solar cell at an angle of −45° (facing towards the floor) is expected to perform better than one that is at 0°, as this gives higher metric values. Furthermore, when comparing the left and right plots of each configuration, we can see that cell configurations performing best for targets at a close distance do not necessarily perform best for targets passing by at a further distance. For example, when positioning the solar cell at −45°, placing the solar cell at a height of around 50 cm gives the highest expected detection rate for a target passing at 80 cm distance. However, when the target passes at 150 cm distance, this configuration does not give the best result. Instead, to be able to detect at a larger range the solar cell should be positioned around 155 cm. This illustrates how the simulation output can be used to optimally position solar cells to get the best detection probability for the required detection range.

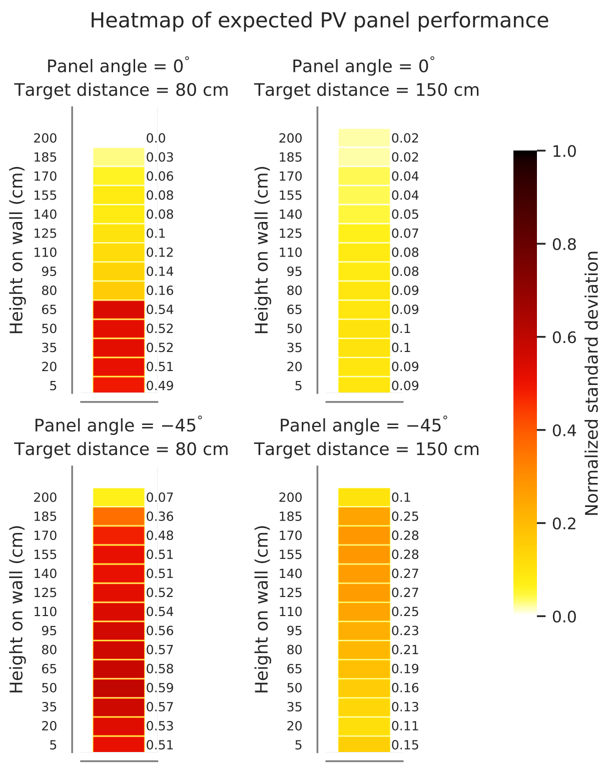


Figure 3.5: Heatmap showing the expected detection performance of a subset of all simulated solar cells. The values are shown for all different heights of two different angles and two different target distances. The performance is based on the normalized value of the standard deviation of simulated output. The higher this value, the better the expected detection performance.

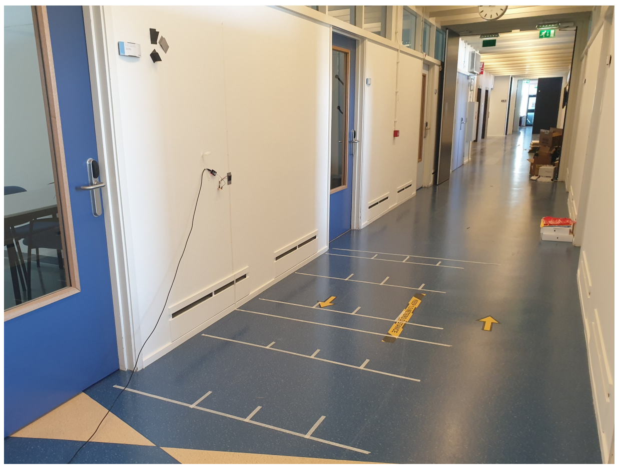
3.3 Validation

To verify the accuracy of the simulation, we execute a simulated scenario in real life and compare this data to the simulated results. The goal of the validation is to determine whether the expected performance of different configurations according to the simulator, corresponds with the performance in reality. The experiments were carried out in an office hallway. The validation process consists of three steps:

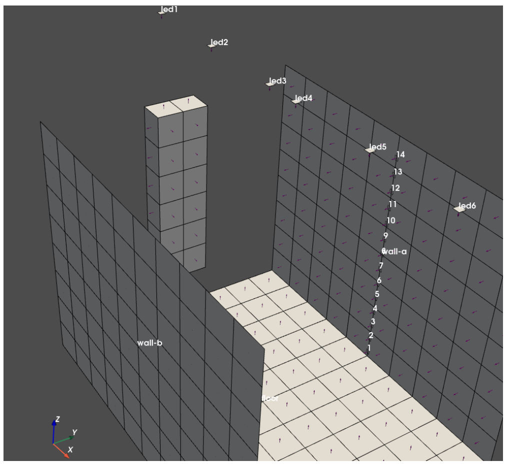
1. Model the hallway used for the experiment in the simulator. Then, run the simulator to calculate the metric from Section 3.2.1 for many different solar cell configurations and different target distances.

2. Using the simulation output, select eight configurations that will be used to perform the actual experiments with.

3. Perform the experiments with the selected configurations and compare the results of the simulation with the results of the experiments.



(a) Picture of the hallway during the experiments (with only one solar cell).



(b) Screenshot of the hallway in simulation.

Figure 3.6: Picture of the real hallway and the hallway in simulation (no rays traced). The objects in the simulation are meshed into cells of 30x30 cm. The small arrows show the surface normal of each mesh cell. The vertical numbers are the IDs of the solar cells located on the wall. During one simulation, the output of a cell can be determined for multiple cell angles. However, this is implemented in the back-end of the code and just one angle is shown in the render. The cuboid target, representing a person, can be directed to move forward through the environment.

3.3.1 Simulation

First, the actual hallway (walls/floor/light locations) was modelled in the simulator using reflection coefficients based on their materials and color [15]. A picture of the actual hallway and the simulated environment is shown in Figure 3.6. A human target modelled as a cuboid was directed to walk through the simulated environment from one end to another. On one wall, solar cells were positioned at different heights at different angles. Four ‘runs’ of the simulation were performed, with the target passing the solar cells from left to right at different distances (50, 80, 110, 150 cm). To keep the runtime of the simulation down, the maximum number of simulated reflections was limited to two. Increasing the number of simulated reflections does not cause a significant change in results because the contribution of higher-order reflections to the total received power is marginal. The resolution of the mesh was set to 30 cm.

3.3.2 Configuration selection

Only a subset of the simulated solar cell configurations was selected to be used in the the real-life experiment. The selection was based on the previously described metric score that is calculated from the simulation output. As with the heatmaps, we normalized the score for easy comparison. Based on the score, we rated the expected performance of each solar cell ‘good’, ‘medium’, ‘medium-low’ and ‘low’, based on the normalized metric value (the higher the better). For example, solar cells with a metric score ≥ 0.9 for a target distance of 50 cm would be rated ‘good’. For each rating, two cell configurations were selected. This results in a total of 8 selected configurations. Table 3.1 shows the selected configurations and the calculated metric for different target distances. We can see that solar cells placed at a ‘good’ configuration are expected to perform consistently better than other configurations, for any target distance. Medium and medium-low configurations are expected to perform

Table 3.1: Normalized standard deviation calculated from the simulation output for selected solar cell configurations. The target distance y is in cm.

				Metric value for different target distances y			
				y = 50	y = 80	y = 110	y = 150
	Rating	**Height** (cm)	**Angle** (°)				
Solar cell config.	Good	50	-45	0.95	0.61	0.31	0.15
		35	-25	1.00	0.63	0.31	0.18
	Medium	110	-10	0.80	0.15	0.13	0.08
		35	0	0.88	0.58	0.15	0.08
	Medium-low	140	-5	0.79	0.09	0.08	0.04
		125	0	0.81	0.09	0.08	0.05
	Bad	200	-45	0.00	0.05	0.11	0.09
		185	-10	0.01	0.02	0.03	0.02

equally at closer distances, however at further distances the medium-low configurations are much worse. Bad configurations give a bad performance overall.

3.3.3 Experiment

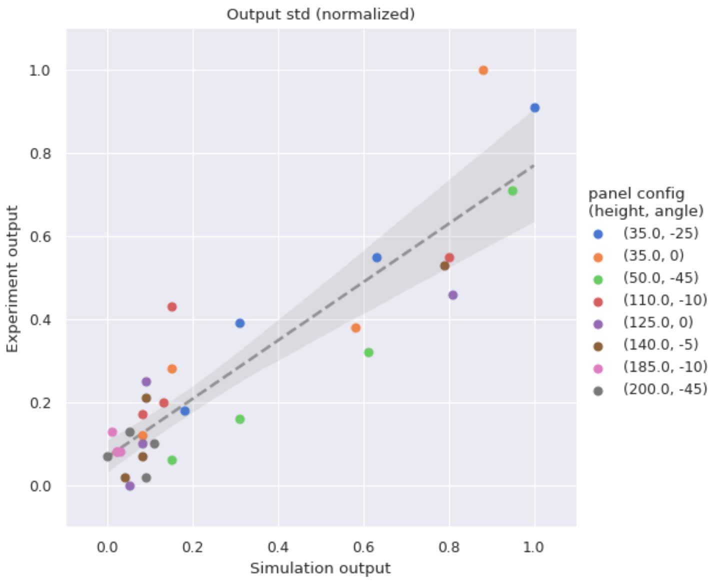
In the hallway used for the experiment, the solar cells were positioned according to Table 3.1. The solar cell output was recorded while a person walked past the solar cell. To keep the the same timing as in the simulation, a metronome was used. The data was filtered using a 5 Hz low-pass filter (see Section 4.1). Similar to the simulation, the calculated metric for each run/configuration was normalized over all calculated metrics. The reason for this, is that the relationship between solar cells is important, and not the absolute value of the output. The results can be used to validate whether the relationship between solar cells in simulation is similar to that of the experiment.

3.3.4 Results

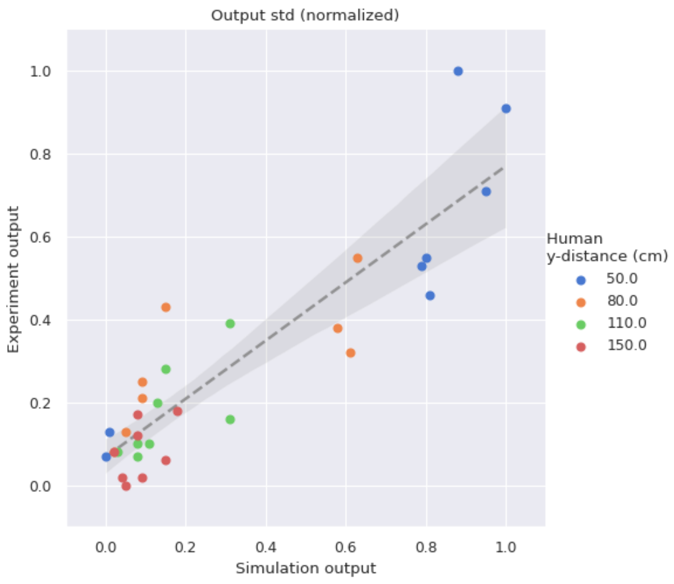
Figure 3.7 shows the normalized standard deviation of the simulated values compared to the actual values from the experiment. Ideally, all points should follow the same linear upwards trend. When a point A has a higher simulation output than point B, it should also have a higher experiment output than point B. The results show a mostly linear relationship with some outliers.

In Figure 3.7a, points that were generated using the same configuration, but at different target distances, are colored equally. Here, we see that the simulation has overestimated the performance of configurations that were rated 'good' ((50, -45) and (35, -25)). For these points, the output value of the simulation is higher than in the experiment. It can also be seen that the the configurations deemed 'medium' ((140,-5), (125,0)) are often underestimated. However, in the majority of the cases, the configurations that were rated better than other configurations by the simulation, had the same relation in the experiments. For example, each point for the configuration (35,-25) is rated better than each corresponding point of (50,-45), which is the case for the experimental results as well. The same holds for (110, -10) and (140,-5) for which the former always scores better than the latter. For the combination (35,-25) and (35,0) this holds for almost all points, except for the highest scoring one where the simulator is mistaken.

Figure 3.7b shows the same points, except they are colored by the target distance from which the output was generated. Here, we can see how the target distance influences the performance of the simulator. The further the points are from the line, the larger the error is. It is clear that large errors



(a) Colored by their configuration



(b) Colored by target distance

Figure 3.7: Validation of the simulated values by comparing the normalized metric values to the normalized metric values of experimental data with the same configurations.

occur in case the distance to the target is 50 cm. The error decreases with increasing target distance. At shorter target distance, the influence of the obstacle (target) is largest which apparently is a source of error in the simulation.

A number of things can be done to improve the result:

- **Improving reflectively coefficients** - the real world can be modeled more accurately by determining the accurate reflectivity coefficients for the scene. They were now based on estimations from literature and manual tuning.
- **Improved 'human' modeling** - In the simulation, the human is now represented as a cuboid with a single reflectivity coefficient. Because at close distance to a solar cell the influence of the reflections off the human is relatively high, an improved model could improve the results.
- **Improve lamp modelling** - The lamps have now been modelled as point sources, even though the actual lamp is a line source. More points can be added to give a more accurate representation of the real scene. This was not done now to keep the computation time of the ray tracing algorithm down.
- **Increase mesh resolution** - Dividing the simulation in a more fine mesh will naturally result in more accurate results - but is also more computationally intensive.

Chapter 4

Target detection

This chapter discusses changepoint detection methods for extracting a binary output from a solar cell. When a person passes in front of a solar cell this will cause a change in its output because the person affects the amount of incident light reaching the cell. This can be a temporary dip caused by the person's shadow, or a temporary peak caused by extra light reflected from the person. To convert the solar cell output to a binary signal denoting whether or not it is detecting a target's presence, we need to find the start and endpoint of this temporary change (see Figure 4.1). A straightforward approach to this, is to use thresholding where the solar cell is said to be detecting when its output falls below a certain threshold. However this method cannot be made robust due to the following reasons:

- The baseline changes. When ambient light is present in the environment, the no-target signal will be higher on a sunny day, than on a cloudy day.
- 'Dips' can also occur due to sudden changes in the environment not caused by a human target. E.g, a cloud moves in front of the sun, or a curtain is closed somewhere.
- The output of a solar cell is nonlinear [62] and therefore the relative change in output due to shadowing is dependant on the environmental light intensity at that time.

Therefore the goal of target detection is to process the signal in real-time such that it robustly detects human targets and is not affected by changes in the environment. The process can be divided into three steps:

- **Filtering**: The incoming data is noisy and needs to be filtered before further analysis.
- **Marking changepoint events**: The data is analyzed for changes in the mean/variance of the incoming data. The timesteps at which this happens, are marked as 'changepoints'.
- **Marking detections**: The changepoint data is analyzed, and used to determine periods when a human target is detected by the solar cell.

4.1 Filtering

A solar cell is not optimized for sensing. There is noise present in its photocurrent output that should be filtered. Additionally, indoor lighting flickering at AC frequency can be a source of noise in the signal. There are two approaches to filtering: analog filtering and digital filtering. For analog filtering,

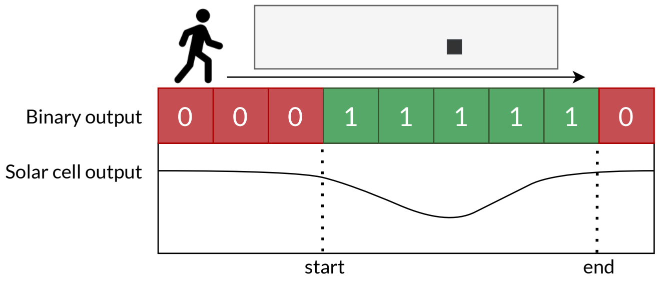


Figure 4.1: When a person passes a solar cell, the output of a solar cell changes. This signal is converted to a binary output that outputs '1' when a person is detected.

we use a simple low-pass RC circuit to filter the worst high-frequency noise and to stabilize the ADC input of the microcontroller that digitizes the signal.

We also use a digital filter so it is easy to change the filtering frequency as needed. A low-pass Butterworth filter has been implemented. Because humans walk relatively slow, a low cut-off frequency can be used. In [50] they find that human gesture frequency is less than 5 Hz. Lower cutoff frequencies can be used as well, but lead to a longer initial settling time. This is no problem in applications where the filter is running continuously, but can pose a problem when doing an analysis with data of a short time period. Figure 4.2 shows the result of the filtering applied to solar cell output data and shows that all high-frequency noise has been filtered out.

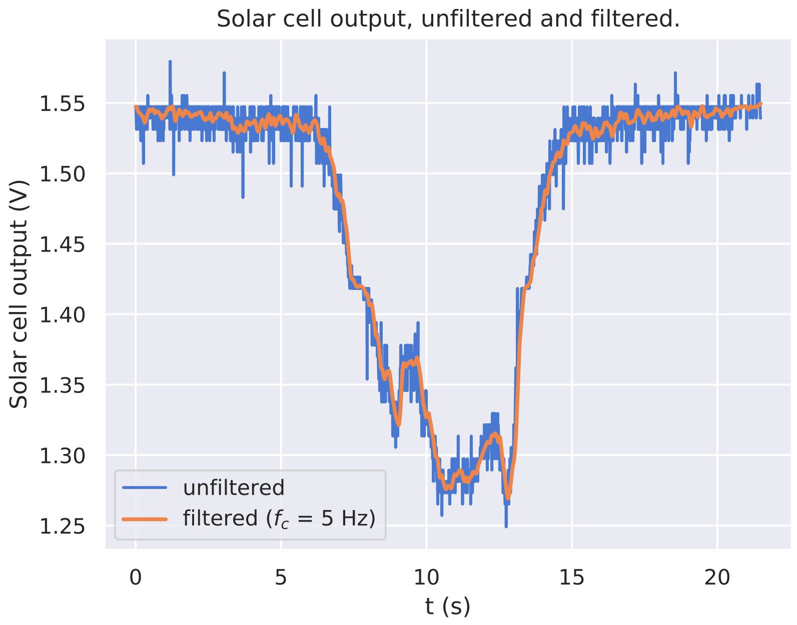


Figure 4.2: Result of filtering a solar cell output with a Butterworth filter with cutoff frequency $f_c = 5\,\text{Hz}$.

4.2 Changepoint Detection

When working with timeseries data, it can be interesting to know when the data changes in some way. For example in finance, it can be beneficial to detect when a stock value suddenly takes on a decreasing trend. In our application, we want to know when the solar cell output suddenly changes due to a person passing by. We need to do this without knowing anything about the magnitude or the duration of change beforehand.

A change in a timeseries is usually the result of a change in the generative parameters of the underlying model [4, 70]. These abrupt changes can be detected using changepoint detection (CPD) methods. These methods return *changepoints*: the timesteps at which the changes happen. Within CPD methods, two main categories exist: (a) offline and (b) online methods. Offline methods look at the data retrospectively and try to segment it based on a loss function using the least amount of segments. The transition between segments are marked changepoints. Online CPD methods learn about the data distribution over time, and detect when this distribution has suddenly changed. This learning property is useful in our application, as the data distribution is different for different solar cells and, as described in the introduction, can change over time.

For both offline and online changepoint detection, approaches using Bayesian statistics give good results [70]. The reason for this is that Bayesian methods allow the use of domain knowledge of the underlying system. Because our application is real-time, we focus on online changepoint detection. Two algorithms for Bayesian online changepoint detection were developed simultaneously by [1] and [23].

In this section, we describe the implementation of two changepoint detection methods for our application. First, we describe the Bayesian online changepoint detection algorithm based on the approach given by [1]. Next to that, we implement a simple self-developed algorithm dubbed 'gradient' CPD (or GCPD). This is based on thresholding of the gradient of the signal.

4.2.1 Bayesian online changepoint detection

This section focuses on the implementation of Bayesian Online Changepoint Detection (BOCD) for our application. BOCD keeps track of an estimated mean and variance of the signal's probability distribution, and updates these with every new data value received. This compensates for slow changes in the environment, which in our application can be caused by changes in daylight. When a fast change happens, such as a person passing a solar cell causing the output to change significantly in a very short period of time, BOCD will recognize that the data is from a new distribution. First, we describe the basic BOCD algorithm from [1]. Then, we describe how we implement the underlying model, changepoint extraction and initialization for our application.

Algorithm

The most important variable in the BOCD algorithm is the run length r. At any given timestep n, the run length r_n gives the time since the last changepoint. For example, if $r_n = a$ it means that the last changepoint occurred at timestep $n - a$. Figure 4.3 shows an example of how the value of r is updated based on the input data x_n. The second plot shows the possible transitions for values of r_n. For each new data point x_n, BOCD determines whether it is from a new distribution (changepoint) or not. When a changepoint is detected, it follows the dashed transitions and $r_n = 0$. When no changepoint is detected, it follows the solid lines and $r_n = r_{n-1} + 1$. The colored dots indicate the values for r for the the example signal x. There are changepoints at $t = 3$ and $t = 5$, so $r_3 = 0$ and $r_5 = 0$. At other timesteps it indicates the time since the last changepoint. $r_2 = 2$ means that at time $t = 2$, the last changepoint happened two timesteps ago.

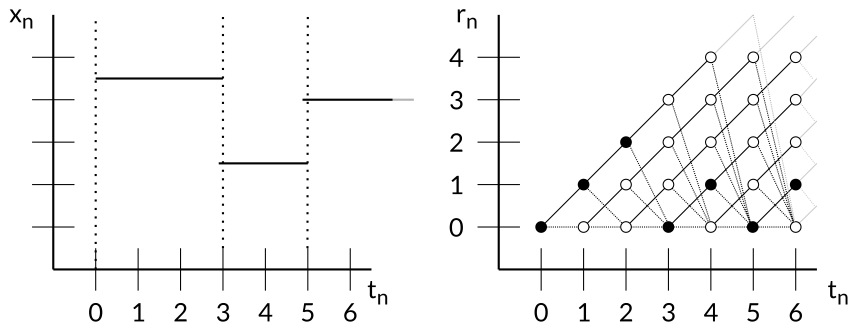


Figure 4.3: Evolution of the value for run length r_n for a signal x_n with changepoints at t=3 and t=5. For each timestep, the value of r_n increases by one (solid line), or is reset to $r_n = 0$ when a changepoint occurs (dashed line). The value of r_n that is most likely for the timestep is colored black.

The value of r shown in Figure 4.3 can be modeled as a matrix $R_{n,r}$. The matrix contains for each timestep $n \in \{0,1,\ldots,T_N\}$ the probability of the run length being length $r \in \{0,1,\ldots,T_N\}$. E.g., in the example we could have $R_{1,0} = P(r_1 = 0) = 0.1$ and $R_{1,1} = P(r_1 = 1) = 0.9$. This means that at timestep $t = 1$ it is most likely that a changepoint occurred 1 timestep ago. To calculate these probabilities, BOCD uses a model of the underlying data distribution. The parameters of this model are constantly updated to estimate the distribution of the current run. From this model we can calculate the predictive probability of a new data point x_n. This is described by Equation 4.1 and gives the probability of x_n belonging to the currently estimated model. The implementation of this model is described later in this section.

$$\pi_n^{(r)} = P(x_n | v_n^{(r)}, \chi_n^{(r)}), \quad \forall r \tag{4.1}$$

The predictive probability is used to calculate two different probabilities: the *growth probability* and the *changepoint probability*. The growth probability (Equation 4.2) gives the probability that the new data point x_n is from the same distribution as the previous timestep and that the run length r should be incremented by one. Here, the hazard function H gives the probability that a changepoint will occur at time n, if it did not occur at time $n-1$.

$$P(r_n = r_{n-1}+1, \mathbf{x}_{0:n}) = P(r_{n-1}, \mathbf{x}_{0:n-1}) \cdot \pi_n^{r_{n-1}} \cdot (1 - H(r_{n-1})), \quad \forall r_{n-1} \tag{4.2}$$

The changepoint probability is the probability that the new data point x_n is from a new distribution and the run length r gets reset to 0:

$$P(r_n = 0, \mathbf{x}_{0:n}) = \sum_{r_{n-1}} P(r_{n-1}, \mathbf{x}_{0:n-1}) \cdot \pi_n^{r_{n-1}} \cdot H(r_{n-1}) \forall r_{n-1} \tag{4.3}$$

The probability values are calculated for all values of r_{n-1}. Then, all calculated probabilities are normalized (Equation 4.4) to get the joint probability distribution. The new values are appended to the R matrix:

$$R_{n,r} = P(r_n = r | \mathbf{x}_{0:n}) = P(r_n, \mathbf{x}_{0:n}) / \sum_{r_n} P(r_n, \mathbf{x}_{0:n}), \quad \forall r \in r_n \tag{4.4}$$

After updating the R matrix, the final step is to update the hyperparameters of the estimated model using the new value x_n to get $v_{n+1}^{(r)}$ and $\chi_{n+1}^{(r)}$.

Estimating model mean and variance

BOCD uses an underlying model of the signal to calculate the predictive probability in Equation 4.1. For our application, we assume that the solar panel output can be modelled as a Gaussian. The hyperparameters of a Gaussian are the mean μ and variance σ^2. For each new received data point x_n, we need to update our estimate of the mean and variance. In Bayesian inference, hyperparameters can be estimated using conjugate priors. In [57] we find that for a Gaussian with an unknown mean and variance, the conjugate prior is the Normal-Gamma. For univariate data, the Normal-Gamma can be marginalized to a Student's t-distribution with center μ_n, precision $\Gamma = \frac{\alpha_n \kappa_n}{\beta_n(\kappa_n+1}$ and $2\alpha_n$ degrees of freedom [57]. This enables us to calculate the predictive probability for a new sample x_{n+1}:

$$P(x_{n+1}|\nu_n^{(r)}, \chi_n^{(r)}) = t_{2\alpha_n}\left(x_{n+1}|\mu_{0:n}, \frac{\beta_n(\kappa_n+1)}{\alpha_{0:n}\kappa_{0:n}}\right) \tag{4.5}$$

According to [57], for a new data point x_{n+1} the parameters of this model can be updated as follows:

$$\mu_{n+1} = \frac{\mu_n \cdot (\kappa_n + n) + x_{n+1}}{\kappa_0 + n + 1} \tag{4.6}$$

$$\alpha_{n+1} = \alpha_n + 0.5 \tag{4.7}$$

$$\kappa_{n+1} = \kappa_n + 1 \tag{4.8}$$

$$\beta_{n+1} = \beta_n + \frac{\kappa_{0:n}(x - \mu_{0:n})^2}{2(\kappa_{0:n}+1)} \tag{4.9}$$

From these values the estimated variance can be calculated using:

$$\sigma_n^2 = \frac{\beta_{0:n}(\kappa_{0:n}+1)}{\alpha_{0:n}\kappa_{0:n}} \tag{4.10}$$

It is important to note that the model parameters from Equations 4.6 - 4.9 are calculated and stored for each run length separately. As an example, the parameter for mean is stored as a vector $\mu_n^{(r)}$. $\mu_n^{(4)}$ stores the estimated mean in the case that the current distribution has been running for 4 timesteps. This value is thus only generated using the last four samples of x. The value $\mu_n^{(0)}$ contains the initial value. At each timestep n, each value in the vector is updated.

Changepoint extraction

The probability distribution $P(r_n|\mathbf{x}_n)$ from Equation 4.4 can be used to determine when a changepoint occurs. In an ideal situation, as in the original paper [1], we can say that a changepoint happens at time n, when $P(r_n = 0|\mathbf{x}_{0:n})$ exceeds a threshold $P_{r,min}$. However, we found that in reality this method did not give good results, missing many changepoints.

Figure 4.4 shows the real output of a solar cell with the estimated mean and variance, as well as a plot of the R matrix. In the ideal case, we would see sharply defined lines (as the first line) that restart at 0 when a changepoint happens. However, this is not the case. Firstly, the lines are not sharp. There is a clear difference between the line between 0 and 200, and the line between 1200 and 1400. Between 0 and 200, the algorithm is very certain about the value of r, as for each timestep only one value gets assigned a high probability. Between 1200-1400, multiple values of r get assigned a high probability, which makes the line appear more thicker.

The next problem is that when there is a changepoint, the value of r with the highest probability is not necessarily $r = 0$, but can also be $r = 10$. When the change of distribution is not fast enough, it can take a few samples before the algorithm 'picks up' on the new distribution because the values for predictive probability are relatively low. This can mean that it does not detect a new distribution until it has already been running for multiple timesteps.

The final problem is that the algorithm can change its mind about when a changepoint began. This can happen because it constantly updates the probabilities for each possible run length, including

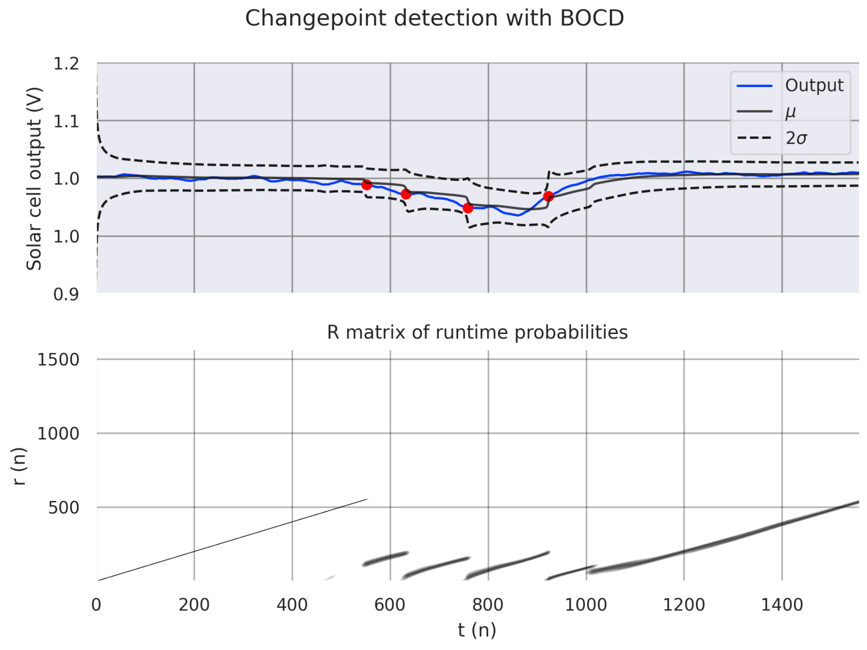


Figure 4.4: Bayesian online changepoint detection. The upper plot shows the estimated model parameters mean μ and standard deviation σ and the detected changepoints. The bottom plot shows the values of R at each timestep, where a darker color marks a higher probability for run length length r.

the model parameter values. In Figure 4.4 this can be seen around timestep 1000. Here, it updates its estimated run length to a lower value.

To tackle these errors, three additional conditions have been introduced for determining when a changepoint has occurred. At each time n, the run length $\hat{r}$ is the run length with the highest probability: $\hat{r}_n = \text{argmax}_r R_{n,r}$. This is considered a changepoint when the following conditions have been met:

1. The probability of the estimated run length is a sum of surrounding run lengths. Thus a changepoint is detected when $\sum_{r=\hat{r}-n_d}^{r=\hat{r}+n_d} R_{n,r} \geq P_{r,min}$. Where n_d is a number of samples called doubt samples. This is needed as the algorithm is bad at determining the 'exact' run length (which gives the thick lines).

2. The time difference between two consecutive changepoints should be larger than a threshold: $\Delta r = \hat{r}_n - \hat{r}_{n-1} \geq \Delta_{r_{\max}}$. Sometimes the algorithm needs some time to fully adjust to a new run length, leading to small jumps in a steadily increasing slope (around timestep 1000 in the example). Using this method, most of these small jumps are filtered out.

3. The start of the new run length may not lie less than $n_{start,max}$ seconds in the past. Thus for a changepoint at $\hat{r}_n \neq \hat{r}_{n-1} + 1$ it should hold that $\hat{r}_n \leq n_{start,max}$. This also filters out small jumps.

Initialization

Correct initialization plays an important role in the performance of the BOCD algorithm. When not initialized correctly, the estimated model parameters are not able to converge enough to calculate the predictive probabilities from Equation 4.1 well.

- The following values algorithm parameters are always initialized the same: $R_{0,0} = 1$, $\alpha_0 = 1, \kappa_0 = 1$.
- Mean μ_0 is initialized to x_0 when the first sample arrives (this lead to a large increase of performance compared to initializing the mean as 0).
- β_0 is in this case initialized as 1000, but is dependent on the application and the magnitude of the output. This value can be changed to change the sensitivity of the algorithm as it is related to the expected variance of the data. A lower value means higher sensitivity.
- The hazard function is set to 1^{e-4}. A larger hazard rate increases the influence of predictive probability of the most recent data point [77]. When this is the case, the algorithm will detect a changepoint when a new data point comes from a new distribution. With a lower hazard rate, the algorithm will need more samples to detect a changepoint.

Implementation and efficiency

The amount of values for r the algorithm tracks grows linearly in time (this is also the case for κ, α and β). This increases the required memory and computation time. To make it possible to run the algorithm continuously, some adaptations need to be made. However, this has to be done in a smart way, as deleting entries can delete information that will become important long-term. The following methods were used:

- Due to changepoints occurring, the probability for high values of r can become very small. Therefore, values of $r > \hat{r} + n_{tail}$ for which holds $\sum_r R_r > P_{prune}$ can be deleted, where P_{prune} is empirically determined.
- When no changepoint occurs and the size of R grows larger than a user-defined threshold based on memory limitations, R can be reset to its initial values. The model can be reset using its current values of mean μ and β as new initial values. Note that if this happens right before someone enters the solar cell's field of view, they will not be detected. Therefore it is advisable to not do this too often or at times when no one is expected.

4.2.2 Gradient changepoint detection

To compare the performance of BOCD, we implemented another simple ad-hoc method of changepoint detection dubbed 'Gradient CPD', or GCPD. The uses the basic idea that changepoints occur when there is a steep change in the signal, which causes a high gradient. When the gradient is high enough, we can say a changepoint has occurred. This method has the advantage that it does not assume a model (e.g., a Gaussian) for the signal beforehand, such as BOCD does.

After a new sample x_n arrives, the gradient is calculated for a time window W:

$$G_n = \sum_{k=n-W}^{k=n} \nabla_k \tag{4.11}$$

Here, W is the window size, and G_n is the sum of gradients at time n. ∇_k is calculated using a first-order finite distance approximation for the edges of the window, and a second-order finite distance approximation elsewhere:

$$\nabla_k = \begin{cases} x_{k+1} - x_k & \text{if k = n-W} \\ (x_{k+1} - x_{k-1})/2 & \text{if n - W < k < n} \\ x_k - x_{k-1} & \text{if k = n} \end{cases} \tag{4.12}$$

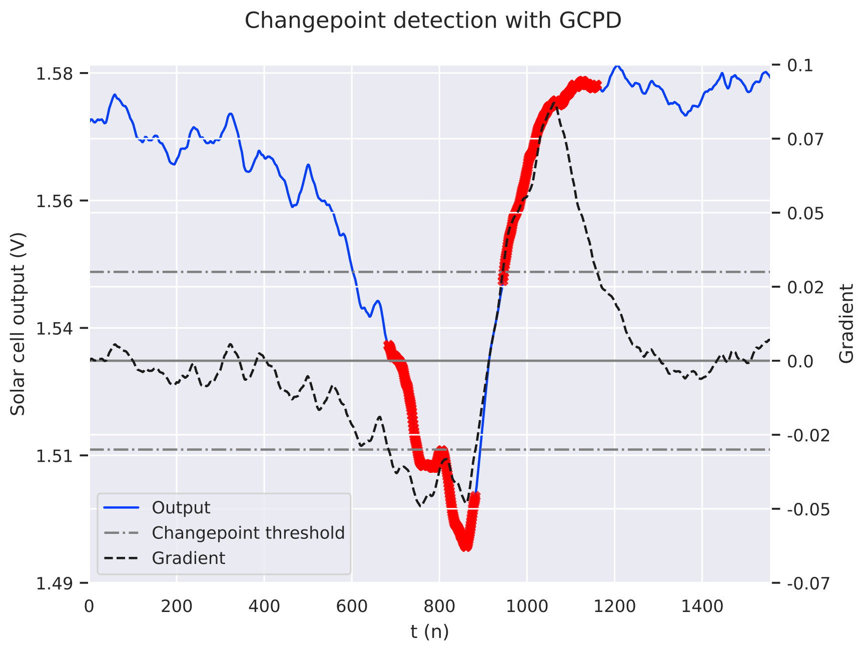


Figure 4.5: Changepoint detection with GCPD. Changepoints (red markers) are detected when the calculated gradient exceeds the threshold. The window size is 200.

After calculating the gradient $G_n x$ for a window, we conclude that there is a changepoint if it exceeds a threshold value τ:

$$c_n = \begin{cases} 1 & \text{if } G_n > \tau \\ 0 & \text{otherwise} \end{cases} \tag{4.13}$$

We use a window W because the changepoints for this application are relatively slow changes over time (someone passing by), as opposed to quick changes that can often be cause by noise. An appropriate window and threshold value are crucial for GCPD to work properly. A larger window leads to quick changes not being detected, a too small window leads to high frequency noise being detected as changepoint. The threshold that is used should be dependant on both the window size as the sampling frequency of the signal:

$$\tau = |W| \cdot \frac{1}{f_s \cdot S} \tag{4.14}$$

Where the sensitivity $0 < S \leq 1$ is an empirically determined value. For smaller S, the algorithm will become less sensitive.

4.3 Marking detections

As is clear from the large number of marked changepoints in Figures 4.4 and 4.5, a single target passing by can trigger multiple changepoint events. These changepoints need to be converted into 'detections'. A detection is the period of a target moving into and exiting the solar cell' detecting range. To be able to mark start- and ending times of a detection, each changepoint is associated with an extra value called the *detection signaller*. The value used for this purpose depends on the algorithm.

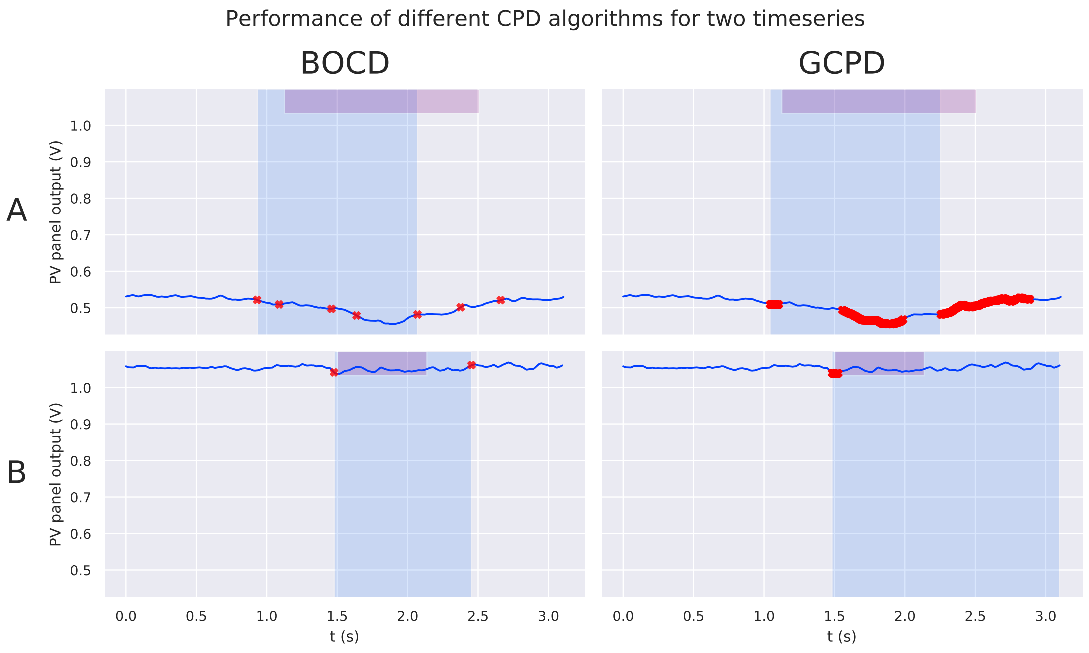


Figure 4.6: Target detection by different CPD algorithms. The red markers are changepoints. Timesteps marked as ‘detection’ are colored blue. The red band denotes when the target is in vicinity of the specific solar cell. In the bottom-right plot, the algorithm is unable to detect the end of the detection. Data is from the experiments described in Section 3.3.3. Configuration for A: angle = −45°, height = 50 cm, target distance = 50 cm. Configuration for B: angle = −5°, height = 140 cm, target distance = 110 cm.

For BOCD, the detection signaller is the change in estimated mean. Figure 4.4 shows that for the first changepoint the mean has a decreasing trend. We therefore store a negative value as detection signaller. The decreasing trend continues for the second and third changepoint, meaning they are part of the same detection. At the final changepoint, the mean is trending upwards. This is a positive value, which is different from the stored negative detection signaller. From this we conclude that the current detection has ended.

For GCPD the detection signaller is the sign of the gradient at the changepoint. In Figure 4.5, GCPD detects many changepoints. In the beginning of the dip that is caused by a person walking by, the gradient is negative for all changepoints. This is stored as detection signaller. At the end of the detection, all changepoints have a positive gradient, which signals the end of the detection. Note that in case a passing target does not cause shadowing but instead causes extra reflections, the start and end of the resulting peak is detected similarly. The difference is that the sequence of signals is reversed. A new detection will start with a positive gradient, and end with a negative gradient. This does not have to be adjusted manually, as the detection will simply end when the new signal value is opposite of the stored detection signaller.

Errors can occur when the end of a detection is not detected properly. When this happens, the changepoint detection will output that a solar cell is detecting something, even though the target has already exited the cell’s detecting range. This can be caused by a changepoint not being detected, or by detection signallers not finding the end of a detection due to noise. To avoid this problem, a timer can be implemented that automatically ends a detection after a certain time of inactivity. Another notable point is that this algorithm ends the detection at the first changepoint that signals an ‘end’. In the case of Figure 4.5, this means that the detection will end before timestep 1000, even though from visual inspection we can see that after this time the target is still within the solar cell's detecting range for a while.

Figure 4.6 shows examples of detection marking in practice. It can be seen that the detection periods are biased towards the time when the targets enters the FOV of the solar cell. Furthermore, in one case the GCPD algorithm is unable to detect the 'end' of a detection. This type of fault is in some cases unavoidable because the change in output is not defined well enough. This type op false positive therefore needs to be taken into consideration when the detection data is used further down the processing pipeline.

4.4 Evaluation

To compare the performance of the two algorithms, the experimental data from the simulation evaluation (Section 3.3) is used. For this data, the target position at each timestep is known, making it possible to compare the detection accuracy for both algorithms.

4.4.1 Marking true detections

To evaluate the accuracy of the detection algorithms, it needs to be determined at what timesteps the target is considered detectable by a solar cell. For this, first the distance of the target to the receiver is calculated for each timestep:

$$dx_t = |x_{\text{receiver}} - x_{\text{human},t}| \tag{4.15}$$

$$dy_t = |y_{\text{receiver}} - y_{\text{human},t}| \tag{4.16}$$

$$ds_t = \sqrt{dx_t^2 + dy_t^2} \tag{4.17}$$

A target is considered detectable when $dx_t < 40\,\text{cm}$ and $ds_t < 100\,\text{cm}$.

4.4.2 F1 score

To compare the performance of the algorithms, the F1-score is used as metric. The F1 score is a popular measure that incorporates precision and recall. Precision is the fraction of detected 'events' that are real events. Recall is the fraction of real events that are actually detected. These are combined in a F1-score. The F1-score is given Equation 4.18.

$$\text{F1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{4.18}$$

The F1-score is most notably used in classification problems, as calculating precision and recall are easily calculated with point-based data (an event is correctly classified, or not). However, events in time-series span a period of time and estimated events may only partially overlap a true event. To be able to calculate the F1-score for ranges of time series data, an adaptation is provided by [68] describing how precision and recall can be calculated.

The equations for determining recall and precision are described below and use the following inputs:

- R is the set of N_R non-overlapping time periods when the target is within detection range. These are marked as 'true detections' (Section 4.4.1).
- P is the set of N_P non-overlapping time periods when the changepoint detection algorithm signals that there is a detection (Section 4.3).

Recall Recall rewards an algorithm being able to detect a target when it passes by (true positives) and penalizes algorithms detecting a target when there is no target (false negatives). Recall can be calculated by iterating over all true detection ranges $R_i \in R$ to calculate the recall for each range, which is added up to a total recall score. The total score is then divided by the total number of true detections (N_R) to obtain an average recall score for the whole time series:

$$\text{Recall}(R,P) = \frac{\sum_{i=1}^{N_R} \text{Recall}(R_i,P)}{N_R} \tag{4.19}$$

To calculate recall for each R_i the following need to be considered according to [68]:

- Existence: Detecting a true detection is valuable, irrespective of the time range being accurate.
- Size: If a larger portion of R_i is covered by P, the recall score is higher.
- Position: The portion of R_i that is detected correctly might matter depending on the application.
- Cardinality: It is better if R_i is covered by a single prediction range $P_j \in P$ than with multiple different ranges in P in a fragmented matter.

For this application, the focus is on *existence* and *cardinality*: it is important that a target is detected, and that each target is detected only once. Equation 4.20 is used to calculate recall for a single true detection range R_i. To reward existence and cardinality equally, $\alpha = 0.5$ is used.

$$\text{Recall}(R_i,P) = \alpha \cdot \text{ExistenceReward}(R_i,P) + (1-\alpha)\cdot \text{CardinalityReward}(R_i,P) \tag{4.20}$$

The ExistenceReward is given when R_i is identified, e.g., $|R_i \cap P_j| \geq 1$ for any $P_j \in P$. This means that R_j is considered identified if at least one timestep of R_i and P overlap.

$$\text{ExistenceReward}(R_i,P) = \begin{cases} 1 & \text{if } \sum_{j=1}^{N_P} |R_i \cap P_j| \geq 1 \\ 0 & \text{otherwise} \end{cases} \tag{4.21}$$

The CardinalityReward can be calculated using Equation 4.22 where the overlap size function ω is defined in [68]. It gives a weight to the overlap between a true range R_i and an estimated range P_j, while incorporating a bias δ. In this case, middle bias is used, which gives a larger weight when P_j covers the center point of R_i. This places an importance on recognizing the center of the true detection.

$$\text{CardinalityReward}(R_i,P) = \text{CardinalityFactor}(R_i,P) \cdot \sum_{j=1}^{N_p} \omega(R_i, R_i \cap P_j, \delta) \tag{4.22}$$

The CardinalityFactor is determined as in Equation 4.23. Different from [68], it has an additional term for no overlap. For $\gamma()$ the equation $\gamma(x) = 1/x$ is used where x is the number of $P_i \in P$ that overlap R_i. This gives a lower reward to true detection ranges that are covered by multiple estimated detections.

$$\text{CardinalityFactor}(R_i,P) = \begin{cases} 0 & \text{if } R_i \text{ overlaps with no } P_j \in P \\ 1 & \text{if } R_i \text{ overlaps with one } P_j \in P \\ \gamma(R_i,P) & \text{otherwise} \end{cases} \tag{4.23}$$

Precision The key difference between precision and recall is that precision penalizes false positives instead of false negatives. The algorithm iterates over all estimated detection ranges $P_i \in P$ and computes the precision for each. N_P is the total number of estimated detection ranges.

$$\text{Precision}(R, P) = \frac{\sum_{i=1}^{N_P} \text{Precision}(R, P_i)}{N_P} \tag{4.24}$$

The precision per estimated target range is calculated using Equation 4.25. To calculate CardinalityReward, Equation 4.22 is used. For bias δ, flat bias is used because of how marking detections is implemented. An estimated detection is not centered around the true detection, so it is not important to detect the center.

$$\text{Precision}(R, P_i) = \alpha \cdot \text{ExistenceReward}(P_i, R) + (1-\alpha) \cdot \text{CardinalityReward}(P_i, R) \tag{4.25}$$

4.4.3 Results

Figure 4.7 shows a violin plot for the F1-scores calculated with the values from the simulation described in Section 3.3. This data has 32 'runs' (8 different panel configurations, 4 different target distances) for which the score can be calculated, with one detection per run. The average F1-score is fairly similar for both algorithms. For BOCD it is 0.56 (average precision: 0.47, average recall: 0.85). For GCPD it is 0.51 (average precision: 0.47, average recall: 0.68).

The average score of the two algorithms is close together, however BOCD is much more consistent in its performance than GCPD. The scores of BOCD are spread evenly around the average value. GCPD has a larger number of runs performing very well (F1-score larger than 0.75), but on the other hand also has a significant number of runs scoring very bad (near 0). This in contrast to BOCD, where the lowest value is 0.25. BOCD scores significantly better than recall, meaning that it is better at detecting the existence of changepoints.

The results indicate that BOCD is a more robust algorithm giving good results for most situations. When looking at the actual detection plots in Figure 4.6, we see that when the target is passing at a further distance and the 'dip' is relatively small, BOCD is able to detect it more often than GCPD. For configuration B in Figure 4.6, the target passing is almost imperceptible by the human eye. BOCD is able to pick up on the change, whereas GCPD is unable to properly do so. It is possible to make GCDP more sensitive to small changes as well by changing the algorithm parameters, however this will also lead to a larger number of misdetections.

Tuning GCPD to work better for a specific situation is easy, as there are only two parameters (window size and threshold value) that can be tuned. BOCD is more difficult to tune for a specific situation as there are many parameters such as threshold values, windows, pruning strategies and an adjustable hazard function. Many of these parameters are also not mutually exclusive, meaning that when one parameter is adjusted, this requires for another parameter to be adjusted as well. For example, when we define the threshold for detecting a changepoint $P_{r,min}$ relatively high, it will take longer to detect a changepoint. This means that the value that limits the passed starting time of a changepoint $n_{start,max}$, should not be too low. The parameter values for both GCPD and BOCD have to be tuned empirically using real data.

In conclusion, GCPD works better when tuned to a specific situation with no changes in environmental light and targets passing at approximately the same distance. BOCD is more robust and gives a better average performance when using the same set of parameter values in different situations.

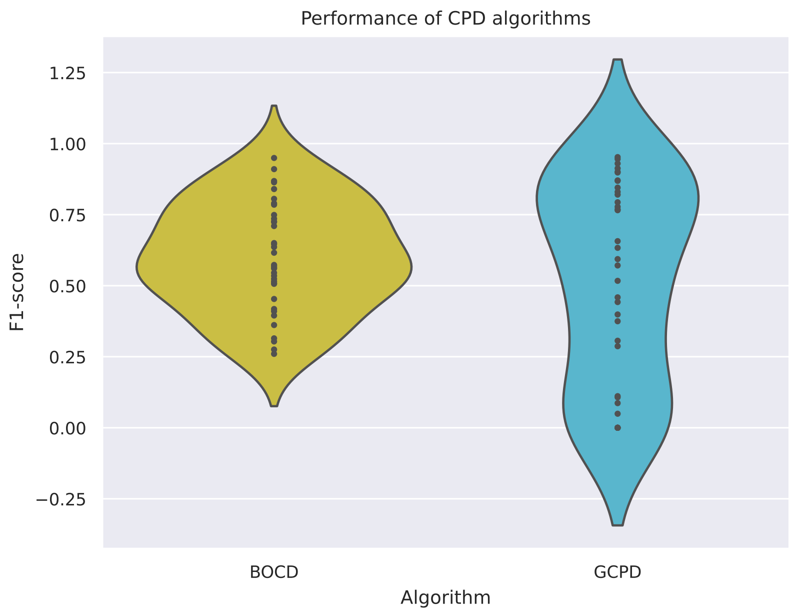


Figure 4.7: Violin plot of F1-scores of the BOCD and GCPD algorithms used for changepoint detection. Each point shows the score of one of the 32 experiment runs.

Chapter 5

Multi-target tracking

In the previous chapter, we used changepoint detection methods to produce a binary output from the solar cell output. This binary output indicates whether or not a solar cell is detecting a human target within its detection range. We want to use this data to be able to track an unknown number of human targets walking in an indoor area in which solar cells are placed. For this purpose, we design a multi-target tracking system.

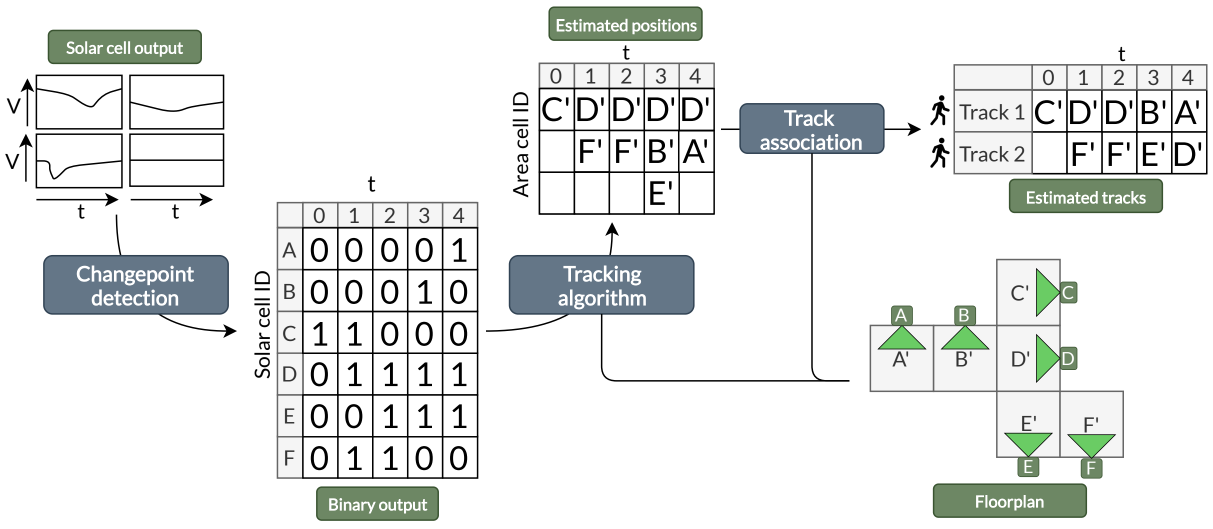


Figure 5.1: Overview of the steps required for multi-target tracking with example data. The tracking algorithm uses the binary data and the floorplan to estimate track positions. Track association uses this data to create the final sequential tracks.

Figure 5.1 shows an overview of the whole tracking system. The tracking algorithm takes two inputs. First, a floorplan that specifies the layout of the tracking environment, the positions of different solar cells and their expected detection ranges. The second input is the binary output of each solar cell. The floorplan is divided into smaller sections, or area 'cells'. How an area is divided, is dependent on the application. The tracking algorithm uses the inputs to estimate in which cells there are people present for each timestep.

The final step is track association, which uses the position estimates to estimate the number of

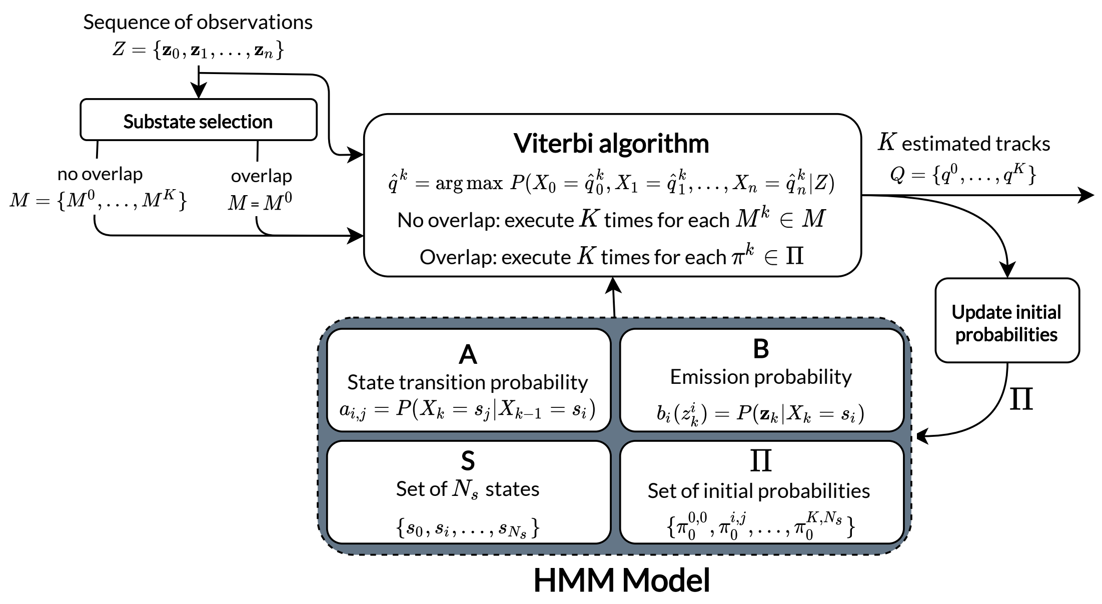

Sequence of observations
$Z = \{\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_n\}$
Substate selection
no overlap
$M = \{M^0, \ldots, M^K\}$
overlap
$M = M^0$
Viterbi algorithm
$\hat{q}^k = \arg\max P(X_0 = \hat{q}_0^k, X_1 = \hat{q}_1^k, \ldots, X_n = \hat{q}_n^k | Z)$
No overlap: execute K times for each $M^k \in M$
Overlap: execute K times for each $\pi^k \in \Pi$
K estimated tracks
$Q = \{q^0, \ldots, q^K\}$
Update initial probabilities
Π
A
State transition probability
$a_{i,j} = P(X_k = s_j | X_{k-1} = s_i)$
B
Emission probability
$b_i(z_k^i) = P(\mathbf{z}_k | X_k = s_i)$
S
Set of N_s states
$\{s_0, s_i, \ldots, s_{N_s}\}$
Π
Set of initial probabilities
$\{\pi_0^{0,0}, \pi_0^{i,j}, \ldots, \pi_0^{K,N_s}\}$
HMM Model

Figure 5.2: Overview of MT-HMM for multi-target tracking using hidden Markov models.

people present and create tracks. A track is a sequence of position estimates that belong to a specific target. Each time the tracking algorithm outputs new position estimates, they are combined with the correct track using track association. Because the data from the changepoint detection can be noisy, both the tracking algorithm and the track association method need to deal with false and missed target detections.

This chapter describes the implementation of two tracking algorithms and a method for data association. Two different types of tracking algorithms have been implemented. The first algorithm, multi-target HMM (MT-HMM), is based on a single-target tracking algorithm and enhanced to work for multi-target tracking. The second algorithm, the probability hypothesis density (PHD) filter, is specifically designed for multi-target tracking. In the final section of this chapter, we investigate how these two approaches influence the performance of the tracking system.

5.1 MT-HMM

Hidden Markov models (HMMs) have many applications including single-target tracking, for which it can lead to good results [5, 59, 69]. However, it is not straightforward to leverage HMMs for multi-target tracking. A method is presented in [12]. However, this approach relied heavily on a correct estimation of the number of targets, and required a second-order HMM, which is more difficult to model accurately. In [78] a method for multi-target tracking is described that uses HMM models. It considers all tracks simultaneously using combined probabilities instead of treating each track separately. A downside of this is that it leads to a computationally complexity that increases exponentially with the number of targets. In this section, we propose a computationally simpler approach dubbed 'MT-HMM'. This is loosely based on the concept of substate selection proposed by [12]. It has the advantage that it does not require the number of targets to be known beforehand and it does not rely on higher-order HMMs. Figure 5.2 shows an overview of this approach.

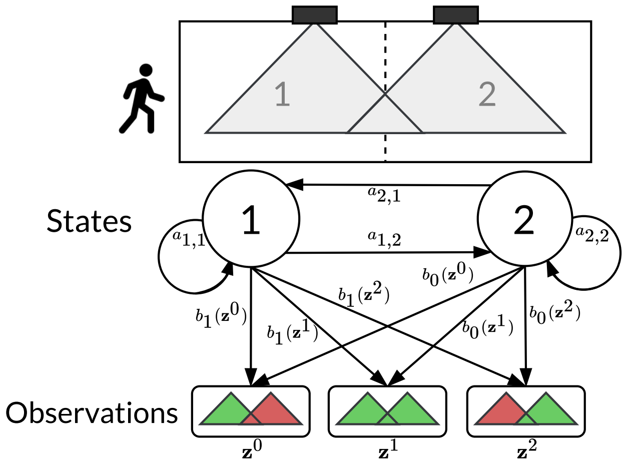


Figure 5.3: Simple example of an HMM. The area the person walks in are the states. The actual state the target is in is unknown. However, from the observations (the solar cell detection outputs), we can infer where the person is.

5.1.1 Hidden Markov model

The hidden Markov model (HMM) is based on the Markov chain. This is a set of states for which the transition probabilities between states are known. A process starts in the initial state, and moves through the chain according to the transition probabilities. In an HMM the actual state the system is hidden, meaning that it cannot be observed directly. However, as the process moves through a chain of states, it emits observable events with a probability based on the state the process is in. Figure 5.3 shows a simplified example with two states of the HMM applicable to our case. Given a process with state X_t at time t, the HMM model can formally be described as $\lambda = (A, B, S, \Pi_0)$ with:

- S - set of N_s states $\{s_0, s_1, \ldots, s_{N_s}\}$
- A - state transition probability $a_{i,j} = P(X_k = s_j | X_{k-1} = s_i)$
- B - state emission probability $b_i(\mathbf{z}_k) = P(\mathbf{z}_k | X_k = s_i)$ for a measurement $\mathbf{z}_k$ at timestep k
- Π_0 - set of initial probabilities $\{\pi_0^{0,0}, \pi_0^{i,j}, \ldots, \pi_0^{K,N}\}$ with $\pi_0^{j,i} = P(X_0^j = s_i)$ for target j (in case of multiple targets K)

5.1.2 Viterbi algorithm

The Viterbi algorithm [25] is a recursive-type algorithm that is used to find a maximum a posteriori (MAP) probability estimation of a state sequence. In other words, given a set of observations and possible states for a time period, it returns the most likely sequence of states that resulted in those observations. It uses the state transition probabilities and state emission probabilities defined in the hidden Markov model of the system. The recursive algorithm is given in Algorithm 2. The Viterbi algorithm is used to periodically determine target tracks given the received observations.

Algorithm 2: Viterbi algorithm

```
Input:
    The HMM model $\lambda = (A, B, S, \Pi_0)$
    A sequence of observations $Z = \{\mathbf{z}_0, \dots, \mathbf{z}_{T_N}\}$
Output:
    The most likely sequence of hidden states $\hat{q} = [\hat{x}_0, \dots, \hat{x}_{T_N}]$ and its Viterbi probability $p$
    Step 0 - Initialization
 1: $\delta_0(i) \leftarrow \pi^i \cdot b_i(\mathbf{z}_0) \quad \forall s_i \in S$        ▷ Stores the Viterbi probability
 2: $\psi_0(i) \leftarrow \emptyset$        ▷ Stores the most likely path
    Step 1 - Recursion
 3: for $\mathbf{z}_k$ in $Z$ do
 4:     for $s_i$ in $S$ do
 5:         $\delta_k(j) \leftarrow \max_{s_i \in S}\big(\delta_{k-1}(i) \cdot a_{i,j}\big) \cdot b_j(\mathbf{z}_k)$
 6:         $\psi_k(j) \leftarrow \arg\max_{s_i \in S}\big(\delta_{k-1}(i) \cdot a_{i,j}\big)$
    Step 2 - Backtracking
 7: $q_{T_N} \leftarrow \arg\max_{s_i \in S} \delta_T(i)$
 8: $p_{T_N} \leftarrow \max_{s_i \in S}\left(\delta_T(i) / \sum_{s_i \in S} \delta_T(i)\right)$
 9: for $k = T_N - 1, \dots, 0$ do
10:     $\hat{q}_k \leftarrow \psi_{k+1}(\hat{q}_{k+1})$
11: return $\hat{q}, p$
```

5.1.3 Multi-target implementation

Substate selection

The Viterbi algorithm has the limitation that it can only decode a single sequence of observations to a single track. For multi-target tracking, Viterbi needs to be applied to each target separately. The available data, however, is only one set of observations $Z = \{\mathbf{z}_0, \mathbf{z}_1, .., \mathbf{z}_{T_N}\}$ for a period T_N. At each timestep i an observation $\mathbf{z}_i$ contains zero, one or multiple possible target location states. From this data, we need to 'fork' the number of targets and determine which observation belongs to which target. We do this through a process called substate selection.

With substate selection, we try to extract non-overlapping tracks from the observations Z. No overlap means that Z can be split into K subsets Z^k that adhere to two conditions. (1) For each track it holds that each position estimate at $t = i$ has an estimate in $t = i + 1$ that is a maximum of one hop away. (2) At each time k the estimate belonging to the same track cannot be more than 1-hop away from eachother. Figure 5.4 illustrates this, where in one case two non-overlapping tracks are found. However, in the overlapping case, at $t4$ both tracks can select cell 5 without failing the conditions, which means that there is overlap. Depending on whether a sequence of observations Z contains overlapping tracks or not, the Viterbi algorithm is executed differently:

- **No overlap**: There will be K forked subsets of Z, corresponding to the expected number of targets. Then, we make K sets M^k that contain all unique position estimates for each Z^k. The 1-hop neighbour states of each state in M^k are added to each set as well. Note that no duplicates are allowed, so each state is only added once. The Viterbi algorithm is run K times, using a different set of states M^k for each run.

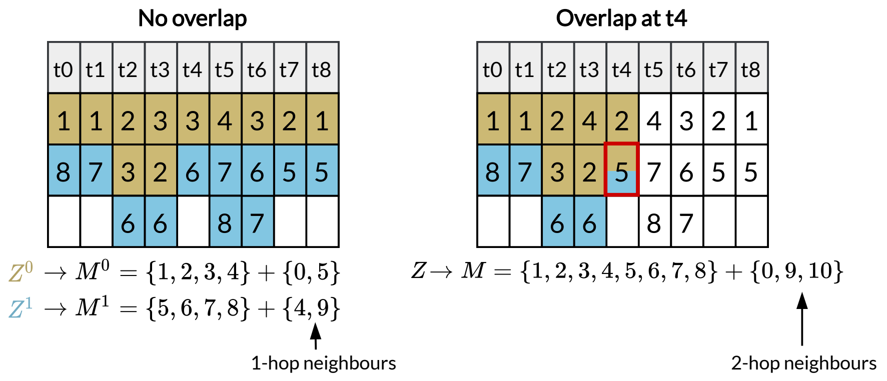

No overlap
Overlap at t4
t0 t1 t2 t3 t4 t5 t6 t7 t8
$Z^0 \rightarrow M^0 = \{1,2,3,4\} + \{0,5\}$
$Z^1 \rightarrow M^1 = \{5,6,7,8\} + \{4,9\}$
1-hop neighbours
$Z \rightarrow M = \{1,2,3,4,5,6,7,8\} + \{0,9,10\}$
2-hop neighbours

Figure 5.4: Example of substate selection with a sequence of states Z for no overlapping tracks and for overlapping tracks. In this example scenario there are 11 cells from 0 - 10 where a person is able to crossover to adjacent cells: so from 0 to 1, from 1 to 2, and so on. In the no overlap case, two potential tracks, Z^0 and Z^1 (yellow and blue) can be extracted where at each timestep the new cell is at most 1-hop away. For the overlap case, cell '5' at t4 can belong to both potential tracks (as it is one hop away from both cell 6 and 4), so Z is an overlapping track that cannot be split.

- **Overlap**: The set of states M is all states in Z and their 1- and 2-hop neighbours. The Viterbi algorithm is run K times where K is the size of Π_k, which is the set of initial probability values generated during the previous timestep.

When executing the Viterbi algorithm, we will use the reduced set of states $M \subseteq S$ instead of the full set of states S of the HMM model. As can be seen in Algorithm 2, this reduces the amount of iterations needed when iterating through the set of states, and therefore conveniently reduces the computation time.

Initial probability After the execution of the Viterbi algorithm (once or K times), the result is a set of estimated tracks $\hat{Q}_k = \{\hat{q}^0, \dots, \hat{q}^K\}$ and their corresponding Viterbi probability $\hat{P}_k = \{p^0, \dots, p^K\}$, with K the estimated number of targets. These values are used to determine the initial probability Π_{k+1} for the next iteration of MT-HMM. For each $\hat{q}^j \in Q$ and state $s_i \in S$, we use Equation 5.1 to calculate $\pi^{j,i}$: the probability of a target with track $\hat{q}^j$ finishing in state s_i. Here, $\hat{x}^j$ is the estimated state of the last timestep in $\hat{q}^j$ and $\mathscr{A}$ is the adjacency matrix. The probabilities are based on the calculated Viterbi probability p^j, which gives the likeliness of each track.

$$\pi^{j,i} = \begin{cases} p^j & i = \hat{x}^j \\ p^j/2, & \text{if } \mathscr{A}[\hat{x}, i] = 1 \\ 1/|S|, & \text{otherwise} \end{cases} \qquad \forall q^j \in Q, \forall s_i \in S \tag{5.1}$$

The values for initial probability are normalized before being used in the next iteration: $\pi^j = \pi^j / \sum_i \pi^{j,i}$. For each execution of the Viterbi algorithm using a set of states M^K (or just M), the values for initial probability that will be used is determined by (5.2).

$$\pi_0 = \arg\max_{\pi^j} \sum_{i \in M^n} \pi^{j,i} \tag{5.2}$$

Transition probability The transition probability is calculated based on the adjacency matrix $\mathscr{A}$ that includes self-loops of cells, so $\mathscr{A}_{i,i} = 1 \forall s_i \in S$. Then, the transition probability is calculated as

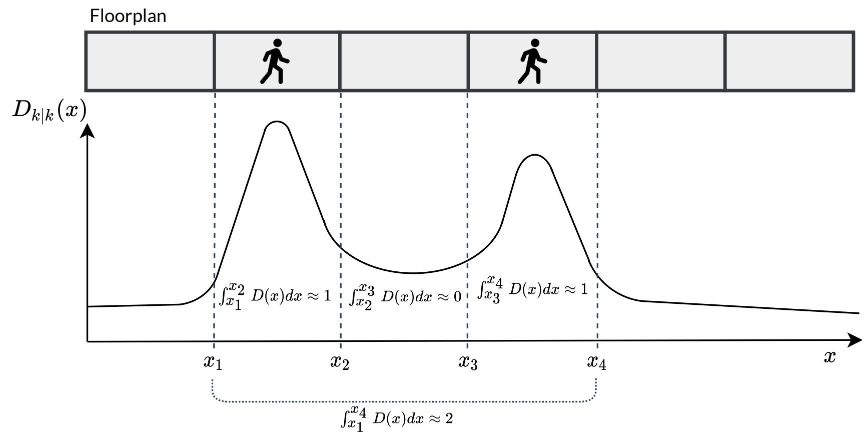

Floorplan
$D_{k|k}(x)$
$\int_{x_1}^{x_2} D(x)dx \approx 1$
$\int_{x_2}^{x_3} D(x)dx \approx 0$
$\int_{x_3}^{x_4} D(x)dx \approx 1$
x_1
x_2
x_3
x_4
x
$\int_{x_1}^{x_4} D(x)dx \approx 2$

Figure 5.5: A representation of the probability hypothesis density (PHD) for an area in which two targets are detected. $D_{k|k}(x)$ gives the density of the expected number of targets at a position x. Integrating the PHD over an area gives the expected number of targets.

$a_{i,j} = \frac{\mathscr{A}_{i,j}}{\sum_{s_k \in S} \mathscr{A}_{i,k}}$. The adjacency matrix of an environment is constructed from domain knowledge about the order of states through which a person can walk unobstructed.

Emission probability The emission probability $b_i(\mathbf{z}_k) = P(\mathbf{z}_k|X_k = s_i)$ is the probability of a set of measurements $\mathbf{z}$ occurring when a target is in state s_i. The measurement set $\mathbf{z}_k = \{z_k^j | z_k = 1\}$ contains a list of solar cells detecting a target (having a binary output of '1') at timestep k. With this measurement, the emission probability can be calculated by (5.3).

$$P(\mathbf{z}_k|X_k = s_i) = \sum_j \Big(P(z^j = 1|X_k = s_i)\Big) \tag{5.3}$$

Here, $P(z^j = 1|X_k = s_i)$ is the portion of cell i covered by the range of solar cell j, or $Area(R_j \cap A_i)/A_i$. Where R_j is the range area of solar cell j determined from simulation or measurements, and A_i is the area of area cell i.

5.2 PHD

The probability hypothesis density (PHD) is a method proposed by [52] and provides an approximation of applying a recursive Bayes filter to a multi-target system. The PHD is denoted as $D_{k|k}(\mathbf{x}|Z)$ for a state $\mathbf{x}$ given a set of measurements Z. It describes the density of the expected number of objects at state $\mathbf{x}$. This has the useful property that integrating the PHD for a random area S, gives the expected number of targets in the area. E.g., $\int_S D_{k|k}(\mathbf{x}|Z^{(k)})d\mathbf{x} = \hat{K}$. Peaks in the probability density indicate the positions of the target. Figure 5.5 illustrates the PHD.

5.2.1 Bayesian filtering

The values of the PHD can be updated using Bayesian filtering. Bayesian filtering has two steps: the prediction step and the correction step. During the prediction step, the new state of the system is predicted using a model of the system dynamics. In our case, we can model the expected transitions between area cells. During the correction step, the predicted position estimates are correct using sensor measurements. The measurements we have are the binary outputs of the solar cells. We can correct our predictions based on how probable the solar cell binary outputs are when the targets are at the positions we predicted them to be at.

$$D_{k|k} \xrightarrow{\text{Prediction}} D_{k+1|k} \xrightarrow{\text{Correction}} D_{k+1|k+1} \tag{5.4}$$

The Bayes filter equations for the PHD were derived by [52]. The prediction step is given as:

$$D_{k+1|k}(\mathbf{x}) = b_{k+1|k}(\mathbf{x}) + \int (p_s(\mathbf{w}) f_{k+1|k}(\mathbf{x}|\mathbf{w}) + b_{k+1|k}(\mathbf{x}|\mathbf{w}) d\mathbf{w} \tag{5.5}$$

Where $b_{k+1|k}(\mathbf{x})$ is the birth model describing the probability of birth of new targets, $p_s(\mathbf{w})$ is the survival probability of existing targets and $f_{k+1|k}$ models the state transition probability between area cells. For a new measurement Z_{k+1}, the update step is given as:

$$D_{k+1|k+1}(\mathbf{x}) \cong F_{k+1}(Z_{k+1}|\mathbf{x}) D_{k+1|k}(\mathbf{x}) \tag{5.6}$$

with

$$F_{k+1}(Z|\mathbf{x}) = \sum_{\mathbf{z}\in Z} \frac{p_D(\mathbf{x}) L_{\mathbf{z}}(\mathbf{x})}{\lambda c(\mathbf{z}) + D_{k+1|k}[p_D L_{\mathbf{z}}]} + 1 - p_D(\mathbf{x}) \tag{5.7}$$

Here, $D_{k+1|k}[h] = \int h(\mathbf{x}) D_{k+1|k}(\mathbf{x}) d\mathbf{w}$. The function $L_{\mathbf{z}}(\mathbf{x}) = f_{k+1}(\mathbf{z}|\mathbf{x})$ is the sensor likelihood function. The function $p_D(\mathbf{x})$ gives the probability of detection. $\lambda c(\mathbf{z})$ gives the distribution of false positives.

Birth probability For birth probability b, we make the assumption that targets can only appear in cell areas that are covered by the solar cells that are detecting (binary output '1') at a time instant. The birth probability distribution is a fixed value for birth probability divided uniformly over this area.

Surival probability The survival probability $p_s(x)$ denotes the probability of a target 'surviving' into the next timestep when it is at state x. We model the probability of a target surviving into state j from state i as $p_s \cdot a_{i,j}$. Here, p_s is a fixed surviving probability value and $a_{i,j}$ is the state transition probability as described in Section 5.1.3.

Detection probability The probability of detection $p_D(\mathbf{x})$ gives the probability of a target at state $\mathbf{x}$ being detected and thus being visible in the sensor data. For this, we assume a constant value based on the performance of the solar cells in the area.

Transition probability The state transition probability $f_{k+1|k}$ is equal to the state transition probability described in Section 5.1.3.

Sensor likelihood The sensor likelihood $f_{k+1}(\mathbf{z}|\mathbf{x})$ gives the probability of sensor measurement $\mathbf{z}$ when there is a target in state x. This is calculated as $Area(\mathbf{z} \cap A_i)/A_i$. Where $Area(\mathbf{z} \cap A_i)$ is the intersection of the cell area and the area covered by the estimated range of the solar cells that are detecting a target.

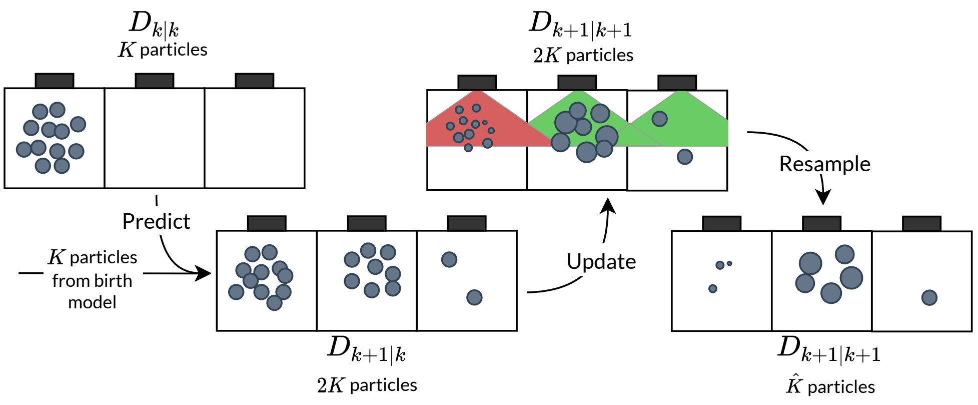


Figure 5.6: Overview of the particle filter implementation of the probability hypothesis density (PHD). The particles represent the probability density. When there is a cluster of highly-weighted particles in a certain position, it is likely that there is a target.

5.2.2 Particle filter implementation

Particle filters provide a way of tracking the state of a system, even when the system is non-linear. A particle filter represents a probability density with random samples. The PHD filter can efficiently be implemented using a particle filter. In this case, the weighted particles and their states (positions) represent the PHD. This approach was first proposed by [72], after which many adaptations with different variations have emerged [9,30,37,51,65].

Figure 5.6 shows an overview of the steps in the Bayesian filter implemented as a particle filter. Algorithm 3 describes the particle filter implementation in detail. We have adapted the approach used by [65] to conform better with the original PHD Bayesian filter equations from [52]. When the first measurement arrives, all K particles are spawned in random locations in the area covered by the detections. For further iterations, the update step $D_{k+1|k}$ is performed by sampling from the transition probability described in Section 5.1.3. To be able to detect new targets as well, K additional particles are spawned using the birth model.

During the correction step, $D_{k+1|k+1}$ is calculated by updating the weights of all particles base on sensor measurements. Particles that are in a position that correspond better with the measurements, get a higher weight. After updating the weights, the number of targets $\hat{K}$ in the system can be estimated by summing the weights of all particles - essentially integrating the PHD. The estimated locations of the targets are extracted by finding the maxima in the resulting density function. In the particle filter implementation, these maxima are represented as clusters of high-weighted particles. The positions of the the $\hat{K}$ targets can be inferred from the location of the clusters. For geometric data such as (x,y) position coordinates, clustering algorithms such as K-means clustering and Gaussian mixture model clustering can be used. For our application however, we have divided the area into discrete cells. Therefore, we use K-modes clustering for clustering with categorical data, which uses the Hamming distance to calculate the distance between different particles. Using this algorithm, $\hat{K}$ clusters are found and the cluster centroids are the estimated target positions. It should be noted that due to the nature of the PHD, two or more targets that are together in the same cell cannot be distinguished and will be counted as one target. This should be accounted for in the post-processing algorithm in case crossovers between target tracks occur. In the final step, the set of particles is resampled by randomly selecting $\hat{K}$ particles, with particles with a larger weight having a higher chance of being selected.

Algorithm 3: PHD particle filter implementation

```
    Step 0 - initialization
1:  for $i$ in $0 \dots K$ do
2:      $\mathbf{x}_0^i \sim p(\mathbf{x}|\mathbf{Z}_0)$
3:      $\mathbf{w}_0^i = 1/K$
    Step 1 - Predict new particles by sampling from the motion model
4:  for $i$ in $0 \dots K$ do
5:      $\hat{\mathbf{x}}_{k+1|k}^i \sim p(\mathbf{x}_{k+1}|\mathbf{x}_{k+1|k})$
6:      $\hat{\mathbf{w}}_{k+1}^i = p_s(\mathbf{x}_k^i) \cdot p(\mathbf{x}_{k+1}^i|\mathbf{x}_{k+1|k}^i) \cdot \mathbf{w}_k^i$
    Step 2 - Generate new particles from the birth model
7:  for $i$ in $K+1 \dots 2K$ do
8:      $\hat{\mathbf{x}}_{k+1}^i \sim p(\mathbf{x}_{k+1}|\mathbf{Z}_{k+1})$
9:      $\hat{\mathbf{w}}_{k+1}^i = \frac{p_B(\mathbf{x}_{k+1}^i)}{N}$
    Step 3 - Update weights using the measurement observation.
10: for $i$ in $1 \dots 2K$ do
11:     $\mathbf{w}_{k+1}^i = ((1-p_D) + \sum_{z_i \in Z^{k+1}} \left( \frac{p_D \cdot p(z_i|\mathbf{x}_k^i)}{\sum_{i \in 0..2N} p_D \cdot p(z_i|\mathbf{x}_{k+1}^i) \cdot \hat{\mathbf{w}}_k^i} \right) \cdot \hat{\mathbf{w}}_{k+1}^i$
12: Step 4 - Estimate the number of targets
13: $\hat{K}_{k+1} \leftarrow \sum_i \mathbf{w}_{k+1}^i$
    Step 5 - Resample
14: Normalize weights: $\hat{\mathbf{w}}_{k+1} = \hat{\mathbf{w}}_{k+1} / \sum_i \hat{\mathbf{w}}_{k+1}^i$
15: Randomly sample $\hat{K}_{k+1}$ particles with replacement to get $\{\mathbf{x}_{k+1}^i\}_{i=0}^{\hat{K}}$ and $\{\mathbf{w}_{k+1}^i\}_{i=0}^{\hat{K}}$
```

5.3 Track association and cleaning

A tracking algorithm returns one or more estimated positions for the current target(s) at each timestep. The stream of estimated position data needs to be converted into tracks. For example, when at the previous timestep there are one or more pre-existing tracks and two new position estimates arrive, these positions have to be associated with the correct track, or start a new track. Determining which estimate belongs to which track is called the data association problem and is a core challenge in multi-target tracking. When track association is done well, this leads to a good performance of the algorithm in terms of track continuity. Additionally, applying cleaning methods to these tracks can improve track continuity as well as accuracy.

5.3.1 Data association

There are numerous data association methods [9, 10, 60]. Most approaches calculate the distance of a new estimate to existing tracks, combining the estimate with the closest track. Distance metrics that can be used include the Hausdorff distance and Euclidean distance. Likeliness can also be used [12] [6], where the probability of an estimate belonging to an existing track is calculated by calculating the likeliness of a target moving to that new position given the most recent positions in an existing track. In particle filters, labels can be assigned to particles [9] based on the track they were associated with in the past. This label can be used to determine to which track a particle's descendants should be associated with.

There are also cases where a new position value cannot be estimated to an existing track. A new track is started when a new estimate has a distance or likeliness to each existing target that exceeds a set threshold value. A track is considered finished when it does not have a new estimate associated to it for a certain time period.

Two association methods have been implemented and tested, one based on likeliness and one based on distance. For both methods, the final assignment of new estimates to tracks is made by modelling it as a linear sum assignment that is solved using the Hungarian algorithm. Overall, the distance association approach, although relatively simple, proved the most reliable.

A2 association For a new position estimate arriving, the likeliness of it belonging to each existing track can be determined using state transition probabilities. This can use the state transition probability A with $a_{i,j} = P(x_k = s_j | x_{k-1} = s_i)$ as well as the 2-hop state transition probability A^2 with $a^2_{i,j,l} = P(x_k = s_i | x_{k-1} = s_j, x_{k-2} = s_l)$. In this implementation, when a new estimate $\hat{x}_t$ arrives, the likeliness of belonging to each existing track i is determined as $a^2_{\hat{x}_t, x^i_{t-1}, x^i_{t-2}}$. The new estimate is associated to the track with the highest probability. In case the existing track is too short and there is no value for x^i_{t-2}, the regular state transition probability A is used.

Distance association For a new position estimate arriving, the distance between the new estimated position and the existing tracks can be calculated. Because the tracking algorithms return the values of discrete cells instead of position coordinates, common distance measures such as the Euclidean distance cannot be used. Instead, the distance between two cells is the minimum amount of hops needed to go from one to the other. This can easily be calculated from the adjacency matrix $\mathscr{A}$ using the Floyd-Warshall algorithm, which generates a distance matrix D width $d_{i,j}$ giving the number of hops required to go from state s_i to state s_j.

5.3.2 Cleaning

The estimated tracks are periodically cleaned to remove noisy data within the tracks, as well as to remove spurious tracks created by noise.

Removing invalid tracks Tracks that fully overlap (part of) another track will be deleted. Both algorithms are not able to distinguish two targets walking in the same cells, and therefore these spurious tracks are deemed invalid.

Removing close tracks After association it can happen that two tracks exists that are very similar and are actually the same track. Therefore, tracks are deleted when they are within a threshold distance from another track for the past n timesteps. If two overly similar tracks are found, the one with the most recent position estimate is kept as the 'true' track. However, if for some timesteps the position estimate is unknown for this track, values from the similar track are merged into these timesteps.

Smoothing tracks Especially during transitions of a target between area cells, it can happen that there is noise in the estimated position where the estimated track frequently switches between the new and the previous cell. These moments are smoothed by filtering out short sequences of cell estimates, and replacing these with the preceding values.

5.4 Evaluation

This section describes the evaluation of the two tracking algorithms.

5.4.1 Metric

To quantify the performance of a tracking algorithm, we use a metric that incorporates the ability of the tracker to accurately estimate the number of targets, as well as the accuracy of the position estimates of correctly detected targets. These properties are accounted for in the CLEAR MOT metrics as proposed in [8]. This metric consists out of two values:

- Multiple object track accuracy (MOTA). Gives an indication of the ability of correctly detecting the number of targets and track continuity, regardless of the position estimation accuracy of the detection.
- Multiple object tracking precision (MOTP). Gives an indication of the estimation precision, regardless of targets being detected or not.

This metric requires for each timestep t a set of hypotheses $H_t = \{h_0, ..., h_m\}$ (position estimates) for a set of targets $O_t = \{o_0, ..., o_n\}$ (true known position of each target). At each time t, the best possible assignment of each hypothesis h_j with a target o_i is found. This is done using the Hungarian algorithm, with the cost function being the distance between the actual position o_i and estimated position h_j. In this case, $D[o_i, h_j]$ is used where D is the distance matrix between cells. Targets are only assigned to hypothesis when the distance falls within a threshold value $D[o_i, h_j] < \delta_d$. After the assignment step, the following assignment errors are determined:

- Targets with no hypothesis are counted as misses m_t.
- Hypotheses that are not matched with an target are counted as false positives fp_t.
- Switches where a target gets to a different hypothesis than before are counted as mismatch errors mme_t (in case of two targets passing, their hypotheses could be swapped due to occlusion).

Finally, MOTA and MOTP are calculated using the following equations:

$$\text{MOTA} = 1 - \frac{\sum_t (m_t + fp_t - mme_t)}{\sum_t g_t} \tag{5.8}$$

$$\text{MOTP} = \frac{\sum_{i,t} d_t^i}{\sum_t c_t} \tag{5.9}$$

With d_t^i the distance between o_i and the assigned hypothesis, c_t the number of valid matches found for time t and g_t the number of objects present at time t. Note that in some cases we do not care about track continuity. When this is the case, we set $mme_t = 0$ in the MOTA equation. Henceforth this will be denoted as MOTA*

5.4.2 Comparison of tracking algorithms

To compare the two tracking algorithms, we use data from a simulation in which the solar cells detect human targets perfectly when the target is within their detection range. The layout of the scene is shown in Figure 5.7. For evaluation with just one target, only Human 1 is simulated. For evaluating multi-target performance, two targets walking in opposite directions are simulated. Human 2 starts entering the scene when Human 1 is in cell 1. There are five area cells divided evenly between the solar cells.

To also evaluate the algorithm performance in case of misdetections (this occurs when the change-point detection algorithm fails), the data was modified to simulate a situation where a false negative occurs, as well as a situation where a false positive occurs. The following modifications were made:

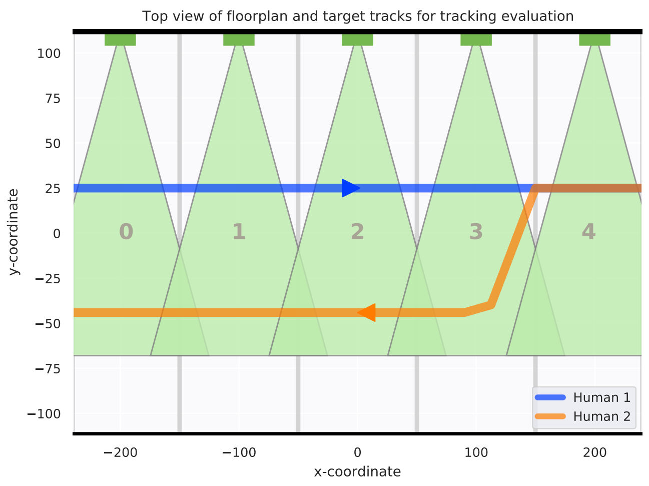


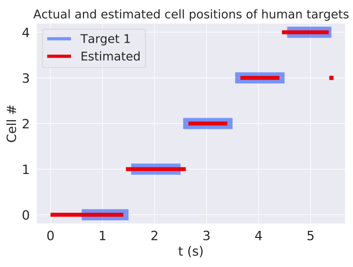
Figure 5.7: Top-view of floorplan used for evaluation of tracking algorithms. Walls are present at the top- and bottom side. The top wall is lined with solar cells, and areas are divided over the solar cells evenly. Human 2 'departs' when Human 1 is in cell 1, meaning that the two targets crossover in cell 3. In evaluations with one target, Human 2 is left out.

1. False negative (FN) - for both configurations the solar cell located in cell 1 is set to not detect anything at any time.
2. False positive (FP) - (1 target) the solar cell at cell 1 detects something for 1 s when the target is moving from 3 to 4 ($2 \leq t \leq 3$). (2 targets) the solar cell at cell 1 detects something for 1 s when both targets are moving into cell 3 ($2.75 \leq t \leq 3.75$).

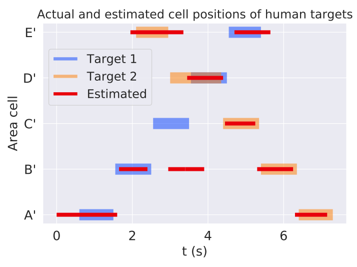
Table 5.1 gives the results for the comparison of the two algorithms. Figure 5.8 shows a few examples of actual tracks plotted against estimated tracks. For the test cases with one target, the algorithms perform similar. There is one notable exception, however, which is the case with 1 target and a false negative, which is also shown in Figure 5.8a and 5.8b. Here, MT-HMM is able to reconstruct the correct states better by backtracking through the internal state model, whereas for the PHD this is harder to compensate for. The reason is that for the PHD the weights are calculated using measurements. When there is no measurement for the particular cell (due to a false negative), particles in this cell will likely get low weights, moving the weight center to the adjacent cells.

Table 5.1: Comparison of two tracking algorithms.

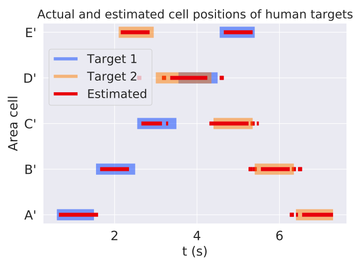
	MT-HMM		PHD	
	MOTA*	MOTP	MOTA*	MOTP
Configuration				
1 target	0.96	0.27	0.98	0.29
1 target + FN	0.96	0.27	0.79	0.46
1 target + FP	0.96	0.29	0.98	0.34
2 targets	0.61	0.27	0.67	0.26
2 targets + FN	0.52	0.42	0.57	0.41
2 targets + FP	0.60	0.41	0.6	0.27



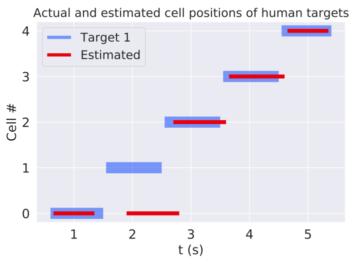
(a) 1 target + FN (MT-HMM)



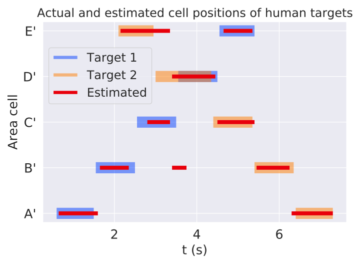
(c) 2 targets + FP (MT-HMM)



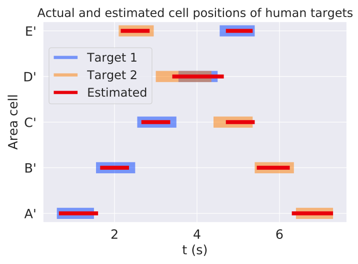
(e) 2 targets, no cleaning (PHD)



(b) 1 target + FN (PHD)



(d) 2 targets + FP (PHD)



(f) 2 targets, with cleaning (PHD)

Figure 5.8: Position estimation results for three different datasets.

For the test cases with two targets, the results show that the PHD algorithm is the better option for multi-target tracking. Especially because the tracking area is small, the probability that two target tracks overlap within the time period is relatively high. This makes it increasingly harder for MT-HMM to execute the substate selection part of the algorithm and extract multiple tracks. for this purpose, the PHD is better as it incorporates multiple targets in the underlying model. An example for two targets with a false positive is shown in Figure 5.8c and 5.8d. The false positives influences the MT-HMM algorithm in such a away that one cell goes undetected. The PHD is also influenced by the false positive and naturally starts detecting a target at the cell that emits the false positive. However, this does not influence its ability to recognize all correct cells as well.

The effect of cleaning is shown in Figure 5.8e and 5.8f, where the cleaning algorithm is able to smooth the data which increase position accuracy, but also gives a more accurate number for the number of targets. E.g., between $t = 5$ and $t = 6$, the algorithm detects a target at cell 2 and cell 1 simultaneously (which is incorrect, as it is the same target). The cleaning algorithms applied correctly smooth these out to be one consecutive non-overlapping track.

The evaluation was done using a relatively simple environment in which each area cell is covered equally by a solar cell, and the areas are connected sequentially. For now, we make the assumption that an algorithm performing better in a basic environment, will also perform better in a more complicated environment. From the evaluation it can be concluded that the PHD algorithm is better suited for multi-target tracking algorithms than MT-HMM. It should be noted that due to the large number of particles required for the PHD it is significantly more computationally expensive than MT-HMM. Furthermore, applying proper track association and cleaning methods is essential for improving the results of tracking algorithms as they correct and smoothen the final estimated output tracks.

Chapter 6

Experimental evaluation

To evaluate the system in a real-world setting, we conducted an experiment in a residential indoor environment. Up to two targets were tracked while walking through rooms with solar cell sensing devices placed at different locations. This chapter explains how the experiment was carried out and the results.

6.1 Hardware

The system was implemented on a developmental platform in order to carry out the experiments. This section describes the hardware and infrastructure setup.

6.1.1 Overview

The tracking system consists of a set of deployed sensing devices and a base station that the sensing devices communicate with. Figure 6.1 shows an overview of the system.

The sensing device contains the following elements:

- A microcontroller with wireless communication capabilities.
- A power source or power storage component.
- A solar cell.

The design used for evaluation was build around an ESP32 microcontroller development board as shown in Figure 6.2. A small 39 mm x 35 mm solar cell with a nominal output of 4 V, 35 mA was connected to the 10-bit ADC of the microcontroller. The circuit connecting the solar cell consists of a simple voltage divider to not exceed the input limits of the ADC, and a 100 nF decoupling capacitor. The device was powered through micro-usb for ease of use during testing. The devices communicated the ADC data to the PC, which acted as the base station. Data was sent over Bluetooth at a rate of 100 Hz, which is more than strictly necessary but allowed for extra analysis. The PC ran the changepoint detection algorithms, as well as the tracking algorithms. The rest of this section describes considerations for designing the actual system.

6.1.2 Communication

The sensing devices need to be able to communicate to the base station. The selected communication protocol should be able to cover the required range to the base station, be able to connect to

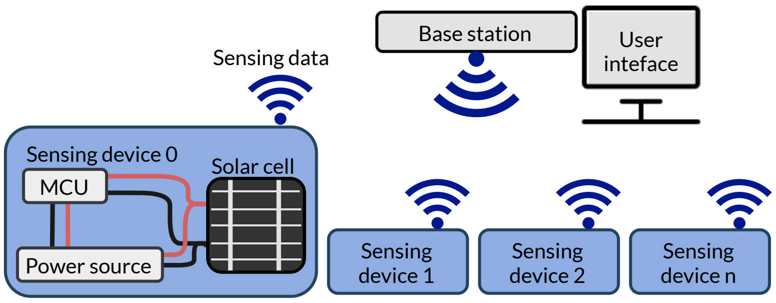


Figure 6.1: Overview of the hardware for the tracking system. The sensing devices communicate the data acquired from the solar cell to a base station over a wireless link.

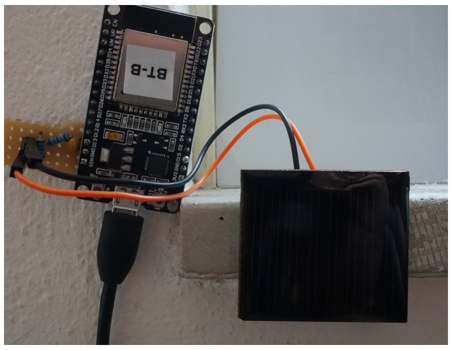


Figure 6.2: Microcontroller with solar cell attached, as used during the evaluation.

all devices, and have a low power consumption for the required data rate. The data rate depends on whether changepoint detection is executed on the device, or on the base station. In the former case, the data rate should be selected based on how fast a target is able to move through different area cells (the rate can be $\leq$ 5 Hz). In case the base station executes the changepoint detection algorithm, the required data rate is at least twice that of the cutoff frequency of the data filter (Section 4.1). In most cases, the data rate will likely not exceed more than 10-20 Hz. The most obvious choice for communication protocols include Bluetooth Low Energy (BLE) and ZigBee [16].

6.1.3 Solar cell output power

Ideally, the solar cell of the sensing device provides enough power to charge a battery that powers the microcontroller, so the battery does not deplete during its lifetime. However, the characteristic non-linear IV-curve of solar cells make this a challenge. The output power depends on the load resistance that is connected the solar cell, and the curve changes with the light intensity incident to the solar cell. This is illustrated in Figure 6.3.

Usually, a maximum power-point (MPP) tracker is used that adjusts the resistive load when the light intensity changes in order to achieve maximum power output. However, in this case this is not possible as it will also influence the ADC input, which will in turn influence the target detection algorithms. Looking at Figure 6.3, we determine that the resistance should be sized to intersect the curve of the highest expected light intensity at a voltage equal to or smaller than V_{MPP}. This guarantees the largest change in output for different light intensities, making it easier for the changepoint detection algorithms to pick up on the change. For a certain light intensity, the V_{MPP} (voltage that gives

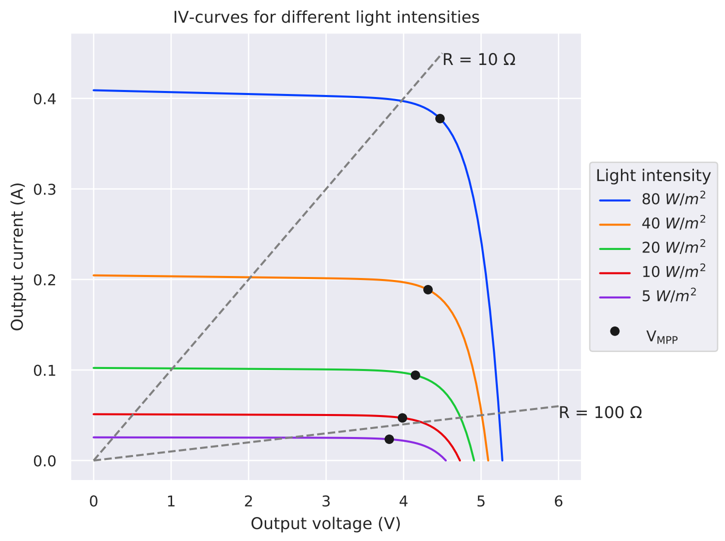


Figure 6.3: Characteristic IV-curves of an example solar cell for different light intensities. The value of the output resistance R determines the output voltage and current. V_{MPP} marks the voltage at which maximum power is reached for the specific light intensity.

maximum output power) can be approximated by Equation 6.1 [18].

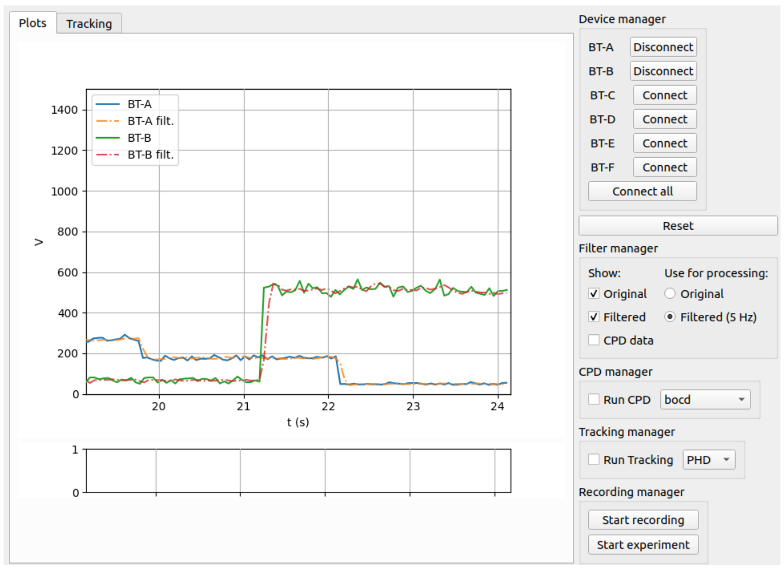
$$V_{MPP} \approx K \cdot V_{OC} \tag{6.1}$$

Here, K is a constant between 0.71-0.78 and V_{OC} is the solar cell open circuit voltage. For the experiment we used a solar cell with a limited amount of information supplied by the datasheet. The value of load resistance was set by first measuring V_{OC} directly over the solar cell terminals at high light intensity. Then, we connected resistors of different values to the solar cell terminals, measuring the output voltage of the solar cell over the resistor. The resistor value causing the output to near the value of V_{MPP} according to Equation 6.1, was considered the optimal resistor value for using the solar cell as a sensor.

Depending on the situation, the solar cell, battery and output resistance need to be sized to supply enough power to the operating device. To conserve power, the detection algorithm could be run on the devices themselves as well and periodically communicate their state of detection or non-detecting. This reduces the amount of communication activity, which is generally the highest contributor to power consumption.

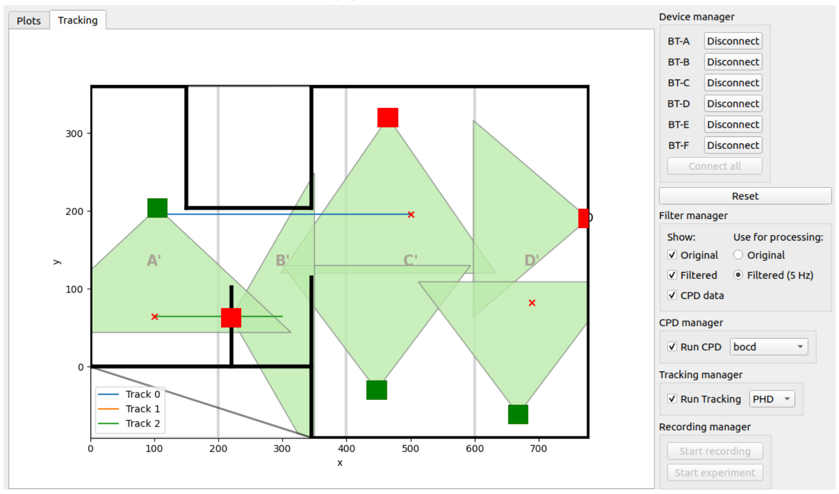
6.1.4 Base station

All processing occurs on the base station (in this case a PC). The base station runs the detection algorithms, as well as the tracking algorithms. The detection algorithm is run every time new data is received from a device. The tracking algorithm runs periodically. An interface enables the user to connect to different devices. It shows the data received by each device, and the resulting detections and tracks. Furthermore, the user is able to select which changepoint detection algorithm and which tracking algorithm is used. Figure 6.4 shows a screenshot of the interface.



Device manager
BT-A Disconnect
BT-B Disconnect
BT-C Connect
BT-D Connect
BT-E Connect
BT-F Connect
Connect all
Reset
Filter manager
Show: ✓ Original, ✓ Filtered, ☐ CPD data
Use for processing: ○ Original, ● Filtered (5 Hz)
CPD manager
☐ Run CPD bocd
Tracking manager
☐ Run Tracking PHD
Recording manager
Start recording
Start experiment

(a) Data interface



Device manager
BT-A Disconnect
BT-B Disconnect
BT-C Disconnect
BT-D Disconnect
BT-E Disconnect
BT-F Disconnect
Connect all
Reset
Filter manager
Show: ✓ Original, ✓ Filtered, ✓ CPD data
Use for processing: ○ Original, ● Filtered (5 Hz)
CPD manager
✓ Run CPD bocd
Tracking manager
✓ Run Tracking PHD
Recording manager
Start recording
Start experiment

(b) Tracking interface

Figure 6.4: User interface

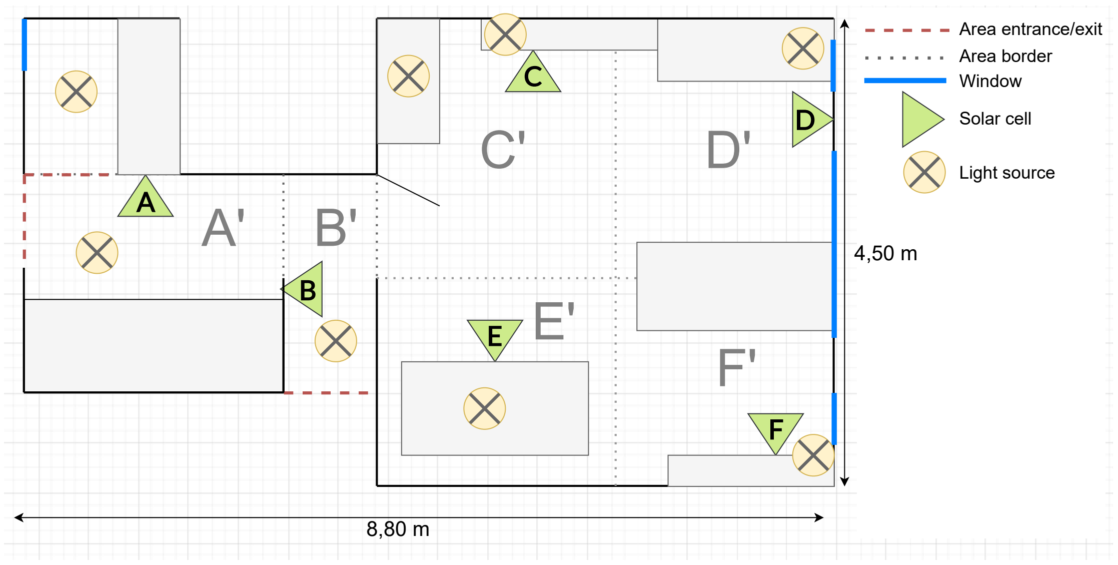


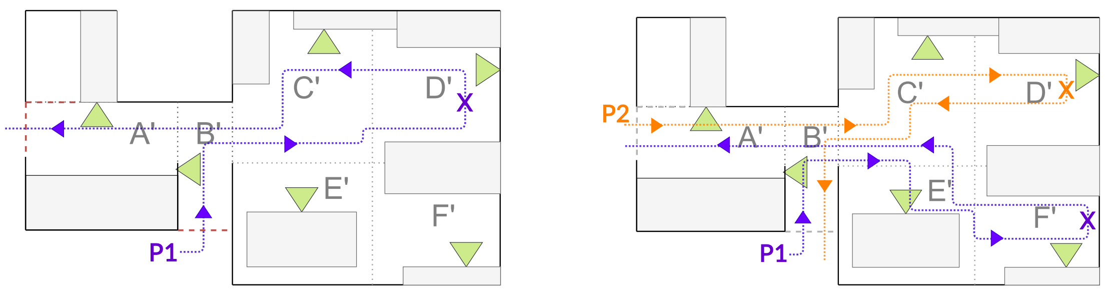
Figure 6.5: Layout of the evaluation area including positions and directions of solar cells. Some solar cells are attached to the wall, whereas others have been attached to furniture. The area has been subdivided into cells. The area cell name is based on the solar cell name that covers the largest part of the area, e.g., area A' is mostly covered by solar cell A.

6.2 Experiment configurations

The experiments were carried out in an indoor residential environment. The floorplan of the environment is shown in Figure 6.5, which also shows the positions and directions of the solar cells. The area was divided into 6 cells. The experiment was carried out multiple times while changing three variables: the number of targets (1 or 2), the light source (only natural light during the day or artificial light from lamps at night) and the distance of the target to the panels when passing by ('far' at 150 cm, or 'close' at 50 cm). Note that the far distance was not always achievable due to they physical limitations of the layout. In these cases the targets were instructed to keep the largest possible distance to the solar cell. Table 6.1 shows all the combinations of variables. During the experiment, each configuration was repeated three times. The targets were instructed to walk a fixed sequence of cells. They do this at a normal walking speed while also keeping to the requirement of passing the solar cells at 'far' or 'close' distance. The track sequences are listed below. Figure 6.6 shows these sequences.

- **1 target**
  - P1: (start outside B') - B'- C' - D' (wait approx. 5 seconds) - C' - B'- A' - (exit at cell A')
- **2 targets**
  - P1: (start outside B') - B' - C' - E' - F' (wait until P2 enters C') - E' - C' - B' - A' (exit at A')
  - P2: (start outside A', wait until P1 is at F') - A' - B' - C' - D' (wait until P1 enters B') - C' - B' (exit at B')

The data was filtered at 5 Hz, after which changepoint detection was done using the BOCD algorithm. The selected tracking algorithm was PHD because of its performance for multi-tracking, and the cleaning methods were applied every 5 timesteps.



(a) 1 target **(b)** 2 targets

Figure 6.6: Visualization of target tracks. Note that this only shows the sequence of cells visited. The actual position within the cell depends on whether the targets are instructed to pass the solar cells at 'far' or 'close' distance. An 'X' marks a location where the target stands still for a period of time.

Table 6.1: All combinations of variables used for the experiment. Note that each combination was repeated three times. A far distance is 150 cm, a close distance is 50 cm.

No. of targets	Light	distance
1	Natural light	Close
1	Natural light	Far
1	Artificial light	Close
1	Artificial light	Far
2	Natural light	Close
2	Artificial light	Close

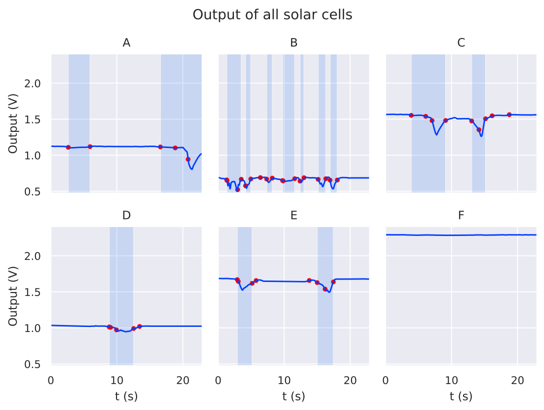
6.3 Results

In this section the results for the experiment are presented. First the results for changepoint detection are analyzed to determine whether peaks in the output caused by passing targets are succesfully detected. This is followed by an analysis of the tracking results, to see whether the actual target paths are estimated correctly.

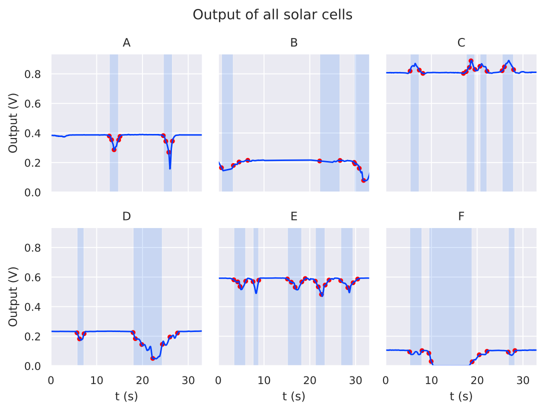
6.3.1 Detection results

The success of changepoint detection depends on two key factors: target distance and lighting conditions. Figure 6.7 shows two instances of the experiment where detection is easy due to well-defined peaks. In both instances, the targets pass the solar cells at a close distance of 50 cm. As described in Section 3, the power of incident light is proportional to the angle of incidence. A larger angle of incidence means that less power is absorbed. The closer a target is to a solar cell, the larger the angle gets at which light can pass the target and hit the solar cell. This means that the relatively high-power fraction of the incident light is unable to reach the solar cell, which results in a clear dip in the output.

Figure 6.7 also shows that peaks are more clearly defined in the scenario with only artificial light, than in the scenario with natural light. When a target blocks the direct line of sight between an artificial light source and the solar cell, it has a large impact on the output since it accounts for a major portion of the total power received. Compared to artificial light, natural light is highly diffuse as it is reflected by the sky, the ground, and other objects [15]. Because of the diffuse nature of natural light, the light reaches the solar cell from many different angles making the influence of blocking a part of

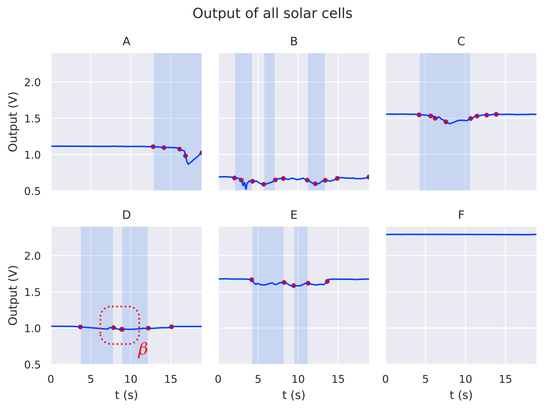


(a) 1 target, natural light, close

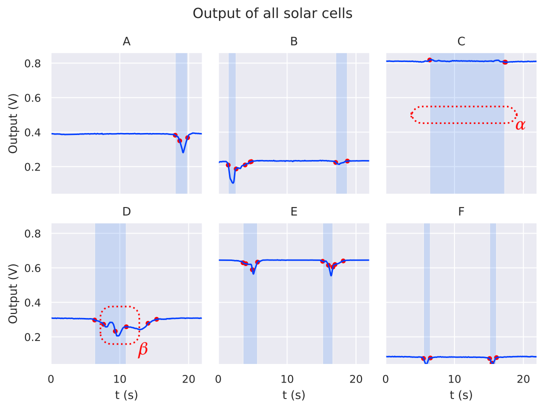


(b) 2 targets, artificial light, close

Figure 6.7: Solar cell outputs and detections for straightforward cases. The peaks are clearly defined, and are all correctly detected by the changepoint detection algorithm (marked with blue).



(a) 1 target, natural light, far



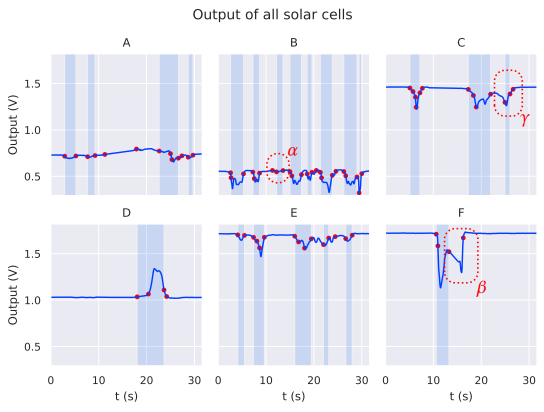
(b) 1 target, artificial light, far

Figure 6.8: Solar cell outputs and detections with some misdetections. α shows a misdetection because the end of the first peak goes undetected. β shows to cases in which a detection is too short due to smaller peaks within the large dip that falsely signal the end of a detection.

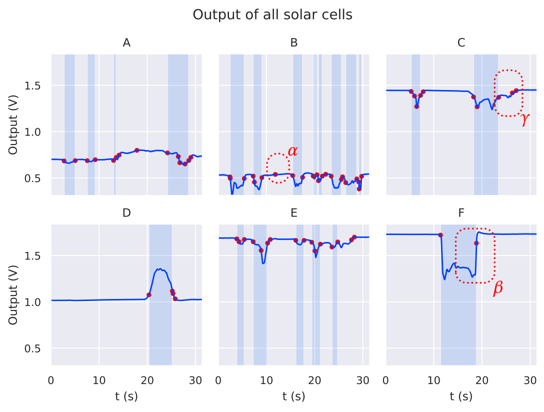
the light less noticeable.

What is important to note is that depending on the lighting situation, the same solar cell can output peaks or dips when a target passes. This can be seen in solar cell C in Figure 6.7, which has output dips during the day because light from the direction of the window is blocked, and peaks at night when a human reflects extra light from the neighboring light source. This illustrates the need of a changepoint detection algorithm that can detect both a quick increase and a sudden reduction in received power.

When looking at the detection results in Figure 6.7, it shows that for these types of scenarios the changepoint detection algorithm is able to perform well. We can clearly see dips and peaks as the target moves in the vicinity of the solar cell, with the output being constant otherwise. The start and endpoints of the peaks are correctly detected and the resulting detections (marked in blue) give a good indication of when a target is at the specific solar cell.



(a) 2 targets, natural light, close (run 1)



(b) 2 targets, natural light, close (run 2)

Figure 6.9: Solar cell outputs and detections with unclearly defined detections. α and γ show that, even though the environment and target paths are the same, the detections can differ. β shows a case where the length of a detection is marked incorrectly, probably caused by movement of the target in front of the cell.

Figure 6.8 shows two instances of the experiment that are more challenging for the changepoint detection algorithm. In both scenarios, the targets were instructed to pass the panels at a larger distance of 150 cm. This makes the peaks less clearly defined for some solar cells because a large fraction of the incident light is still able to pass the targets. The difference between the two scenarios, is that one was recorded with natural light present, and one with artificial light present. It can be seen that in the far-distance case it still holds that artificial light makes the detections better defined compared to natural light. One noteworthy example in this case is solar cell F. There is no visible difference in output in natural light, but there is in artifical light because a significant source of light gets blocked directly.

Due to the peaks being less defined, there are some misdetections present. There is a misdetection (marked α) at solar cell C in Figure 6.8b because the changepoint detection algorithm fails to detect the end of the first peak, resulting in one long detection rather than two shorter ones. In both figures, the detection of solar cell D (marked β) ends too soon because the target's movement causes smaller output variations within the large dip.

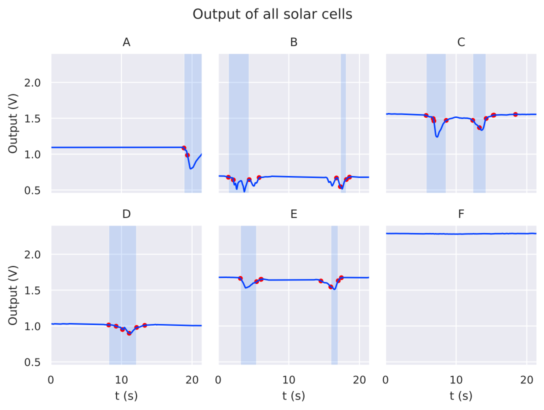
Figure 6.9 shows two different runs of an experiment scenario that gives an output for which it is difficult to visually determine the starting and end points of detections, also posing a significant challenge for the changepoint detection algorithm. In this scenario, there are two targets that pass the solar cells at close proximity. Even though the targets pass at a close distance, which should lead to clearly defined peaks, this is not always the case. Because there are multiple targets, they can simultaneously or consecutively block light from different positions. This leads to large variations in the output.

The targets' paths and the lighting conditions are the same in both plots, but the resulting detections differ. For both runs, the number of detections and the order in which they occur are different. For example, the third detection of solar cell B in 6.9a (marked α) does not exist in 6.9b. For cell F, the length of the detection marked β is determined correctly in one runs only. Towards the end of the experiment, cell C has one or two detections (marked γ), depending on the run. Knowing the true path of the targets, we know that there should be two detections because both targets pass cell C on the way out. However, by looking at the data this is impossible to say as the double peak could have also been caused by a single target passing in a certain way (which is the case for β). This shows that the performance of the changepoint detection algorithm depends on many factors and misdetections cannot be completely avoided.

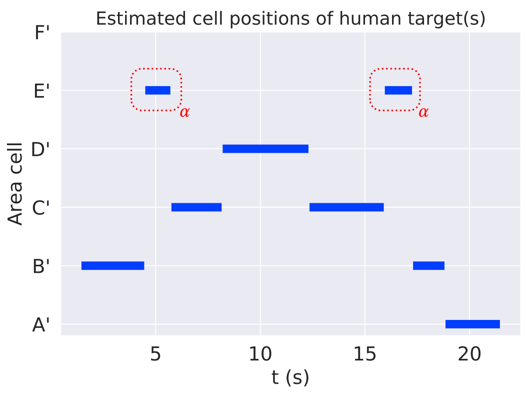
In conclusion, the performance of the changepoint detection algorithm depends on the circumstances as this influences whether the solar cell output shows clear peaks. It is easiest to correctly detect the targets when they walk by at close proximity, and in conditions with only artificial light present. Situations with natural light, or targets passing at a far distance are more challenging as peaks are less defined. However even in difficult cases, the algorithm is able to identify most peaks caused by passing targets correctly. There will be false or missed detections, but due to the nature of the data this will be unavoidable.

6.3.2 Tracking results

Next, we analyze the tracking results. Tracking gives estimates of the tracks of the targets in the environment using the detections that result from changepoint detection. Figure 6.10 shows the result for tracking with data that has well-detected changepoints. The tracking algorithm does not have to handle false detections, and the sequence of position estimates can easily be determined by following the sequence of detections from the solar cell outputs. The tracking algorithm only marks E' incorrectly twice (marked α). This causes for the estimated tracking to start with B'-E'-C', when it should be B'-C'. However, this makes sense as the person enters and exits C' at a point that directly borders E' and is considered an acceptable misdetection.



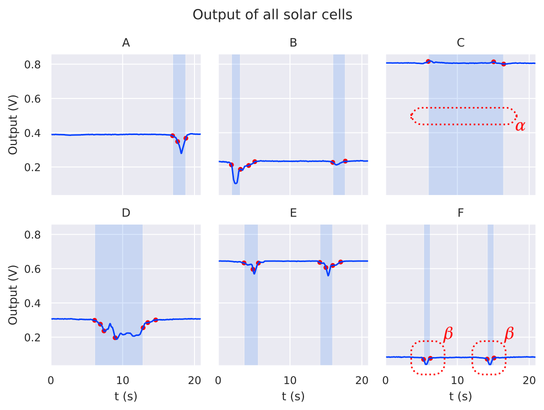
(a) Solar cell outputs with marked detections



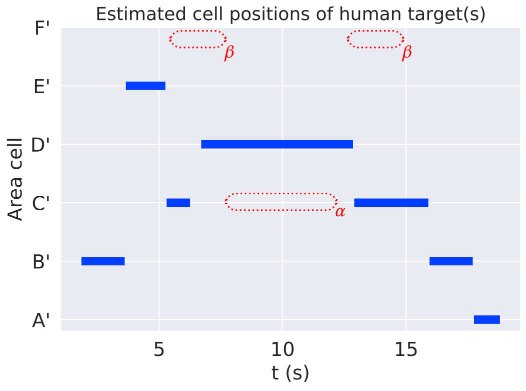
(b) Tracking output

Figure 6.10: 1 target, natural light, close - straightforward tracking due to clear and correctly marked detections. α shows a minor misdetection due to the person entering cell C' at a point that borders cell E'.

Figure 6.11 shows an instance of the result for tracking when there are misdetections present. In this case, the track cannot be estimated directly by looking at the sequence of solar cell detections. In this case, the filtering properties of the tracking algorithm becomes important. For example, solar cell C gives a misdetection (marked α). However, the tracking algorithm is able to filter this out correctly and does not estimate the target to be at position C' during this time period. Next, there are two detections by solar cell F (marked β). These are also filtered out correctly, and do not appear in the output of the tracking algorithm. Overall, the tracking algorithm successfully merges all inputs and accurately identifies the number of targets to be one throughout the duration of this experiment instance. The path is also estimated correctly except for the confusion between cell C' and E' (equal to the case in Figure 6.10).



(a) Solar cell outputs with marked detections



(b) Tracking output

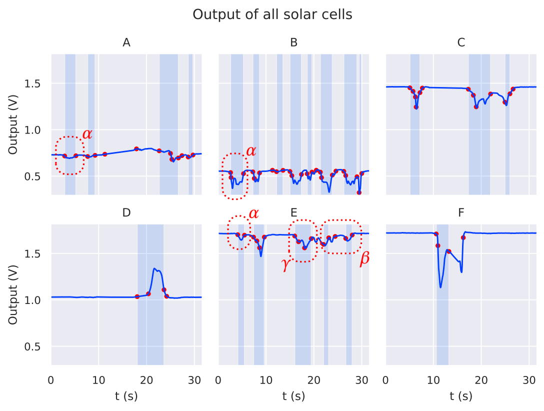
Figure 6.11: 1 target, artificial light, far - tracking with some misdetections. α shows a misdetection that was correctly filtered out by the tracking algorithm and thus does not appear in the path. β shows spurious peaks that are detected correctly, but are not caused by the target being in cell F'. This information is also correctly merged by the tracking algorithm, and the estimated path show the correct cells the target is in.

Figure 6.12 shows the performance of tracking for a difficult case . Here, there is no direct one-to-one correspondence with the events reported by solar cells and the positions of the targets. To be able to interpret Figure 6.12a better, Figure 6.12c shows a visual representation of the detection output for different time windows. In some instances, the tracking algorithm is able to filter out the large number of detections correctly. For example in the first 5 seconds, multiple solar cells output a detection (marked α), but the tracking algorithm is able to merge this information and correctly estimate the position to be B'. Between timesteps 20-30, there are some detections from solar cell E (marked β) that are caused by targets being in nearby cells. These are filtered out correctly, as they do not appear in the tracking output. The tracking algorithm is not perfect however, and sometimes is unable to correctly merge all information. Between timesteps 15-20, solar cell E has a clear detection (marked γ) but this is incorrectly filtered out which causes the path of target 1 to miss the transition from F' to E'. This result also shows the inability of the tracking algorithm to handle crossovers properly. Around the time marked by δ, the targets crossover in C'. For this period, there are no overlapping position estimates. Even at the end of this period when it correctly estimates position D' (belonging to target 2), it does not provide an estimate for target 1.

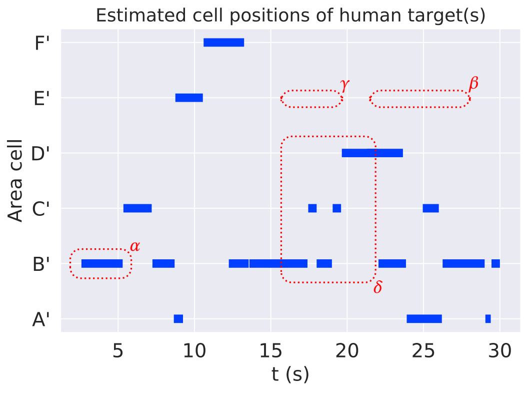
In conclusion, applying the tracking algorithm to the data improves tracking results in cases with misdetections compared to just using the sequence of solar cell detections directly.

In some cases, false positives or true positives outside of the expected detection range of a solar cell can lead to mistakes in the estimated tracks. When looking at the data, it would be simple to say that this could be solved by proper cleaning methods. However, it is impossible to make this work robustly for any case. For example, in the case of Figure 6.12b, the last estimation for B' could be cleaned to become one solid line. When doing this, you make the assumption that the target did not visit A' in the meantime. This is a perfectly realistic scenario however, as someone could step into A' for a brief moment, before changing their mind and continuing into B'. Adjusting cleaning methods so they perfectly fit the tested scenarios has a risk of overfitting. A tracking system based on binary data has a limited resolution based on the sensor density, and therefore the accuracy can better be improved by deploying a larger number of sensors.

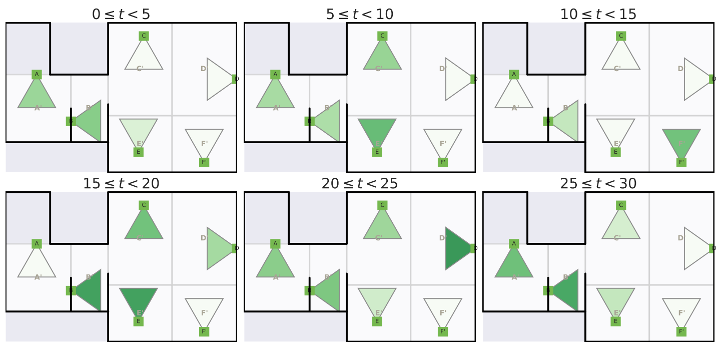
Finally, it is important to note that the tracking algorithms are based on the expected detection probability when a target is in a certain cell (the emission probability in Section 5.1.3 and the sensor likelihood in Section 5.2.1). This is based on approximations of the sensor range and also changes for different lighting situations. However, we have shown here that a coarse estimation of the detection ranges of the solar cells already enable a good result for tracking.



(a) Solar cell outputs with marked detections



(b) Tracking output



(c) Visual representation of the detection output for all solar cells for different time windows. A darker color means that the solar cell is detecting a target for a longer time period during that window.

Figure 6.12: 2 targets, natural light, close - tracking with many detections. α shows how the tracking algorithm is succesfully able to merge multiple solar cell inputs to one position estimate. β shows how spurious detections are filtered out correctly. γ shows a correct detection mistakenly filtered out by the tracking algorithm. δ marks a period where the two targets crossover in cell C' causing the tracking algorithm to temporarily only determine the position estimate of one target.

Chapter 7

Conclusion

This thesis describes the development of a multi-target tracking system using solar cells. Solar cells are predominantly used as power source, but can be used as sensors as well. This new approach enables indoor tracking that does not require extra infrastructure and is maintenance-free as well. To reach this outcome, we identified the different challenges for tracking people with solar cells. The final result can be separated into three components: a simulation for optimal cell configuration, changepoint detection for detecting people with solar cell output and multi-target tracking with binary data.

First, we developed a simulation tool that can be used to determine the optimal configuration of solar cells used for tracking. The height and angle at which the solar cells are placed influence their detection rate. The simulator evaluates which configurations are expected to perform well. The input is a model of the environment, and the output is a score for different solar cell configurations. Even though it uses a number of assumptions to simplify the ray tracing calculations, it is still able to give an idea on what good configurations are. Furthermore, this tool enables the possibility to test different tracking configurations and the result of the tracking algorithm without executing them in real life.

Second, we have developed a method for extracting a binary output from a solar cell. When a person passes a solar cell, its output changes. This magnitude and duration of the change depend on different factors. This includes the distance of the person to the solar cell, and the lighting conditions of the environment which can vary continuously. Therefore, we identified the best way of implementing a changepoint detection algorithm that can robustly detect output changes caused by passing people. For this approach we used Bayesian online changepoint detection and adapted its parameters to perform well with solar cell output. We have also identified a method based on the changing gradient of the signal, called gradient changepoint detection. This method is easier to tune, but is less resistant to environment changes.

Third, we analyzed different tracking algorithms. The algorithms needed to track multiple targets using the binary output, which provides little information, and without knowing the number of targets beforehand. Next to that, the algorithms needed to be resilient to misdetections. Two algorithms were identified as suitable and implemented. The first algorithm was based on the hidden Markov model. To make it work for multi-target systems, we derived information from the sequence of measurements to estimate the number of targets. The second algorithm was a particle filter that implemented the probability hypothesis density. This algorithm is specifically designed for multi-target tracking, and worked better. Because the output of the tracking algorithms is not perfect, we applied different data association and cleaning methods, which improved the results.

We have shown that it is possible to track up to two targets using solar cells as sensors. However, due to the limited information that binary output gives, it is difficult to make it completely accurate. It would be interesting to explore this topic further as, depending on the application, the advantage of easy deployment outweighs the limited accuracy.

7.1 Future work

This work provides a starting point for making a more robust and improved version of the system. There are different limitations of the current system that can be improved on.

1. A major aspect is that the final system used for evaluation was not powered by the solar cell. It would be interesting to determine how the solar cell should be sized to generate enough power to power the device and communication, depending on its environment. This also requires the algorithms described in this thesis to be implemented in a power-efficient manner and an analysis on what data should be communicated over the wireless connection. Moreover, a circuit needs to be designed such that a solar cell can be used for powering a battery, as well as for providing input to an ADC such as in [47].
2. The simulator is now limited to a simplified model of the environment. It gives a good indication on how solar cells should be positioned, but it is not always fully accurate. The simulator is now based on many assumptions such as diffuse reflections, simple modelling of targets and estimated reflection coefficients. More work can be done on studying the effect of integrating more types of reflections [15] and modelling the environment more accurately including the reflection coefficients of objects, and light sources. This might require the implementation of a Monte-Carlo based ray tracing method to keep the runtime of the simulation down [32, 35]. It would also be interesting to determine how the difference between natural light and artificial light can be modelled, as this turned out to have a big impact on the performance of the changepoint detection algorithms.
3. The changepoint detection algorithms currently convert the solar cell output to a binary signal. With this process, you loose the information on how much the output changes, as this could also be used (maybe in combination with other solar cells) to infer something about the distance of the target to the solar cell. Further research is needed to determine whether there is a relation between target distance and solar cell output change, that can be used independent of the lighting conditions in the environment.
4. The performance of the tracking algorithms is influenced by the accuracy of two elements: the state transition probability and the observation or measurement probability. The accuracy could be improved by basing the state transition probability on measurements, instead of basing it on the adjacency matrix of the cells. Furthermore, the measurement probability is now determined by making assumptions about the detection range of a solar cell, even though this changes when the lighting environment changes. These measurement probabilities can also be inferred from actual measurements in the environment taken over time. Further research is needed to see to what extent this can be used to improve the accuracy of the tracking algorithms.
5. The tracking system was only tested up to two targets. Next steps should focus on testing the system for more than two targets. Next to that, more work can be done on guaranteeing track continuity, as this might be important in some applications. This was not a focus point in this thesis, and therefore only simple track association and cleaning methods for track continuity were presented. Applying backtracking methods such as in [12] to 'fix' track estimates based on likely path sequences could improve the results.

Bibliography

[1] Ryan Prescott Adams and David J. C. MacKay. Bayesian Online Changepoint Detection. oct 2007.

[2] Elnaz Alizadeh Jarchlo, Xuan Tang, Hossein Doroud, Victor P.Gil Jimenez, Bangjiang Lin, Paolo Casari, and Zabih Ghassemlooy. Li-Tect: 3-D Monitoring and Shape Detection Using Visible Light Sensors. *IEEE Sensors Journal*, 19(3):940–949, 2019.

[3] Farah M. Alsalami, Zahir Ahmad, Stanislav Zvanovec, Paul Anthony Haigh, Olivier C.L. Haas, and Sujan Rajbhandari. Indoor intruder tracking using visible light communications. *Sensors (Switzerland)*, 19(20):1–14, 2019.

[4] Samaneh Aminikhanghahi and Diane J. Cook. A survey of methods for time series change point detection. *Knowledge and Information Systems*, 51(2):339–367, 2017.

[5] B. Amutha and M. Ponnavaikko. Energy efficient hidden Markov model based target tracking mechanism in wireless sensor networks. *Journal of Computer Science*, 5(12):1082–1090, 2009.

[6] Asma Azim and Olivier Aycard. Multiple pedestrian tracking using viterbi data association. *IEEE Intelligent Vehicles Symposium, Proceedings*, pages 706–711, 2010.

[7] Yaakov Bar-Shalom, Fred Daum, and Jim Huang. The Probabilistic Data Association Filter: Estimation in the presence of measurement origin uncertainty. *IEEE Control Systems*, 29(6):82–100, 2009.

[8] Keni Bernardin and Rainer Stiefelhagen. Evaluating multiple object tracking performance: The CLEAR MOT metrics. *Eurasip Journal on Image and Video Processing*, 2008(June 2014), 2008.

[9] Daniel E. Clark and Judith Bell. Multi-target state estimation and track continuity for the particle PHD filter. *IEEE Transactions on Aerospace and Electronic Systems*, 43(4):1441–1453, 2007.

[10] D.E. Clark and Judith Bell. Data Association for the PHD Filter. In *2005 International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, pages 217–222. IEEE, 2005.

[11] Aaron S. Crandall and Diane J. Cook. Coping with multiple residents in a smart environment. *Journal of Ambient Intelligence and Smart Environments*, 1(4):323–334, 2009.

[12] Debraj De, Wen Zhan Song, Mingsen Xu, Cheng Liang Wang, Diane Cook, and Xiaoming Huo. FindingHuMo: Real-time tracking of motion trajectories from anonymous binary sensing in smart environments. *Proceedings - International Conference on Distributed Computing Systems*, pages 163–172, 2012.

[13] Francesca De Rossi, Thomas M. Brown, and Tadeo Pontecorvo. Flexible photovoltaics for light harvesting under LED lighting. *2015 IEEE 15th International Conference on Environment and Electrical Engineering, EEEIC 2015 - Conference Proceedings*, pages 2100–2103, 2015.

[14] Kenneth Deprez, Sander Bastiaens, Luc Martens, Wout Joseph, and David Plets. Passive visible light detection of humans. *Sensors (Switzerland)*, 20(7), 2020.

[15] David L. DiLaura, Kevin W. Houser, Richard G. Mistrick, and Gary R. Steffy. *The lighting handbook*. 2011.

[16] Jie Ding, Mahyar Nemati, Chathurika Ranaweera, and Jinho Choi. IoT connectivity technologies and applications: A survey. *IEEE Access*, 8:67646–67673, 2020.

[17] Petar M. Djuric, Mahesh Vemula, and Mónica F. Bugallo. Target tracking by particle filtering in binary sensor networks. *IEEE Transactions on Signal Processing*, 56(6):2229–2238, 2008.

[18] D. Dondi, D. Brunelli, L. Benini, P. Pavana, A. Bertacchini, and L. Larcher. Photovoltaic cell modeling for solar energy powered sensor networks. *Proceedings of the 2nd IEEE International Workshop on Advances in Sensors and Interfaces, IWASI*, 2007.

[19] Zanyang Dong, Tao Shang, Yan Gao, and Qian Li. Study on VLC channel modeling under random shadowing. *IEEE Photonics Journal*, 9(6), 2017.

[20] Enocean-Alliance. Wireless Lighting Controls : A Total Cost Analysis. 2010.

[21] Maria Pia Fanti, Gregory Faraut, Jean Jacques Lesage, and Michele Roccotelli. An Integrated Framework for Binary Sensor Placement and Inhabitants Location Tracking. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, PP(99):154–160, 2016.

[22] Nathaniel Faulkner, Fakhrul Alam, Mathew Legg, and Serge Demidenko. Watchers on the Wall: Passive Visible Light-Based Positioning and Tracking with Embedded Light-Sensors on the Wall. *IEEE Transactions on Instrumentation and Measurement*, 69(5):2522–2532, 2020.

[23] Paul Fearnhead and Zhen Liu. On-line inference for multiple changepoint problems. *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, 69(4):589–605, sep 2007.

[24] A Fernando, R James, and Wing Chung Organic. Organic photovoltaic cells – promising indoor light harvesters for self- sustainable electronics. 2017.

[25] G.D. Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

[26] Ander Galisteo, Ambuj Varshney, and Domenico Giustiniano. Two to tango: Hybrid light and backscatter networks for next billion devices. *MobiSys 2020 - Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, pages 80–93, 2020.

[27] Divya Ganti, Weizhi Zhang, and Mohsen Kavehrad. VLC-based indoor positioning system with tracking capability using Kalman and particle filters. *Digest of Technical Papers - IEEE International Conference on Consumer Electronics*, pages 476–477, 2014.

[28] James Grant. Bayesian Changepoint Detection in Solar Activity Data. (September), 2014.

[29] Timothy W. Hnat, Erin Griffiths, Ray Dawson, and Kamin Whitehouse. Doorjamb: Unobtrusive room-level tracking of people in homes using doorway sensors. In *SenSys 2012 - Proceedings of the 10th ACM Conference on Embedded Networked Sensor Systems*, 2012.

[30] S. H. Hong, L. Wang, Z. G. Shi, and K. S. Chen. Simplified particle PHD filter for multiple-target tracking: Algorithm and architecture. *Progress in Electromagnetics Research*, 120(November 2014):481–498, 2011.

[31] Hamid Hosseinianfar and Maite Brandt-Pearce. Cooperative Passive Pedestrian Detection and Localization Using a Visible Light Communication Access Network. *IEEE Open Journal of the Communications Society*, pages 1–1, 2020.

[32] Hamid Hosseinianfar, Jie Lian, and Maite Brandt-pearce. Probabilistic Shadowing Model for Indoor Optical Wireless Communication Systems. pages 936–941, 2019.

[33] Hyunwoo Hwangbo, Jonghyuk Kim, Zoonky Lee, and Soyean Kim. Store layout optimization using indoor positioning system, 2017.

[34] Mohamed Ibrahim, Viet Nguyen, Siddharth Rupavatharam, Minitha Jawahar, Marco Gruteser, and Richard Howard. Visible light based activity sensing using ceiling photosensors. *Proceedings of the Annual International Conference on Mobile Computing and Networking, MOBICOM*, 03-07-Octo:43–48, 2016.

[35] Henrik Jensen. Monte Carlo Ray Tracing. *Realistic Image Synthesis Using Photon Mapping*, (July 2003):33–50, 2001.

[36] Soo Yong Jung, Swook Hann, and Chang Soo Park. TDOA-based optical wireless indoor localization using LED ceiling lamps. *IEEE Transactions on Consumer Electronics*, 57(4):1592–1597, 2011.

[37] Bharath Kalyan, Arjuna Balasuriya, and Sardha Wijesoma. Multiple target tracking in Underwater Sonar Images using Particle-PHD filter. *OCEANS 2006 - Asia Pacific*, 2006.

[38] Avinash Kalyanaraman, Erin Griffiths, and Kamin Whitehouse. TransTrack: Tracking multiple targets by sensing their zone transitions. *Proceedings - 12th Annual International Conference on Distributed Computing in Sensor Systems, DCOSS 2016*, pages 59–66, 2016.

[39] Daniel Konings, Nathaniel Faulkner, Fakhrul Alam, Edmund M.K. Lai, and Serge Demidenko. FieldLight: Device-Free Indoor Human Localization Using Passive Visible Light Positioning and Artificial Potential Fields. *IEEE Sensors Journal*, 20(2):1054–1066, 2020.

[40] C M Kreucher, Mark Morelande, Keith Kastella, and Alfred O. Hero III. Chapter 4 JOINT MULTITARGET PARTICLE FILTERING. *Foundations and Applications of Sensor Management (Signals and Communication Technology)*, page 328, 2007.

[41] Frank Kruger, Martin Kasparick, Thomas Mundt, and Thomas Kirste. Where are my colleagues and why? Tracking multiple persons in indoor environments. *Proceedings - 2014 International Conference on Intelligent Environments, IE 2014*, pages 190–197, 2014.

[42] John Lambert and Brian Jackson. Markov Chain Monte Carlo Multi-target Tracking. pages 1–8, 2018.

[43] Harrison K.H. Lee, Zhe Li, James R. Durrant, and Wing C. Tsoi. Is organic photovoltaics promising for indoor applications? *Applied Physics Letters*, 108(25), 2016.

[44] Kwonhyung Lee, Hyuncheol Park, and John R. Barry. Indoor channel characteristics for visible light communications. *IEEE Communications Letters*, 15(2):217–219, 2011.

[45] Tianxing Li, Chuankai An, Zhao Tian, Andrew T. Campbell, and Xia Zhou. Human sensing using visible light communication. *Proceedings of the Annual International Conference on Mobile Computing and Networking, MOBICOM*, 2015-Septe:331–344, 2015.

[46] Tianxing Li, Qiang Liu, and Xia Zhou. Practical Human Sensing in the Light. *GetMobile: Mobile Computing and Communications*, 20(4):28–33, 2017.

[47] Yichen Li, Tianxing Li, Ruchir A. Patel, Xing Dong Yang, Xia Zhou, and Co Primary Authors. Self-powered gesture recognition with ambient light. *UIST 2018 - Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*, pages 595–608, 2018.

[48] F.J. Lopez-Hernandez, R. Perez-Jimenez, and A. Santamaria. Modified Monte Carlo scheme for high-efficiency simulation of the impulse response on diffuse IR wireless indoor channels. *Electronics Letters*, 34(19):1819, 1998.

[49] Giulia Lucarelli, Francesco Di Giacomo, Valerio Zardetto, Mariadriana Creatore, and Thomas M. Brown. Efficient light harvesting from flexible perovskite solar cells under indoor white light-emitting diode illumination. *Nano Research*, 10(6):2130–2145, 2017.

[50] Dong Ma, Wen Hu, Guohao Lan, Mushfika B. Upama, Moustafa Youssef, Mahbub Hassan, and Ashraf Uddin. Solargest: Ubiquitous and Battery-free Gesture Recognition using Solar Cells. *Proceedings of the Annual International Conference on Mobile Computing and Networking, MOBICOM*, 2019.

[51] Emilio Maggio, Elisa Piccardo, Carlo Regazzoni, and Andrea Cavallaro. Particle PHD Filtering for Multi-Target Visual Tracking. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*, volume 1, pages I–1101–I–1104. IEEE, apr 2007.

[52] Ronald P.S. Mahler. Multitarget Bayes Filtering via First-Order Multitarget Moments. *IEEE Transactions on Aerospace and Electronic Systems*, 39(4):1152–1178, 2003.

[53] Khaqan Majeed and Steve Hranilovic. Passive Indoor Localization for Visible Light Communication Systems. In *2018 IEEE Global Communications Conference, GLOBECOM 2018 - Proceedings*, pages 0–5. IEEE, 2018.

[54] Sumit Majumder, Emad Aghayi, Moein Noferesti, Hamidreza Memarzadeh-Tehran, Tapas Mondal, Zhibo Pang, and M. Jamal Deen. Smart homes for elderly healthcare—Recent advances and research challenges. *Sensors (Switzerland)*, 17(11), 2017.

[55] Sebastian Matthias Müller and Andreas Hein. Multi-Target Data Association in Binary Sensor Networks. *AMBIENT 2015, The Fifth International Conference on Ambient Computing, Applications, Services and Technologies*, (c):79–84, 2015.

[56] Sebastian Matthias Müller and Andreas Hein. Tracking and Separation of Smart Home Residents through Ambient Activity Sensors. *Proceedings*, 31(1):29, 2019.

[57] K. P. Murphy. Conjugate Bayesian Analysis of the Gaussian Distribution. *Def*, 1(7):1–29, 2007.

[58] Viet Nguyen, Mohamed Ibrahim, Siddharth Rupavatharam, Minitha Jawahar, Marco Gruteser, and Richard Howard. Eyelight: Light-and-Shadow-Based Occupancy Estimation and Room Activity Recognition. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 351–359. IEEE, apr 2018.

[59] Yogesh Nijsure, Yifan Chen, Charan Litchfield, and Predrag B. Rapajic. Hidden Markov model for target tracking with UWB radar systems. *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC*, pages 2065–2069, 2009.

[60] Kusha Panta, Ba Ngu Vo, and Sumeetpal Singh. Novel data association schemes for the probability hypothesis density filter. *IEEE Transactions on Aerospace and Electronic Systems*, 43(2):556–570, 2007.

[61] Jon Peddie. Applications of Ray Tracing. In *Ray Tracing: A Tool for All*, pages 91–128. Springer International Publishing, Cham, 2019.

[62] Dorin Petreus, Christian Farcas, and Ciocan Ionut. Modelling and simulation of photovoltaic cells. *ACTA TECHNICA NAPOCENSIS*, 49(1), 2008.

[63] D. Reid. An algorithm for tracking multiple targets. *IEEE Transactions on Automatic Control*, 24(6):843–854, dec 1979.

[64] Dirk Schulz, Dieter Fox, and Jeffrey Hightower. People tracking with anonymous and ID-sensors using Rao-Blackwellised particle filters. *IJCAI International Joint Conference on Artificial Intelligence*, 1:921–926, 2003.

[65] H. Sidenbladh. Multi-target particle filtering for the probability hypothesis density. pages 800–806, 2015.

[66] Jaspreet Singh, Rajesh Kumar, Upamanyu Madhow, Subhash Suri, and Richard Cagley. Multiple-target tracking with binary proximity sensors. *ACM Transactions on Sensor Networks*, 8(1):1–26, 2011.

[67] Shuai Tao, Mineichi Kudo, Bing Nan Pei, Hidetoshi Nonaka, and Jun Toyama. Multiperson Locating and Their Soft Tracking in a Binary Infrared Sensor Network. *IEEE Transactions on Human-Machine Systems*, 45(5):550–561, 2015.

[68] Nesime Tatbul, Tae Jun Lee, Stan Zdonik, Mejbah Alam, and Justin Gottschlich. Precision and recall for time series. *Advances in Neural Information Processing Systems*, 2018-Decem(NeurIPS):1920–1930, 2018.

[69] Scott R. Thompson, Neil F. Chamberlain, and Satyanarayana V. Parimi. Using hidden Markov models to track human targets. *Sensor Fusion: Architectures, Algorithms, and Applications III*, 3719(March 1999):380, 1999.

[70] Gerrit J.J. van den Burg and Christopher K.I. Williams. An evaluation of change point detection algorithms. *arXiv*, pages 1–33, 2020.

[71] Ambuj Varshney, Andreas Soleiman, Luca Mottola, and Thiemo Voigt. Battery-free visible light sensing. *VLCS 2017 - Proceedings of the 4th ACM Workshop on Visible Light Communication Systems, co-located with MobiCom 2017*, pages 3–8, 2017.

[72] Ba Ngu Vo and Wing Kin Ma. The Gaussian mixture probability hypothesis density filter. *IEEE Transactions on Signal Processing*, 54(11):4091–4104, 2006.

[73] Chengliang Wang, Debraj De, and Wen Zhan Song. Trajectory mining from anonymous binary motion sensors in Smart Environment. *Knowledge-Based Systems*, 37:346–356, 2013.

[74] Thomas Q. Wang, Y. Ahmet Sekercioglu, Adrian Neild, and Jean Armstrong. Position accuracy of time-of-arrival based ranging using visible light with application in indoor localization systems. *Journal of Lightwave Technology*, 31(20):3302–3308, 2013.

[75] Tinghui Wang and Diane J. Cook. *Toward unsupervised multiresident tracking in ambient assisted living: methods and performance metrics.* INC, 2020.

[76] D. H. Wilson and C. Atkeson. Simultaneous tracking and activity recognition (STAR) using many anonymous, binary sensors. *Lecture Notes in Computer Science*, 3468:62–79, 2005.

[77] Robert C. Wilson, Matthew R. Nassar, and Joshua I. Gold. Bayesian online learning of the hazard rate in change-point problems. *Neural Computation*, 22(9):2452–2476, 2010.

[78] X. Xie and R.J. Evans. Multiple target tracking using hidden Markov models. In *IEEE International Conference on Radar*, pages 625–628. IEEE, 1990.

[79] Se Hoon Yang, Hyun Seung Kim, Yong Hwan Son, and Sang Kook Han. Three-dimensional visible light indoor localization using AOA and RSS with multiple optical receivers. *Journal of Lightwave Technology*, 32(14):2480–2485, 2014.

[80] Yanbing Yang, Jie Hao, Jun Luo, and Sinno Jialin Pan. CeilingSee: Device-free occupancy inference through lighting infrastructure based LED sensing. *2017 IEEE International Conference on Pervasive Computing and Communications, PerCom 2017*, pages 247–256, 2017.

[81] Muhammad Yasir, Siu Wai Ho, and Badri N. Vellambi. Indoor position tracking using multiple optical receivers. *Journal of Lightwave Technology*, 34(4):1166–1176, 2016.

[82] Shuai Zhang, Kaihua Liu, Yongtao Ma, Xiangdong Huang, Xiaolin Gong, and Yunlei Zhang. An Accurate Geometrical Multi-Target Device-Free Localization Method Using Light Sensors. *IEEE Sensors Journal*, 18(18):7619–7632, 2018.