Automated Entry System using Multi-Object Tracking

Adam Jalkemo, Emil Westenius

Master's thesis
2017:E33

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

LUND UNIVERSITY, FACULTY OF ENGINEERING

MASTER'S THESIS

Automated Entry System using Multi-Object Tracking

Authors:
Adam JALKEMO
Emil WESTENIUS

Supervisor, ASSA ABLOY:
Roger DREYER

Supervisors, LTH:
Håkan ARDÖ
Martin AHRNBOM

Examiner:
Mikael NILSSON

LUND UNIVERSITY

Lund University
Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

June 12, 2017

Abstract

In this thesis we employ computer vision methods in order to extend and improve the functionality of automatic doors. This thesis is based around the implementation of a door entrance system which uses information from detected pedestrians to make qualified decisions regarding door activation. This can lead to a reduced amount of unnecessary openings which will reduce the energy consumption of buildings. It will also increase comfort for pedestrians passing through the door. A corner stone in the proposed system is multi-object tracking for which different methods are considered and evaluated. To provide input for the tracker a range of different detection methods are evaluated and used in the system. In order to tune and test this system a dataset consisting of realistic scenarios was collected and annotated. Results in the thesis show that we can estimate the walking direction of pedestrians well while the estimated speed is quite inaccurate. We also show that because of the good direction estimate one can employ static increases in prediction time to improve performance. Our tests show that YOLO, a modern object detector, is best at detecting pedestrians. We found that a tracker of relatively low complexity, Hungarian algorithm with Kalman filtering, receives both high scores and is quite robust to noise. It is concluded that this type of method can both extend and improve the automatic entry systems used today.

Keywords: Multi-Object tracking, Pedestrian tracking, Pedestrian detection

Acknowledgements

This thesis would not have been possible without the guidance we have received from our supervisors Håkan and Martin. Your factual knowledge helped us greatly when taking on this challenging task.

We would like to thank ASSA ABLOY for allowing us to experiment with different entrance systems and providing a great environment for this work.

Finally, a special thanks to Roger for all the encouragement, support and area expertise.

Contents

Chapter 1

Background

Automatic doors are a modern convenience designed to increase comfort for users while allowing a large flow of pedestrians. The sensors used today are often based on microwave and infrared technology. The general approach is to measure deviations in output signal of the sensor to register if an object has entered a zone of interest. The sensors are good at detecting both movement and presence of moving and stationary objects. The physical constraints on the sensors used today results in limited activation zones for the controller to use when determining if the door should open. It is also very difficult to calculate the size, direction, speed and exact position of an object. Classifying objects, e.g. bicycles or cars, is very challenging with these sensors. It is therefore also hard to ignore some objects.

The system used today opens the door for pedestrians moving past the door, animals leashed outside shops and other inanimate objects moving in front of the sensor. If the sensor is too sensitive these types of openings become very frequent. This sometimes forces the tuning of the activation sensor to be very restricted which can lead to pedestrians having to reduce their velocity or even stop before passing through the door. Also, when a door opens, the power usage for the building normally increases due to increased workload for the AC/heating equipment.

The work on which this thesis is based was performed in cooperation with ASSA ABLOY Entrance Systems, AAES, which is a division within ASSA ABLOY AB. AAES is focused on development, production and sales of entrance solutions and has identified computer vision technology as an interesting research area to extend the solutions used today. Cameras can be used to extract other information than the sensors used today. By classifying an object and predicting its trajectory from a sequence of images the door control system might be able to make a more qualified decision.

Chapter 2

Problem Formulation

The purpose of this thesis is to extend and improve today's automatic entrance solutions. While today's activation sensors provide a limited range of data, a camera based system could detect pedestrians and extract large amounts of data from a video sequence. In order to achieve this the system is required to track more than one object which is known as multi-object tracking. As this is central to this problem an evaluation of which tracking algorithms are suitable for usage in an entrance system together with a camera is performed. It is investigated how an automatic entrance system can be extended by including object trajectories extracted from the tracking algorithms. Also, which type of camera and how many cameras are necessary for an efficient entrance system?

2.1 Overview

This thesis will approach the problem by constructing a baseline system and evaluating the different components. An initial assumption is that the tracker system will use detected pedestrians as inputs. Our proposed system contains components for object detection, object tracking and door control.

The object detection component is used to provide the position and possibly the velocity of detected objects. Multiple options will be considered and evaluated based on performance of the complete system.

As designing and implementing a complete entrance system is a big challenge, the focus of this paper is on the performance of different tracking algorithms. The tracking component will process detections from different frames and provide a robustness against missed and false detections. The output of the tracking component is the object trajectories which includes direction and speed.

The purpose of the door control component is to interpret the output from the tracking component and decide if the door should open or not.

2.2 Limitations

The system is designed to act as an activation sensor i.e. security sensors for reducing the risk of crushing hazards are handled separately. This work is also limited to using a binary output signal i.e. the possible output states are; open door, closed door. This simplifies the controller and is the current standard used today.

Many tracking algorithms are designed to track objects with distinct identities, this is not a focus of this thesis.

Chapter 3

Theory

This chapter begins with presenting theories for pedestrian detection followed by theories for moving object detection. Pedestrian detection algorithms are focused on single frame data and moving object detection uses multiple frames to find movement in a video sequence. These categories are not necessarily mutually exclusive and can possibly be combined for increased performance.

The final section of this chapter contains theories concerning object tracking which includes association of detections in different frames and filtering.

3.1 Pedestrian Detection

Pedestrian detections is a specialized subfield of object detection which currently is motivated by the automotive, surveillance and security industries. The Caltech Dataset [1] is a popular benchmark and like for many detection benchmarks, neural networks are currently generating the best results[1]. Currently, Fused DNN [2] has the highest reported score. There are, however, many pedestrian detection algorithms available which are less computationally demanding such as those based on deformable parts model (DPM) [3] and aggregated channel features (ACF) [4].

3.1.1 Aggregated Channel Features Based Detector (ACF)

Aggregated channel features [4] are features suited for pedestrian detection. The detection framework in the original implementation is trained on image patches of size 128 by 64 using 10 feature channels. Features are extracted and Adaboost [5] is used to train 2048 depth-two decision trees. In this thesis the detector using these features is referenced as the ACF detector.

The channels used in the original implementation is

- Normalized gradient magnitude.
- 6 channel Histogram of Oriented Gradients, see appendix A.
- The image in LUV color space.

The 10 channels are down-sampled by 4x and then smoothed before fed to the decision trees.

[1]Common Objects in Context, Detections leaderboard, (2016), Accessed 05-06-17, `http://mscoco.org/dataset/#detections-leaderboard`

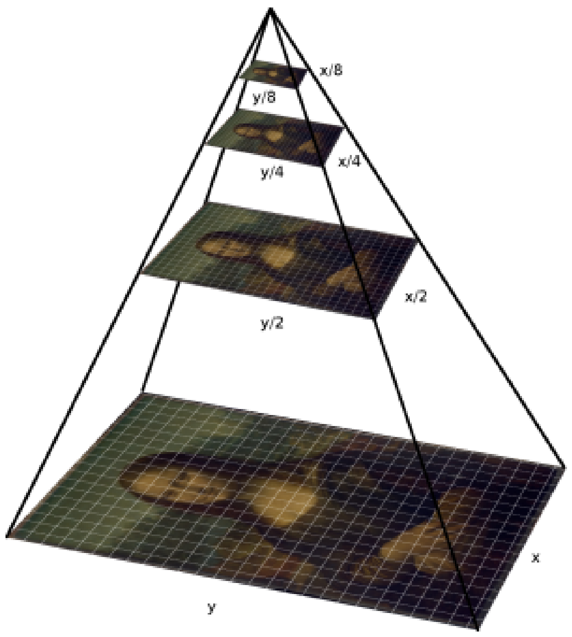


FIGURE 3.1: A scale pyramid is a set of versions of the same image in different resolutions. [6]

The object detection is performed using a sliding window at multiple scales. Additional methods to speed up computations in the ACF framework is to only calculate the features at sparse scales and approximate them at most scales. In the original implementation the computations are done once per octave (once per half image scale) and approximated for seven scales in between using the closest calculated features available using scaling power laws found in the original paper.

3.1.2 Deformable Parts Model Based Detector (DPM)

The deformable parts model approach is essentially a classifier which uses the same HOG features with the extra dimension of viewing a human as a set of different parts [3]. These parts are trained to be characteristic parts and for upright humans they are often chest, feet, head and shoulders.

Each section of the image is scored based on how much it resembles a body part and where it is located with regards to the body. The filter of the full body, root filter, is applied high up in the scale pyramid, see fig. 3.1, while the part filters are applied lower in the pyramid i.e. on a higher resolution. After a score has been calculated for each filter individually, one can combine the scores with regards to the root filter for the final score in order to classify objects, see fig. 3.2.

In order to train this classifier one typically uses a Latent Support Vector Machine [7]. This means that included in the standard Support Vector Machine are unknown, Latent, variables which are the location of the parts with regards to the root filter. These parts are not learned by observing their location in images, as the root filter is, but instead learned without any annotations for their position. These are the latent variables. The location of the part filters is used in fig. 3.2 to score the response of a filter with regards to the root filter. This allows the classifier to more reliably find pedestrians in poses which are different from the pose of the root filter.

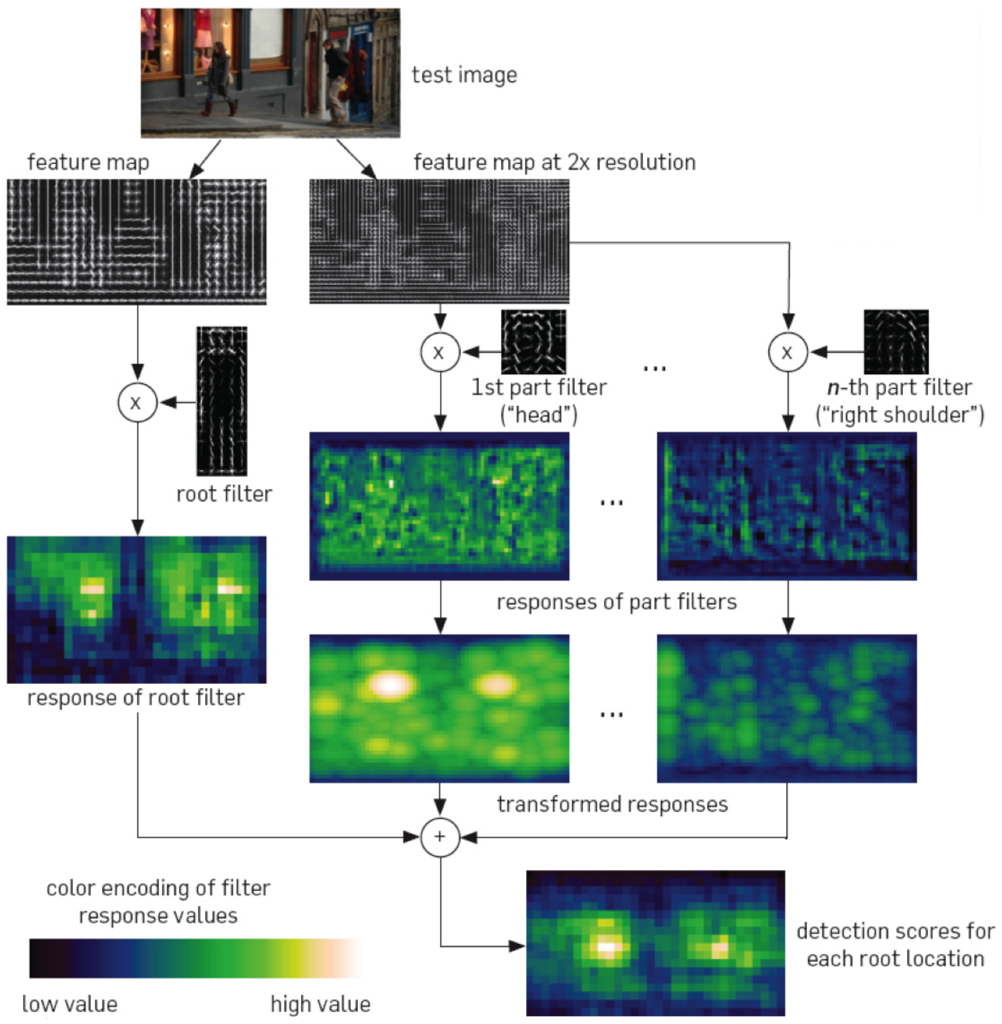


FIGURE 3.2: DPM classification pipeline where each filters is swept over different images in the scale pyramid and added together to a final detection score. The part filter response is also weighted with the deformation cost. [8]

3.1.3 Convolutional Neural Networks (CNN)

The state of the art classifiers and detectors today are based on Convolutional Neural Networks (CNN) which is a variant of Artificial Neural Networks (ANN) suitable for images. A classifier determines the class of an image (or other data) while a detector can provide the class of multiple objects in an image with localization information added. In comparison with DPM (sec. 3.1.2) and ACF (sec. 3.1.1) the features are found by training instead of using predefined features such as HOG (Appendix. A). Therefore the model and training of the network determines the quality of features and performance.

An ANN is typically composed of a number of artificial neurons. The name neuron is inspired by the neurons in the biological brain. Neurons can be connected to other neurons and the connection usually is associated with a weight and direction. The neurons are commonly grouped in layers. In a feed forward network there are no connections between the nodes within each layer and no connection from a later layer to a previous.

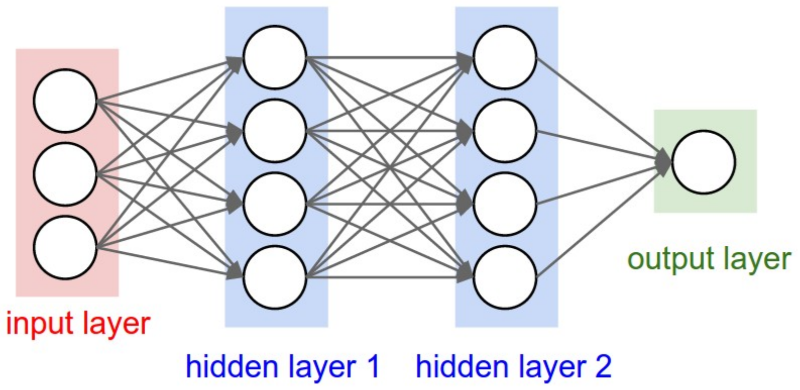


FIGURE 3.3: Artificial Neural Network [9]. Example of fully connected layers in a feed forward network. Note that there are no connections between the neurons in each layer. Each neuron propagates to each neuron in the next layer.

A typical example is the fully connected design, see fig. 3.3. Neurons commonly take input from other neurons, multiplies each input with a weight and sums them up. A bias is added to the the sum and an activation function is applied, see fig. 3.4. Example of an non-linear activation function is the Rectified Linear Unit function, ReLU, which simply changes all negative values to zero.

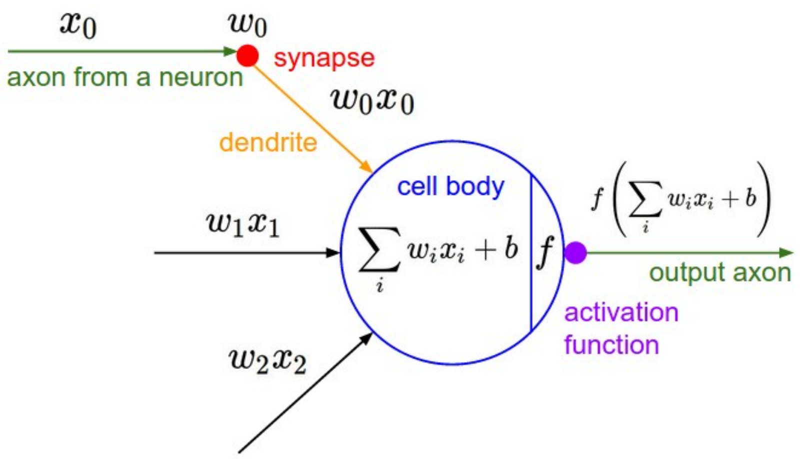


FIGURE 3.4: Artificial neuron [10] with labels from its biological counterpart. The inputs x_i from other neurons is multiplied with weights w_i and then summed. A bias b is added to the sum and an activation function f is applied.

For images, the input to the Neural Networks commonly is the pixel values in the image. The input is converted to an output for classification or regression through a combination of linear and nonlinear operations. The parameters used to calculate the output are found by training the network, commonly using stochastic gradient descent which is an adaptation of gradient descent. Stochastic gradient descent divides the data set into subsets (mini-batches), evaluates the gradient over each mini-batch and performs parameter update after each iteration.

Defining the Loss

When training the network a loss function is used to determine the error as a scalar value. This metric should then decrease as the training proceeds. The last activation function commonly used in classification where classes are mutually exclusive is the softmax activation function given by

$$\sigma(\mathbf{x})_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}} \text{ for } j = 1, ..., K,$$

where $\sigma(\mathbf{x})$ is the softmax activation function, K is the number of classes and $\mathbf{x}$ is the K-dimensional input. The softmax function normalizes the sum of the inputs to unity by assuming the inputs to be log probabilities. Cross entropy loss $L(\mathbf{x}, \mathbf{p})$ is often used to reduce the vector valued output of the softmax activation to a scalar loss. Cross entropy loss is given by

$$L(\mathbf{x}, \mathbf{p}) = -\sum_i p_i \log_e(\sigma(\mathbf{x})_i),$$

where $\mathbf{p}$ is the K-dimensional ground truth vector.

Neural networks can also be trained for regression i.e. training the network to predict a continuous value. This can be used for e.g. predicting an objects height and width. Euclidean L_1 or L_2 distance can then be used as a loss function.

Convolutions

When working with color images the input to the network is three dimensional; image width, height and the different colors. While the input could be flattened to a vector there is a lot of spatial information to gain by keeping the shape and using convolutions to perform the calculations of the neuron. This is the reason for the name Convolutional neural network. To limit the amount of parameters and therefore limit the amount of memory and data required to train the network. It is today most common to use convolutions where the images are convolved multiple times with different trainable kernels.

Backpropagation

When training the network each layer should be updated in a way which minimizes the loss function. Backpropagation (backward propagation of errors) is commonly used for this purpose. The gradient of the loss function is calculated, and the negative (decreasing) direction of the gradient is used to update the layer previous to the softmax activation layer using the chain rule. This procedure is then repeated for all layers in the network by propagating the error and making use of the chain rule.

Overfitting

Overfitting is a common problem when faced with models of many parameters. See fig. 3.5 for an example of overfitting. To mitigate overfitting one either reduces the

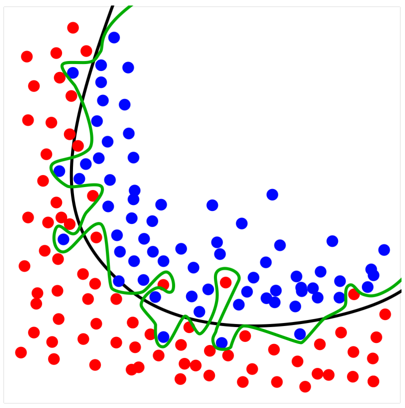


FIGURE 3.5: Example of overfitting [13]. The squiggly (green) line does not generalize well and therefore fails to fit to the curved (black) line. Instead it overfits to some individual samples which should be considered outliers. The loss on the training set will be low since all points are correctly classified but if new points are introduced, the more generalized curved (black) line would classify the set more accurately.

number of parameters, adds more training data or applies regularization. Regularization can be performed by combining the loss function with a fraction of the Euclidean L^2-norm which limits the size of the weights suppressing the risk of the network learning only single features. Other methods are dropout [11] and batch normalization [12]. Data augmentation, i.e distorting, rotating, mirroring and adding noise to the images, can also help to mitigate overfitting.

Pooling Layers

It is common to down-sample features flowing between the layers by the use of pooling layers where max- or average-pooling is often used. The features are then divided into blocks (commonly 2 by 2) and then down-sampled to the maximum value of the region (maxpool) or the average value (average pooling). This is done separately for each depth dimension. Pooling reduces the amount of parameters and calculations necessary, provides some spatial invariance and helps prevent overfitting. Maxpool is most frequently used but it is not uncommon to see networks with average pooling in the last part of the network [14] [15].

From Object Classification to Object Detection

Using a CNN to classify a single image as a single class is useful but we are faced with the problem of detecting multiple objects of one class (pedestrians) with possible overlap between objects. A solution is to classify each image patch of suitable dimension and this can be done at different image scales to account for the different scale of the objects, see fig. 3.1. The bounding boxes are then processed to account for multiple detections in one area.

Another approach of multi-object detection is to let the neural network also do the prediction of where objects are by itself. A number of different approaches exist e.g. Fast-RCNN [16], Faster-RCNN [17] and YOLO [18]. They all have in common

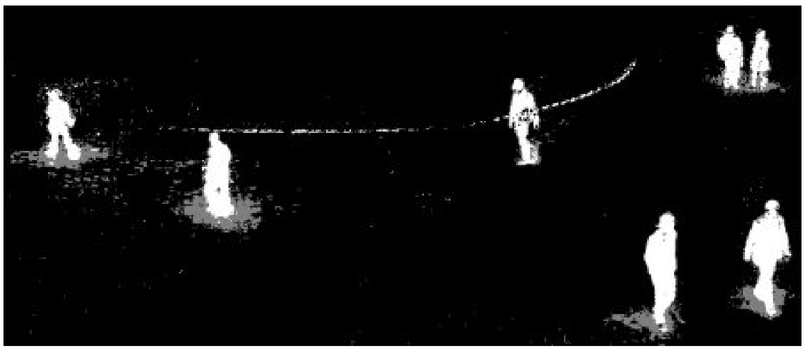


FIGURE 3.6: Example of background subtraction [20]

that they predict both object classes and bounding boxes for the objects by using regression.

In the newest implementation of YOLO [19], You Only Look Once, the network predicts possible location and size of a large predetermined number of objects, the confidence of the objects actually being objects and the class of the objects. By thresholding the confidence of the predictions, the bounding boxes which most likely describes objects are returned.

3.2 Moving Object Detection

Pedestrian detection is needed mainly when the objective is to only open doors for humans. Only if this requirement is relaxed can more general methods be used to approximate the movements of objects. Pedestrian detection methods make use of temporal data i.e. from past times and are therefore able to extract other information than the detectors described in sec. 3.1. Background subtraction and optical flow are two useful tools described in this section.

3.2.1 Background Subtraction

If the background of a number of images is constant it is possible to segment foreground from background by calculating the difference between the background image and the current image. However, due to varying lighting conditions and general changes in the scene the assumption of constant background generally does not hold. By allowing for smaller updates in the background image one can adapt to changes in lighting. If the background is represented as a Gaussian mixture model (GMM) it is possible to define appropriate pixel values and variances for the background in a varying scene [21]. When shadows are present they might be classified as foreground. This is problematic when trying to determine the position of an object since the shadow also depends on the position of the light source(s). As shadows only change the illumination of the background it possible to also segment shadows. An example of background subtraction is shown in fig. 3.6

Background subtraction assumes that lightning conditions do not change too fast. If a foreground objects share the color of the background it might not be completely segmented. If one tries to distinguish separate objects by using connected components on the output some extra post processing is useful to make sure that the bodies

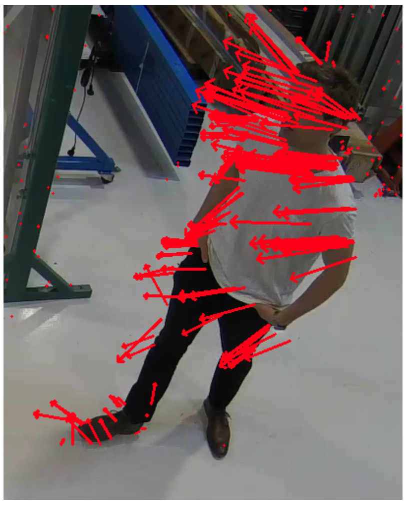


FIGURE 3.7: Example of the extracted optical flow. The arrows represents the pixel velocity amplified by 10.

are fully connected, e.g. that the head and body components are not separated due to noise. Since objects are not classified they might come in any shape or form, this means that nearby adjacent objects might be grouped as one. This can be confusing in the next stage when tracking the objects.

3.2.2 Optical Flow

By comparing two adjacent frames in time it is often trivial for humans to determine motion in the image, i.e. the optical flow. Computers are also, under certain circumstances, able to approximate the flow of pixels in a set of consecutive images. This is intuitively useful since motion is necessary for an object to reach the door and the optical flow of the pixels which make up the object should therefore be computed. The position and velocity of the pixels can then be extracted in contrast to the detectors described in sec. 3.1 which only provide position. An example of optical flow can be seen in fig. 3.7.

If the intensity of a pixel located at (x, y) at time t is given by $I(x, y, t)$ and the pixel is displaced to $(x+\Delta x, y+\Delta y)$ at time $t+\Delta t$ one can define the brightness constancy constraint

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t).$$

Using Taylor expansion the intensity is

$$I(x, y, t) \approx I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t,$$

assuming the pixel displacement is small. Simplification yields

$$\frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t = 0.$$

Dividing with Δt yields

$$\frac{\partial I}{\partial x}\frac{\Delta x}{\Delta t} + \frac{\partial I}{\partial y}\frac{\Delta y}{\Delta t} + \frac{\partial I}{\partial t}\frac{\Delta t}{\Delta t} = 0.$$

one can express it using the velocity components V_x and V_y of the pixel with the intensity $I(x, y, t)$

$$\frac{\partial I}{\partial x}V_x + \frac{\partial I}{\partial y}V_y + \frac{\partial I}{\partial t} = 0.$$

By assuming that the pixels in the local neighborhood of (x, y) has a small approximately constant displacement, one can use the Lucas-Kanade method [22] to solve this equation of two unknowns. The equation system is then on the form

$$Av = b, \tag{3.1}$$

with

$$A = \begin{bmatrix} I_x(x_1, y_1, t_1) & I_y(x_1, y_1, t_1) \\ I_x(x_2, y_2, t_2) & I_y(x_2, y_2, t_2) \\ \vdots & \vdots \\ I_x(x_n, y_n, t_n) & I_y(x_n, y_n, t_n) \end{bmatrix}, \quad v = \begin{bmatrix} V_x \\ V_y \end{bmatrix}, \quad b = \begin{bmatrix} -I_t(x_1, y_1, t_1) \\ -I_t(x_2, y_2, t_2) \\ \vdots \\ -I_t(x_n, y_n, t_n) \end{bmatrix}, \tag{3.2}$$

where n is the number of pixels in the local window used for calculations. The system of equations in eq. 3.1 is usually overdetermined and can be solved by using least squares, which then gives the solution for velocity of the pixel (x, y). The assumption of small displacement holds if appropriate scale is chosen and therefore scale pyramids are used.

3.3 Object Tracking

3.3.1 Introduction

Tracking in image analysis is a general term used to describe when an algorithm follows a certain object in a video. There are multiple ways of performing tracking and a subclass of tracking algorithms is the *Tracking-by-detection* algorithms which we primarily used in this work. This means that the algorithms use information from detected objects in each frame and associates, connects, the detections. This association can be used to extract speed and direction of moving objects as well as making the system more robust to missed detections and noise. The position and velocity of tracked objects, targets, are described by what is referred to as the target state. It is often necessary to estimate the state as noise is present and some states can be hidden, i.e. not measureable, e.g. velocity.

There are multiple ways of associating detections and in this section three methods are presented. In order for the system to have a real-time application only causal methods or causal adaptations of methods are considered. These are methods which does not use any future information.

In this work a Kalman based tracker with data association performed by the Hungarian algorithm, see sec. 3.3.4, was used as an initial tracker. The association in a Hungarian algorithm is only performed between sets of detections in consecutive frames and predictions are performed using a Kalman filter.

A natural extension of this approach is to consider a larger temporal window, i.e. using more frames in the calculation, and thus yielding more complex association results. This problem can be formulated as a graph and partitioned into trajectories e.g. with binary integer optimization, see sec. 3.3.5.

Probability hypothesis density (PHD), see sec. 3.3.6, is the third approach to perform the tracking considered in this work. PHD has similarities with Kalman filtering. However, an explicit data association is not performed but instead posterior probabilities are used to weigh which detections should be associated.

3.3.2 Maximum a Posteriori

Using Bayesian statistics, i.e. the theory which says that we can infer the probability of a measured state as a probability based on what has happened before. This is formulated in Bayes rule

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}. \tag{3.3}$$

Where A and B are event outcomes, we call $P(A)$ and $P(B)$ the prior probabilities while $P(A|B)$ and $P(B|A)$ are called the posterior probabilities.

To view this as a data association problem one can think of the posterior being the probability that a detection belongs to an identified object, track, given the detections already associated to the track. The adaptation of this can take many forms. Considered for this work was frameworks which base themselves on the idea of connecting a new detection to old detections/filters by finding the *Maximum a Posteriori*, MAP.

3.3.3 Recursive Bayesian Estimation

Recursive Bayesian estimation, or Bayes filter, can be used to recursively approximate a probability distribution by using measurements over time. The algorithm uses two steps, prediction and innovation, to predict the next state from previous estimations and to correct internal state from the new measurement. This section will begin with the single-object tracking theory and will lead to the more widely used Kalman filter.

Single-Object Bayes Filter

It is assumed that the system state x is a Markov process where the next state is only dependent on the previous state. The measurements z are the observed states of a hidden Markov model. If the state of the system at time $k-1$ is denoted x_{k-1}, the next state is given by the Markov transition

$$x_k = t_k(x_{k-1}, v_{k-1}),$$

where v_{k-1} denotes process noise. The probability of the system being in state x_k conditioned on the previous state x_{k-1} can be expressed using the Markov transition density

$$f_{k|k-1}(x_k|x_{k-1}). \tag{3.4}$$

The observations z_k at time k of the system will have a noise component w_k

$$z_k = h_k(x_k, w_k).$$

Meaning that the measurements might contain some errors due to noise. This can also be expressed by the likelihood function

$$g_k(z_k, x_k), \tag{3.5}$$

which provides the probability of measuring z_k if the target is in state x_k.

The filtering (or posterior) density is expressed as

$$\pi_k(x_k|z_{1:k}) \tag{3.6}$$

and can be used to extract current state information from the measurements up to time k. Predicting the current state from past measurements is possible by using the transition density eq. 3.4 and previous filtering density:

$$\pi_{k|k-1}(x_k|z_{1:k-1}) = \int f_{k|k-1}(x_k|x_{k-1})\pi_{k-1}(x_{k-1}|z_{1:k-1})dx_{k-1}. \tag{3.7}$$

When a new measurement then arrives, the filtering density can be expressed using the likelihood function eq. 3.5:

$$\pi_k(x_k|z_{1:k}) = \frac{g_k(z_k|x_k)\pi_{k|k-1}(x_k|z_{1:k-1})}{\int g_k(z_k|x)\pi_{k|k-1}(x|z_{1:k-1})dx}. \tag{3.8}$$

Eq. 3.7 and eq. 3.8 describes the recursive propagation of the posterior using an initial density π_0 [23].

When we are tracking a pedestrian we want to estimate its state e.g. position and velocity to make a qualified prediction and this is what eq. 3.4 provides. The true state is unknown but by measuring the position of pedestrians, e.g. the pixel position from the detector output, and using a transition model which couples the position and velocity, we can use eq. 3.6 to estimate the state. The detector might output multiple detections even when following a single object and that is where the prediction step in eq. 3.7 is of importance since it provides information about which detection is most likely.

The Kalman Filter

In the linear Gaussian case a closed form solution of the Bayes filter is the Kalman filter [24]. This filter assumes that we can describe our state transformations and observations as a linear system and also that the noise is an independent Gaussian distribution with zero mean. This allows for fast computations hence it has a large applicability. If the dynamics of the system is described by

$$x_k = F_{k-1}x_{k-1} + v_{k-1},$$

$$z_k = H_k x_k + w_k,$$

where F_{k-1} is the transition matrix and H_k the observation matrix. v_{k-1} and w_k are independent zero-mean Gaussian noise variables with covariances described by the matrices Q_{k-1} and R_k. The transition density eq. 3.9 (cf. eq. 3.4) and measurement likelihood eq. 3.10 (cf. eq. 3.5) are calculated from Gaussian densities with the notation $\mathcal{N}(\cdot; m, P)$ with m the mean and P the covariance of the distribution.

$$f_{k|k-1}(x_k|x_{k-1}) = \mathcal{N}(x_k; F_{k-1}x_{k_1}, Q_{k-1}) \tag{3.9}$$

$$g_k(z_k, x_k) = \mathcal{N}(z_k; H_k x_k, R_k) \tag{3.10}$$

For motion with constant velocity in two dimensions the state x is given by

$$x = \begin{bmatrix} p_x \\ p_y \\ v_x \\ v_y \end{bmatrix}, \tag{3.11}$$

where p_x and p_y denotes the x, y positions and v_x and v_y denotes the velocity in the x, y directions.

The transition matrix F for the constant velocity is then given by

$$F = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.12}$$

and the observability matrix

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{3.13}$$

if only the position states are observable. In the case of observability of all states, H takes the form of the four by four identity matrix.

By describing the posterior using a Gaussian distribution the posterior density at time $k-1$ is

$$\pi_{k-1}(x_{k-1}|z_{1:k-1}) = \mathcal{N}(x_{k-1}; m_{k-1}, P_{k-1}),$$

with the state x_{k-1}, mean m_{k-1} and covariance P_{k-1}. The predicted probability density eq. 3.7 for time k is then also described with the Gaussian distribution

$$\pi_{k|k-1}(x_k|z_{1:k-1}) = \mathcal{N}(x_k; m_{k|k-1}, P_{k|k-1}),$$

with the predicted mean and covariance given by

$$m_{k|k-1} = F_{k-1}x_{k-1},$$

$$P_{k|k-1} = Q_{k-1} + F_{k-1}P_{k-1}F_{k-1}^T.$$

When new measurements z_k are availabile at time k the posterior density eq. 3.8 is also a Gaussian distribution,

$$\pi_k(x_k|z_{1:k}) = \mathcal{N}(x_k; m_k, P_k),$$

with,

$$m_k = m_{k|k-1} + K_k e, \tag{3.14}$$

$$P_k = (I - K_k H_k) P_{k|k-1}. \tag{3.15}$$

Where the innovation e, Kalman gain K_k and innovation covariance S_k are given by

$$e = z_k - H_k x_{k|k-1}, \tag{3.16}$$

$$K_k = P_{k|k-1} H_k^T S_k^{-1},$$

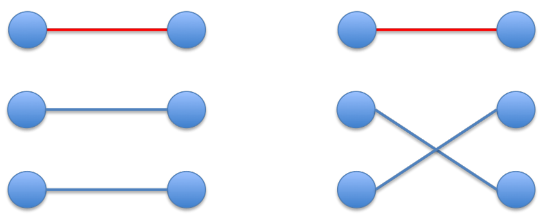
$$S_k = R_k + H_k P_{k|k-1} H_k^T.$$

When faced with non-linear systems, approximations of the system can be formed, e.g. by linearization. Existing techniques are the extended and the unscented Kalman filters [24].

To adapt the Kalman filter to pedestrian tracking, the pixel positions of the available detections are used as measurements z. An association algorithm e.g. nearest neighbor, uses the predicted state (position) $x_{k|k-1} = H_{k|k-1} x_{k-1}$ to associate one of the new detections with the current target. The innovation (error), eq. 3.16, is then calculated and used to update the target state. The target covariance is determined by the measurement and process noise covariance matrices R_k and Q_k, and these can be tuned to favor new measurements over current model state, or vice versa. If the system is assumed linear and time invariant the covariance Kalman gain, K_k, will converge to a steady state.

3.3.4 Hungarian Algorithm with Kalman Extension

When using the theory presented in sec. 3.3.2 one can find out how well a previous detection fits a new detection and use this information to track objects. When there are multiple objects the objective is instead to find the best possible associations between previous detections and new detections. This is analogous to the classical problem of a few workers with different salaries and skills completing a set of tasks. Mathematicians often introduce the subject of optimization with this problem. Assuming that there are n workers and an equal amount of tasks one can start by testing every possible combination of workers and calculate how well the overall work is performed. Solving this problem is $\mathcal{O}(n!)$, this can be observed in fig. 3.8.



(A) Assignment option 1 (B) Assignment option 2

FIGURE 3.8: Two steps in viewing all possible assignments of three nodes to three other nodes. The red link is locked and in fig 3.8a and 3.8b the options for the other two links are displayed. The rest of the combinations can be found by moving the red link between all possible combinations.

This will be a large problem to solve when the two sets grow in size. A better algorithm to solve this problem is the Kuhn-Munkres algorithm or, as it is sometimes called, the Hungarian algorithm [25]. This algorithm reduces the time complexity to polynomial time $\mathcal{O}(n^4)$ and has later been modified to run in $\mathcal{O}(n^3)$ [26].

The actual implementation is easiest to understand when using a matrix to represent the different costs of using a worker for a particular job. This yields a cost matrix as in tbl. 3.1.

TABLE 3.1: Cost matrix for three workers getting assigned to three tasks with c_{ij} being the cost for the assignment.

	t1	t2	t3
w1	c_{11}	c_{12}	c_{13}
w2	c_{21}	c_{22}	c_{23}
w3	c_{31}	c_{32}	c_{33}

To find the optimal assignment one uses the following steps on the matrix in tbl. 3.1 [25]:

1. Subtract the smallest entry in each row from all entries in that row.
2. Subtract the smallest entry in each column from all entries in that column.
3. Draw lines through the rows and columns so that each zero is covered while using the least amount of vertical and horizontal lines possible.
4. Evaluate whether the algorithm is finished: If the minimum amount of lines used is the same as the amount of tasks and workers, in this case $n = 3$, one can optimally assign each worker to a task which corresponds to a zero entry. If the minimum amount of lines is less than n the algorithm is not complete.
5. Find the smallest entry not covered by any line, subtract this number from each uncovered row and add it to each covered column. Return to step 3.

The addition and subtraction from each column/row will not change the optimal solution and the algorithm is constructed to find zeros in each column/row. Each iteration of the algorithm will yield a reduced cost matrix with at least one zero in each column. The logic behind using the least amount of lines to cover the zeros is that if one uses less than the amount of rows and columns there can be a situation where the order of assignment affects the outcome. To avoid this the algorithm iterates until the minimum amount of lines required is equal to the amount of rows. This ensures that each row can be assigned regardless of other assignments.

Tracking Adaptation

In order to utilize this algorithm for tracking the workers and tasks in the previous segment is replaced by detections in different frames. The cost of assigning one detection to another is in this thesis calculated using the distance between two detections. Solving the algorithm for each frame with the new detections getting assigned to the previous will then provide the best matching depending on their respective positions.

There are however a few downsides, it cannot be guaranteed that each pedestrian will be detected in each frame due to occlusion and detector noise. Since the algorithm demands that each detection is assigned to another detection regardless of distance, if no other option is available. This uncovers another problem, the algorithm demands equal amounts of detection in every frame, if there are more in any one frame there will be detections with no assignment.

To proficiently use the Hungarian algorithm there needs to be a possibility of new people entering and people exiting the video sequence. We will start to solve this by naming each detection in an earlier image a track. A new detection can be assigned to an old track, or it can initialize a new track. There is a possibility for a track not to be assigned any detection. This is achieved by adding dummy detections and dummy tracks. The dummy detections represents a track not receiving any new node and a dummy track represents a node starting a new track. Assuming that there are two existing (t1 and t2) tracks and three new detections (d1, d2 and d3), the dummy nodes (Dt1 and Dt2) can be found in the rightmost columns and the dummy tracks (Td1, Td2 and Td3) can be found in the bottom rows in tbl. 3.2.

TABLE 3.2: Cost matrix for three detections getting assigned to two tracks with c_{ij} being the cost for the assignment, C_{ua} is the cost for a track to be unassigned and C_{nt} is the cost of a detection spawning a new track.

	d1	d2	d3	Dt1	Dt2
t1	c_{11}	c_{12}	c_{13}	C_{ua}	inf
t2	c_{21}	c_{22}	c_{23}	inf	C_{ua}
Td1	C_{nt}	inf	inf	0	0
Td2	inf	C_{nt}	inf	0	0
Td3	inf	inf	C_{nt}	0	0

When the algorithm is finished the result will be that all tracks, detections, dummy tracks and dummy detections will be assigned. The case when a dummy track is assigned to a dummy node yields no action. Otherwise either a detection starts a new track or is assigned to an existing one and a track either receives a new detection or it does not.

Kalman Filter Extension

It can be beneficial to let tracks continue to live on even when they receive no new detections. This can be used to lower the effect of occlusion, missed and noisy detections. In order to achieve this, Kalman filtering is used. As described in sec. 3.3.3 a Kalman filter can be used as an estimator for both position and velocity where we update the filter with the measurements assigned to the specific track. With correct tuning, this filter can approximate the continued movement of a pedestrian. If a track's predicted position is used there will be a better chance of finding the correct matches when the Hungarian algorithm is applied, see fig. 3.9.

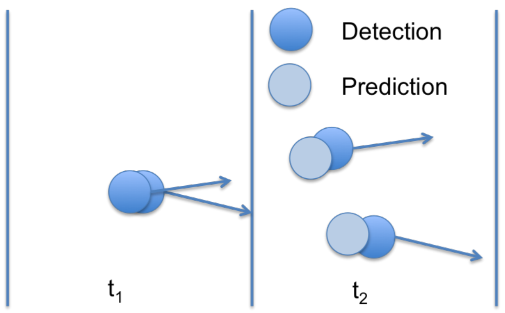


FIGURE 3.9: The detections in two time steps and the predictions from the Kalman filter in t_2. We can see that it is much easier to match the detection in t_2 to the prediction in t_2 than matching the detection in t_2 and the detection in t_1.

Pruning

If a track stops being assigned new detections it is likely that there is no pedestrian which matches the track. The removal of these tracks is called pruning and is done by considering when a filter last was matched to a detection and how many detections it has been matched to in a set amount of frames. This is a tuning parameter which is connected to the robustness of the detector and how the Kalman filters are tuned.

3.3.5 Graph Representation of the Association Problem

When extending the idea of the Hungarian method to allow for an increased temporal length there needs to be a framework in place to associate detections from multiple frames. Graph theory can be used for this. The graph is created as a set of nodes, detections, with edges between each node which is a measure of the probability that the two detections are of the same object. Depending on the method, one can use hard constraints to limit the possible connections. Two commonly used constraints are that two simultaneous detections or two detections which does not match with regards to position and velocity cannot be the detections of the same object. By associating disjoint edge paths, i.e. paths which does not share any nodes, in the graph with trajectories of objects one can now partition the graph to track objects movement. Nevatia et al. pioneered this method and solved this problem as a min-cost flow optimization problem [27].

There are two leading methods today which are based on graph theory for multi object tracking. The GMMCP-tracker developed by Dehghan, Assari and Shah [28] is based on using each frame as a cluster of objects and selecting one node from each cluster to form an optimal short track (tracklet). The tracklets will then be connected to longer tracks by using the same algorithm again. The closely related paper *Tracking Multiple People Online and in Real Time* by Ristani and Tomasi [29] is used as basis in this thesis. They also solve the problem in stages by finding tracklets which they later connect to larger tracks. The algorithm defines a few conditions which enables the algorithm to use graph partitioning for tracking. A sliding temporal window is also used which allows the algorithm to observe a subset of the information from a full video sequence. This allows the algorithm to be used in almost real-time speed.

Both algorithms presented have promising attributes but neither of the algorithms are causal and both focus on the global, in time, association problem. This means that they are developed to follow pedestrians and keep unique identities for each pedestrian over a long video sequence. In this thesis there is little use for a global solution and instead a local solution to extract position and velocity of a pedestrian is more useful. Even if both mentioned algorithms can be executed fast they cannot be used in their current state in this project. As stated before we chose the algorithm by Ristani and Tomasi as a basis for this work because of the sliding temporal window. The window feature makes the algorithm promising for conversion into a causal algorithm which find local optimal solutions.

Algorithm Overview

The actual implementation of the graph solution consists of the following steps:

1. Create a graph and assign egdes between the correct nodes.
2. Calculate the weights corresponding to each edge i.e. find how likely it is that two detections belong to the same object.
3. Set up conditions to ensure that the solution will provide useful information for track generation.
4. Solve the graph partitioning problem.
5. Use the solution to create tracklets and tracks.

The different weights can be calculated in many different ways and in this thesis they are calculated based on the distance, d, between the centers of the bounding boxes and a tuning parameter, α,

$$w = 1 - \frac{d}{\alpha}.$$

This weight, w, can become negative which together with 3.19 keeps tracklets from containing detection from multiple people. This becomes more prominent when the weight uses information of velocity, bounding box size and color histogram. A way to partition the graph is to solve the Binary Integer problem in eq. 3.17 and the constraints in eq. 3.18 and eq. 3.19 ensures that the solution is non trivial.

$$\underset{X}{\operatorname{argmax}} \sum_{(u,v)\in E} w_{u,v} x_{u,v} \tag{3.17}$$

subjected to the constraints

$$x_{u,v} \in 0,1 \quad \forall (u,v) \in E \tag{3.18}$$

$$x_{u,v} + x_{u,w} \leq 1 + x_{v,w} \quad \forall (u,v)(u,z)(v,z) \in E \tag{3.19}$$

X is the set of all the binary variables $x_{u,v}$ which decide whether an edge is "active" or not. u, v are two nodes in the graph, $w_{u,v}$ is the weight assigned to the edge and $x_{u,v}$ is the parameter which is either 0 or 1 depending on if the edge is "active" or not. At a glance one might be inclined to think that the optimal solution is to set all $x_{u,v} = 1$ for which $w_{u,v}$ are positive and 0 when they are negative, however the constraint in eq. 3.19 prevents this. The constraint states that if detection u is connected to v and v is in turn connected to z, then u must also be connected to z. The sum of their independent weights is what will determine if two or three nodes will be connected or not.

Another constraint which is placed on the system is that each node can only be connected to a maximum of one node in the previous frame and one node in the next frame. This ensures that the result only contains disjoint edge paths.

After each iteration, the oldest layer is thrown away and a new one added. We can then use the previous solution as a starting point to solve the algorithm faster.

Track Generation

The graph partitioning will provide a set of edges which connect detections between a set of frames. These edges are then translated into tracklets with 2-3 detections. These tracklets provide both a direction and an estimated speed of the object.

One can then assign these tracklets to tracks for a more stable direction and speed estimate. It is also much easier to see that two tracklets, separated with multiple time frames belong together, see fig 3.10.

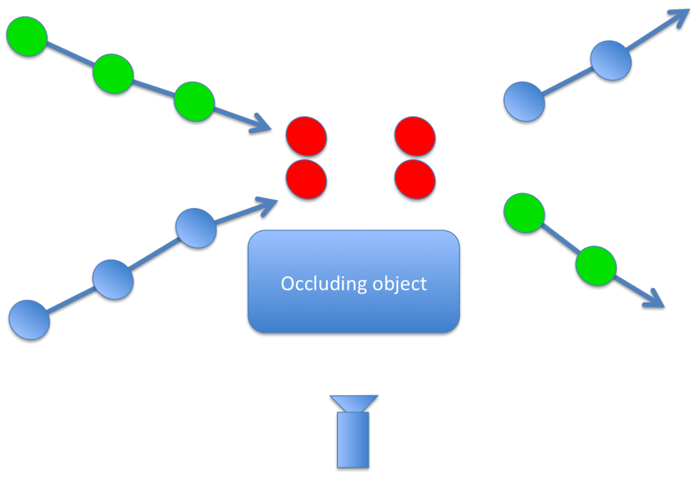


FIGURE 3.10: Using tracklets it is easier to match two objects with occluded detection to their correct new detections.

When assigning tracklets to tracks there are multiple approaches. One can try to match each new tracklet to an old track. If the tracklet does not share any part with an existing track or tracklet it will form the start of a new track. One can also link tracklets with the Hungarian method in sec. 3.3.4. These more simplistic ways of constructing tracks have a few flaws. They are unable to handle missing detections or occlusions. Another issue is that if this problem is solved in each time step there could be ambiguities in the tracks, i.e. two tracklets which contain some of the same nodes are assigned to different tracks. A way to solve this is to only assign tracklets to tracks after a full tracklet length in time. This would mean that no ambiguities will arise and there will be no issue of how to arrange the layers in the graph.

When waiting for a full tracklet before assigning to tracks there will be better direction and speed estimation which can be used in the same general outline of the above algorithm. This means that a new graph can be created with tracklets as nodes instead of detections and velocity as an added parameter. This can be repeated in order to achieve global tracks of peoples movement [29].

Causal Adaptation

In order to associate tracklets the algorithm becomes non-causal since it uses information from the entire sliding temporal window in order to associate tracklets optimally. This means that in order to find an optimal association one needs to delay the output at least the selected length of time of the temporal window, which in Ristani and Tomasi's [29] implementation is too long to be considered real-time viable. This leaves two options, either redo the association with each new frame or hold the output long enough for a good velocity estimate to be made. In both cases the temporal window is reduced to a length which allows for real-time application.

There are pros and cons with each method. If the association is done with each new frame there is little information to be extracted from new detections and the higher order associations will not provide any new information. Ambiguities in track generation can also occur if there are missed detections and a tracklet is created with different containing nodes in consecutive frames. This means that a node can in one frame be assigned to a tracklet and in another frame be assigned to a different tracklet.

If however the output is held until at least an entire tracklet can be created there cannot be any ambiguities and the tracklets can be associated to already created tracks in a more robust way. This introduces the clear drawback of having a delayed output.

3.3.6 Multi-Object Recursive Bayesian Estimation

An alternative to the explicit data association performed in both sec. 3.3.4 and sec. 3.3.5 is using Bayesian probability for the association as well. This section begins with an introduction to multi-object recursive estimation as opposed to the previously described single-object recursion described in sec. 3.3.3. This is followed by a special case of the multi-object Bayes filter, the Probability Hypothesis Density Filter and its Gaussian Mixture adaptation. The Gaussian Mixture PHD filter can be seen as a generalization of the Kalman filter in sec. 3.3.3 to multiple objects. This section is, in large, a summary of the work by Vo et al. [30].

Multi-Object Bayes Filter

The number of observations in a multi-object system may vary due to the number of targets currently being tracked. There is also the possibility of a target not generating a measurement (i.e. a pedestrian not detected) and the possibility of false detections.

Let $x_{k,i} \in \mathcal{X}$ denote the state of the i:th target at time k where $\mathcal{X}$ is the possible states and $z_{k,i} \in \mathcal{Z}$ denote the i:th measurement at time k where $\mathcal{Z}$ is the possible measurements. One can represent the states and observations at time k as finite sets:

$$X_k = \{x_{k,1}, ..., x_{k_{N(k)}}\} \in \mathcal{F}(\mathcal{X})$$

$$Z_k = \{z_{k,1}, ..., z_{k_{M(k)}}\} \in \mathcal{F}(\mathcal{Z})$$

where $N(k)$ and $M(k)$ denoted size of the sets X_k and Z_k at time k. $\mathcal{F}(\mathcal{X})$ and $\mathcal{F}(\mathcal{Z})$ denotes the possible collections of finite subsets of $\mathcal{X}$ and $\mathcal{Z}$.

The states and observations from different time steps might have different cardinality (size of the set), also, the ordering is not fixed. This means that comparing the sets from different time steps cannot be done in a trivial manner e.g. Euclidean distance.

The solution for this is to use random finite sets (RFS) [31]. A random finite set is a random variable which takes values of unordered finite sets, where the number of sets (cardinality) and their values are random according to some distribution.

The Random Finite Set theory enables the Bayesian filtering approach to be used. The advantage of this is that all information from previous measurements are taken into account in the posterior density without the need for explicit data association.

With the use of RFSs the multi-object Bayes recursion is given by

$$\pi_{k|k-1}(X_k|Z_{1:k-1}) = \int f_{k|k-1}(X_k|X)\pi_{k-1}(X|Z_{1:k-1})\mu(dX), \tag{3.20}$$

$$\pi_k(X_k|Z_{1:k}) = \frac{g_k(Z_k|X_k)\pi_{k|k-1}(X_k|Z_{1:k-1})}{\int g_k(Z_k|X)\pi_{k|k-1}(X|Z_{1:k-1})\mu(dX)}, \tag{3.21}$$

where μ is an appropriate measure of $F(X)$, since standard Euclidean distances can not be used. For information about μ the reader is referred to chapter 2.2.2 in *Random finite sets in multi-object filtering* [23]. The above prediction and update step should be compared with eq. 3.7 and eq. 3.8, f_k is the Markov transition density from eq. 3.4 and g_k is the likelihood function from eq. 3.5.

One disadvantage of multi-object Bayes recursion (eq. 3.20 and eq. 3.21) is that it is infeasible for many objects. This is due to all the possible combinations of measurements $Z_{1:k-1}$. A solution for this is to only propagate the first moment (mean) of the posterior, the intensity distribution. The intensity contains information about the expected number of targets in the region.

For a RFS X in $\mathcal{X}$ the first moment measure (or intensity measure) V for any $B \in \mathcal{X}$ is

$$V(B) = E[|X \cap B|].$$

In some cases V(B) can be expressed as

$$V(B) = \int_B v(x)dx$$

and then $v : \mathcal{X} \to [0, \infty]$ denotes the intensity function. The intensity function describes the density of the targets over the state space and can therefore be used to find the expected number of pedestrian in a region by integration.

Probability Hypothesis Density Filter

To enable tracking of multiple objects, approximations to eq. 3.20 and eq. 3.21 are necessary. Probability Hypothesis Density (PHD) Filter achieves this by only propagating the first moment of the posterior distribution, or the posterior intensity. Meaning that the new measurements are only associated with the approximation of the posterior distribution, reducing the computational complexity [32].

The following assumptions are made about the measurements:

- The measurements originating from targets are independent of each other.
- The targets move independently of each other.
- Birth targets (i.e. new objects originating) are modelled as a Poisson distributed RFS which is independent of the surviving target RFS. This is referred to as the birth RFS.
- The clutter is modeled as a Poisson distributed RFS which is independent of the measurements generated from the targets.
- The predicted multi-object RFS is also modelled as a Poisson distribution and so is the posterior RFS.

Note that these assumptions do not generally hold for pedestrians thus possibly affecting the performance of the filter. The PHD recursion is then described by

$$v_{k|k-1}(x_k) = \int p_{S,k}(x_{k-1}) f_{k|k-1}(x_k|x_{k-1}) v_{k-1}(x_{k-1}) dx_{k-1} + \gamma_k(x_k), \tag{3.22}$$

$$v_k(x_k) = (1 - P_{D,k}(x_k)) v_{k|k-1}(x_k) + \sum_{z \in Z_k} \frac{p_{D,k}(x_k) g_k(z|x_k) v_{k|k-1}(x_k)}{\kappa_k(z) + \langle p_{D,k} g_k(z|\cdot), v_{k|k-1} \rangle}, \tag{3.23}$$

where $v_{k|k-1}(x_k)$ denotes the predicted intensity (cf. eq. 3.20) and $v_k(x_k)$ denotes the updated posterior intensity (cf. eq. 3.21). $\langle \cdot, \cdot \rangle$ denotes scalar product. $P_{D,k}(x)$ denotes the detection probability, $P_{S,k}$ the survival probability and κ_k a clutter coefficient. These parameters should be tuned to match the targets interaction with the detector. γ_k is the birth distribution and can be used to model where new objects are most likely to be born, or chosen uniform depending on circumstance.

Gaussian-Mixture Probability Hypothesis Density Filter

Like the Kalman filter, a closed form solution to the GM-PHD filter is possible if the following conditions are assumed

- The target propagates according to eq. 3.9 and eq. 3.10.
- The survival probability P_S and detection probability P_D are state independent, in practice these are constants tuned to work with the detector.
- The birth RFS intensity can be represented using Gaussian mixtures,

$$\gamma_k(x) = \sum_{i=1}^{J_{\gamma,k}} w_{\gamma,k}^{(i)} \mathcal{N}(x; m_{\gamma,k}^{(i)}, P_{\gamma,k}^{(i)}) \tag{3.24}$$

where $J_{\gamma,k}$ is the amount of Gaussian mixtures components used to represent the birth distribution, $w_{\gamma,k}^{(i)}$ the initial birth weight, $m_{\gamma,k}^{(i)}$ the initial state and $P_{\gamma,k}^{(i)}$ the initial covariance. In practice the distribution can be tuned to fit where pedestrians are likely to enter the frame but also chosen fairly uniform by using high covariance.

If the updated (posterior) intensity is represented as a Gaussian Mixture,

$$v_{k-1}(x) = \sum_{i=1}^{J_{k-1}} w_{k-1}^{(i)} \mathcal{N}(x; m_{k-1}^{(i)}, P_{k-1}^{(i)}),$$

with J_{k-1} the number of Gaussian mixture components then the predicted intensity at time k is also a Gaussian Mixture

$$v_{k|k-1}(x) = v_{S,k|k-1}(x) + \gamma_k(x).$$

Where the birth distribution $\gamma_k(x)$ is given by eq. 3.24 and the survival distribution $v_{S,k|k-1}(x)$ is given by

$$v_{S,k|k-1}(x) = p_S \sum_{j=1}^{J_{k-1}} w_{k-1}^{(j)} \mathcal{N}(x; m_{S,k|k-1}^{(j)}, P_{S,k|k-1}^{(j)}).$$

The state prediction of the predicted targets is given by

$$m_{S,k|k-1}^{(j)} = F_{k-1} m_{k-1}^{(j)}. \tag{3.25}$$

The error covariance matrix for the predicted target is given by

$$P_{S,k|k-1}^{(j)} = Q_{k-1} + F_{k-1} P_{k-1}^{(j)} F_{k-1}^T. \tag{3.26}$$

Compare prediction eq. 3.25, eq. 3.26 with eq. 3.14 and eq. 3.15, which are identical update steps.

Assuming the listed assumptions hold then the predicted intensity is now a Gaussian mixture and can be represented on the form

$$v_{k|k-1}(x) = \sum_{i=1}^{J_{k|k-1}} w_{k|k-1}^{(i)} \mathcal{N}(x; m_{k|k-1}^{(i)}, P_{k|k-1}^{(i)}).$$

The update step is then also a Gaussian mixture, given by

$$v_k(x) = (1 - p_D) v_{k|k-1}(x) + \sum_{z \in Z_k} v_{D,k}(x; z),$$

with the first term representing the distribution of previous targets (with a decreased weight) and the second term the old targets (or rather target distribution) associated with new measurements.

$$v_{D,k}(x;z) = \sum_{j=1}^{J_{k|k-1}} w_k^{(j)}(z)\mathcal{N}(x; m_{k|k-1}^{(j)}(z), P_{k|k}^{(j)})$$

is a also a Gaussian Mixture where each new measurement is essentially providing P_D in total weight. The weight $w_k(z)$ is spread out so that the most probable association gets more weight according to a normal distribution (assuming the clutter coefficient κ_k is small)

$$w_k^{(j)}(z) = \frac{p_{D,k} w_{k|k-1}^{(j)} q_k^{(j)}(z)}{\kappa_k(z) + p_{D,k} \sum_{l=1}^{J_{k|k-1}} w_{k|k-1}^{(l)} q_k^{(l)}(z)}.$$

The probability of an association between an old target (or rather Gaussian mixture component) and a new measurement is now determined using the same update formulas as for the Kalman filter (sec. 3.3.3) and rewritten here for completion with regards to indices

$$q_k^{(j)}(z) = \mathcal{N}(z; H_k m_{k|k-1}^{(j)}, R_k + H_k P_{k|k-1}^{(j)} H_k^T),$$

$$m_{k|k}^{(j)}(z) = m_{k|k-1}^{(j)} + K_k^{(j)}(z - H_k m_{k|k-1}^{(j)}),$$

$$P_{k|k}^{(j)} = (I - K_k^{(j)} H_k) P_{k|k-1}^{(j)},$$

$$K_{k|k}^{(j)} = P_{k|k-1}^{(j)} H_k^T (H_k P_{k|k-1}^{(j)} H_k^T + R_k)^{-1}.$$

The derivation of the closed form solution of the Gaussian Mixture PHD filter can be found in *Random finite sets in multi-object filtering* [23].

State Extraction

Integrating over the intensity function gives the number of targets, n, currently tracked. There will be a small positive bias due to the birth RFS which is initialized with small weights and kept with probability $(1 - P_D)$ if no observation is matched. When representing the intensity function as a Gaussian Mixture one can extract the n components with largest weight and consider them as targets. In *The Gaussian Mixture Probability Hypothesis Density Filter* [30], the authors recommends extracting all Gaussian mixture components with a weight larger than 0.5 since it is likely to be a real target. If more than one target is in one location the weight will be larger proportionally to the amount of targets and the component should be extracted more than once. This is generally easier than integrating over all the Gaussian components in the the intensity function.

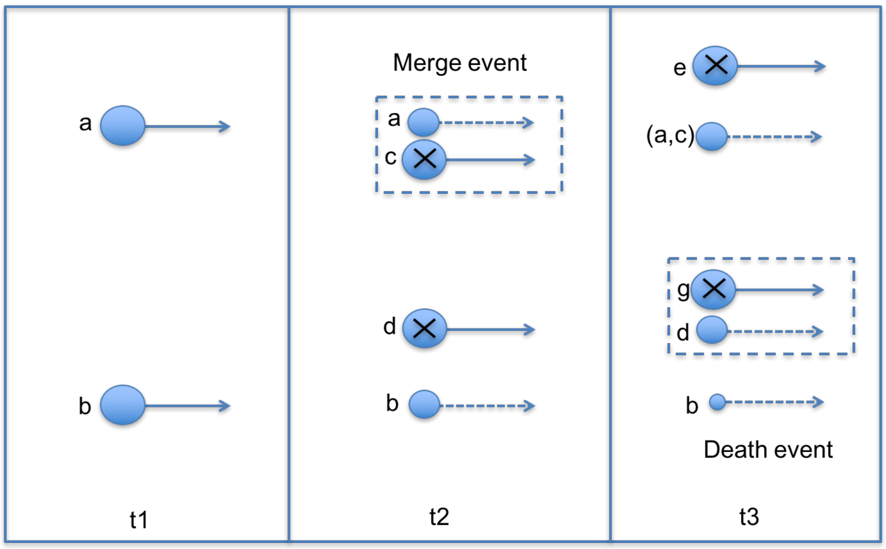


FIGURE 3.11: Illustration of merge and death events, with new detections marked X. At t_1 two targets (a,b) are tracked. At t_2 the two targets still exist but with a decayed weight $w_{t_2} = w_{t_1}(1 - P_D)$. Additionally, two new detections are found (c,d). The top detection (c) is very close to the prediction of the old target (a) so a new component (a,c) with a high weight is created and merged with the old target due to the similarity between the states. The bottom detection (d) is however far away from the old target (b) which subsequently dies at time t_3 as a result of a truncated weight. Similar processes are repeated in t_3.

Managing the Components - Merging and Pruning

Since each new measurement is associated with each old target a lot of new Gaussian mixture components are created. If one does not manage the components by removing unnecessary ones and merging components with similar states the number of components will approach infinity.

The associations with low probability are assigned a low weight. If the weight is below a truncation threshold it should be deleted. See fig. 3.11 for both a merge and a death event. Next the distance between each state should be determined. If the distance between the states is lower than a distance threshold they are merged. The distance d is calculated by

$$d = (m_k^{(i)} - m_k^{(j)})^T (P_k^{(i)})^{-1} (m_k^{(i)} - m_k^{(j)}),$$

where the index j denotes the component with largest weight in the current set and i the index of the component to be compared to. Merged components are removed from the current set and a new index j is selected by finding the largest remaining component. The merged weight, state and covariance matrix are calculated according to

$$\tilde{w}_k = \sum_{i \in L} w_k,$$

$$\tilde{m}_k = \frac{1}{\tilde{w}_k} \sum_{i \in L} w_k^{(i)} m_k^{(i)},$$

$$\tilde{P}_k = \frac{1}{\tilde{w}_k} \sum_{i \in L} w_k^{(i)} (P_k^{(i)} + (\tilde{m}_k - m_k^{(i)}))(\tilde{m}_k - m_k^{(i)})^T,$$

where L is the set of all weights within the distance threshold. It is also possible to specify a maximum capacity i.e. the maximum number of components kept. This is done by keeping the components with the highest weight.

Chapter 4

Methodology and Implementation

4.1 System Setup

The system setup consists of an object detector, a tracking algorithm and a door controller as can be seen in fig 4.1. There are other components used when performing tests which are explained later in the report.

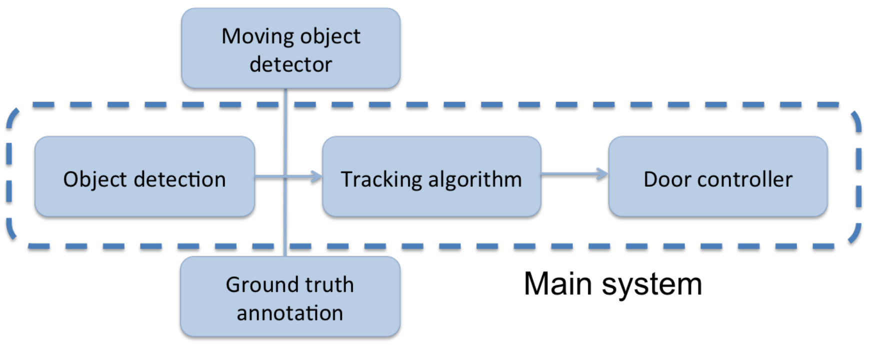


FIGURE 4.1: Main pipeline for image processing and door control. The project is focused around the components in "Main system" with added components for tests and evaluation.

4.2 Software

- *Python* – The project is primarily implemented in the programming language Python as many robust image analysis and mathematical toolboxes are available.
- *Swig* – Software development tool to wrap c/c++ code for usage in Python [33].
- *OpenCV* – Open source computer vision library available for both c/c++ and Python [34].
- *Gurobi* – An optimization tool for Linear and Quadratic programming along with Mixed Integer programming and constrained versions of each optimization type [35]. In this project Gurobi is used for the algorithm in sec. 3.3.5.

4.3 Camera Setup

4.3.1 Camera Setup

In order evaluate our complete system in different situations it was necessary to record our own dataset. This was done using an Axis M3046-V network camera.

We decided to mount a camera centered above the door with an angle that exposes as much of the pedestrians as possible, see fig. 4.2. Depending on the height of the camera mount we are then able to see pedestrians further away. Because of this a tall door is better for a larger field of view and therefore yields more time to reliably track the pedestrians.

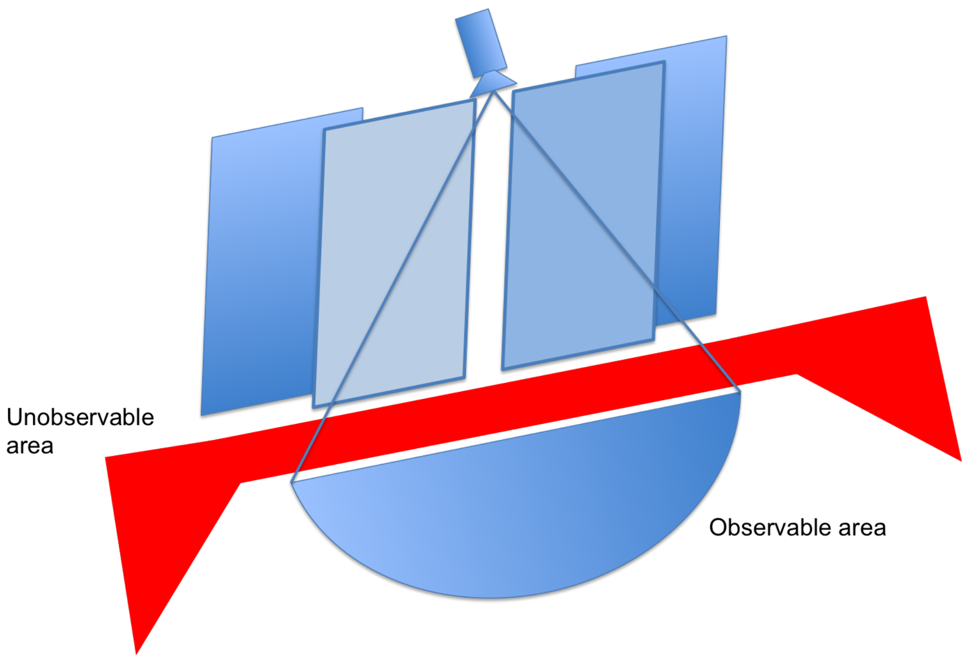


FIGURE 4.2: The chosen camera placement with depicted observable and unobservable areas due to camera selection and placement. Note that the camera is mounted at an angle.

Mounting the camera close to the door has the advantage that the camera mount is more agnostic to the surrounding environment. Cables can be shorter and the power supply can be the same as for the automatic door. There are however advantages of mounting the camera away from the door. Assuming there is a ceiling to mount the camera in, one would get a top view of the pedestrians from which the positions and velocity could be easier to estimate. Assuming there is a wall perpendicular to the door one could mount the camera there and get a good estimate of the y-position, which could be useful in a corridor setting.

The selection between using a mono or stereo vision setup was not obvious. Stereo vision cameras has the advantage of seeing depth which would give valuable 3D-information. It does however come with extra cost and possible complexity so therefore we decided to limit ourselves to mono vision cameras. The reasoning is that if the camera is mounted at an angle which provides enough top-view the position in the room coordinates should be available in the image, see fig. 4.3.

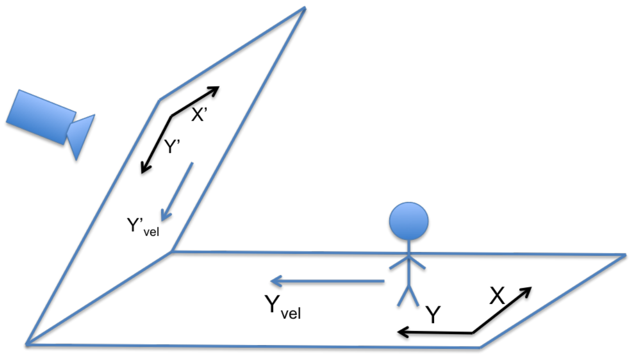


FIGURE 4.3: The 2D image projection of the 3D world. In this work only the x- and y- coordinates are of interest. The hypothesis is that a persons position and velocity can be estimated form a set of consecutive images.

Multiple cameras can also be utilized to cover both the unobservable areas in fig. 4.2 and help find a good estimation of the correct velocity as presented in fig. 4.3. In this project we decided to use one camera with a very wide lens in order to reduce complexity introduced by image stitching and camera calibration.

The velocity of a pedestrian is calculated from a video sequence in the unit pixels/second. This velocity is in the image coordinate space rather than the room coordinates, see fig. 4.3.

The camera used has a vertical field of view of 72° and a horizontal field of view of 128°. Increasing the field of view allowed the system to track a pedestrian further away and minimizes dead zones. A disadvantage is however that the image is rather warped in the edges of the image.

When pedestrians are under the camera it gives a top view of them. While this provides a better position estimate it does have the disadvantage of making detections harder since less of the human features are visible. This can in future work be mitigated by using detectors specifically for this purpose in these areas.

4.4 Pedestrian Detection

In this section we will present how the theories presented in sec. 3.1 are implemented. Although minor attempts were made to train our own detectors, pretrained detectors performed better and were therefore used in the implementation.

4.4.1 Deformable Parts Model Based Detector

The Deformable Parts Model based detector implementation in OpenCV was used. It is a shallow learning algorithm designed to handle the different poses of a pedestrian. The detector was trained on the INRIA data set [36]. A way to increase performance is to feed the detector the same image in different angles.

As this algorithm is only available in the C/C++ version of OpenCV it was implemented with a Swig wrapper to work in Python.

4.4.2 Aggregated Channel Features Based Detector

An ACF based detector implementation trained on the INRIA dataset [36] can be found in the author P. Dollar's MATLAB toolbox [1]. The C++ implementation of the ACF framework that we used can be found here [2]. The default threshold of 50 was lowered to 20 to increase the recall, see sec. 4.9, of the detector. A way to increase performance is to feed the detector the same image in different angles.

4.4.3 Convolutional Neural Network - YOLO

Source code for YOLO is available together with the open source neural network framework Darknet [3]. Networks trained on MS COCO [37] and VOC Pascal 2007+2012 [38][39] are available as both full-size models and smaller faster models with the name Tiny YOLO. To run the detectors on our dataset we took advantage of the Amazon Elastic Compute Cloud since a high-end GPU is required to run the detector at reasonable speeds. The detections are saved to textfiles and later read by Python for evaluation. Only objects classified as persons where used.

4.5 Moving Object Detection

Another approach to track objects and estimate velocities is to make use of background subtraction and optical flow as described in sec. 3.2.1 and sec. 3.2.2. Since both methods use temporal information they provide an interesting comparison to using an object detector. Especially optical flow methods have the advantage that velocity can be extracted and therefore provide additional information for the tracker.

First, a foreground mask is extracted using background subtraction. Feature points are randomly assigned to foreground pixels and tracked backwards in time using the Lucas-Kanade [22] optical flow algorithm in OpenCV. The number of feature points used is chosen proportional to the mass of foreground pixels.

Clustering using the algorithm Density-Based Spatial Clustering of Applications with Noise, DBSCAN [40], is then performed on both the position and velocity of the feature points. The implementation of DBSCAN in scikit-learn [41] is used. Clusters with few points are discarded, effectively filtering outliers. The mean position and velocity of the points in each cluster is then calculated and used as an input to the Hungarian-Kalman tracker.

[1]P. Dollar, Piotr's Image & Video Matlab Toolbox, (2016), GitHub repository, https://github.com/pdollar/toolbox

[2]Andrea Pennisi, The Fastest Pedestrian Detector in the West, (2017), GitHub repository, https://github.com/apennisi/fastestpedestriandetectorinthewest

[3]J. Redmon, Darknet, (2017), GitHub repository, https://github.com/pjreddie/darknet

4.6 Object Tracking

4.6.1 Hungarian Algorithm

The tracker was implemented in Python with the Munkres implementation in pypi as the Hungarian solver. To use the calculated associations a Kalman filter was implemented after the standard Kalman equations described in sec. 3.3.3 and the weights were generated by calculating the L^2-norm for the distance between different bounding box midpoints. The tuning of the tracker includes tuning of the Kalman filter, the pruning rules, the weight generation, the costs for tracks not receiving new detections and the cost of spawning a new track.

4.6.2 Graph Partitioning Algorithm

The implementation of this algorithm was done in Python, and was based on the work of Ristani and Tomasi [29]. The graph framework consisting of nodes, edges and iteration methods used is a software package for python called Networkx [42]. As stated in sec. 3.3.5, the optimization is done with the Gurobi module, see sec. 4.2. Tuning of this tracker included tuning of the weights used, track generation and velocity estimation method.

4.6.3 Probability Hypothesis Density Filter

The PHD filter was implemented in Python. The reference implementation by Dan Stowell [4] inspired by the pseudo code from *The Gaussian Mixture Probability Hypothesis Density Filter* [30] was modified to fit our needs. The center of the bounding boxes are used as position measurements. The tuning of this tracker includes tuning of the filters, birth and clutter distributions, detection and survival probabilities.

4.7 Door Controller

In order to determine if the door should open, the output from the tracker has to be processed. The component responsible for this is the door controller. It is designed as an activation trigger with regards to the current position and velocity of the tracked targets. The component in this work uses the predicted future position together with a set of binary rules to trigger an activation.

In order for the entrance system to perform its task well the controller should:

- Predict an object trajectory and open accordingly.
- Identify and ignore traffic not intending to pass through the door.
- Identify slow moving objects close to the door for which a future position is hard to predict and act accordingly.
- Handle multiple moving objects.

[4]D. Stowell, GM-PHD filter implementation in Python, (2013), GitHub repository, https://github.com/danstowell/gmphd

The controller is implemented in Python as the last component which should communicate with a door.

How far in time the pedestrian trajectory should be predicted depends on the time it takes for the door to open, in our experiments we have assumed it takes two seconds unless stated otherwise. In the optimal case the target is tracked before the door has to open. When the predicted position and velocity is within a defined interval the controller opens the door, see sec. 4.7.1.

In order to predict position we assume movement with constant velocity and the controller can then employ linear extrapolation from the position and velocity.

4.7.1 Activation Trigger

The controller triggers an open signal when the predicted pedestrian position is beyond the door but within the door frame when entering, see fig. 4.4. This is designed to ignore cross-traffic (see fig. 4.6) and open for pedestrians heading for the the door.

If an object moves very close to the door and stops e.g. when entering the camera view from the side and only generating velocity parallel to the door, the activation method described above will not open the door. This was observed in preliminary tests and in order to improve the controller, a proximity zone was assigned to the door, see fig. 4.4. Within the proximity zone pedestrians with low velocity will trigger a door activation as they are assumed to be heading through the door.

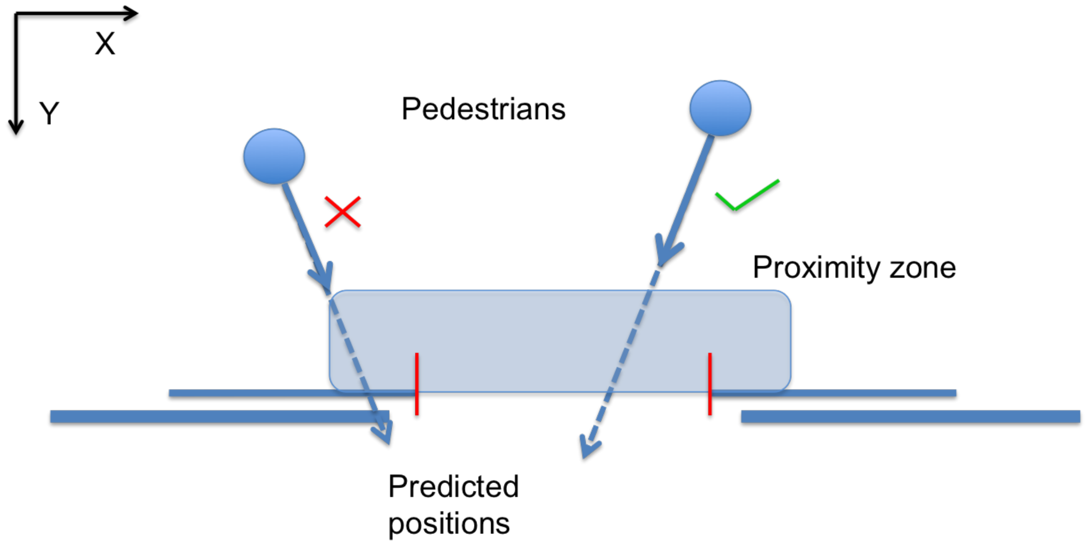


FIGURE 4.4: Top view of door with tracker and controller output. If the predicted position of a pedestrian is past the door and inside the door frame when crossing the door, the door should open. The pedestrian to the left is not predicted to end up within the door frame and therefore the door will not open. The pedestrian on the right is predicted to walk through the door which therefore should open.

4.8 Datasets

4.8.1 Dataset Construction

In order to evaluate the performance of the different trackers a custom data set was required. This dataset was divided into categories depending on the type of movement:

- SM - Straight movement towards the door, see fig. 4.5
- SA - Straight movement from an angle, see fig. 4.5
- CT - Cross traffic i.e. not heading for the door but instead walking past it, see fig. 4.6
- TU - Pedestrians not heading for the door, turning, then entering the door, see fig. 4.6

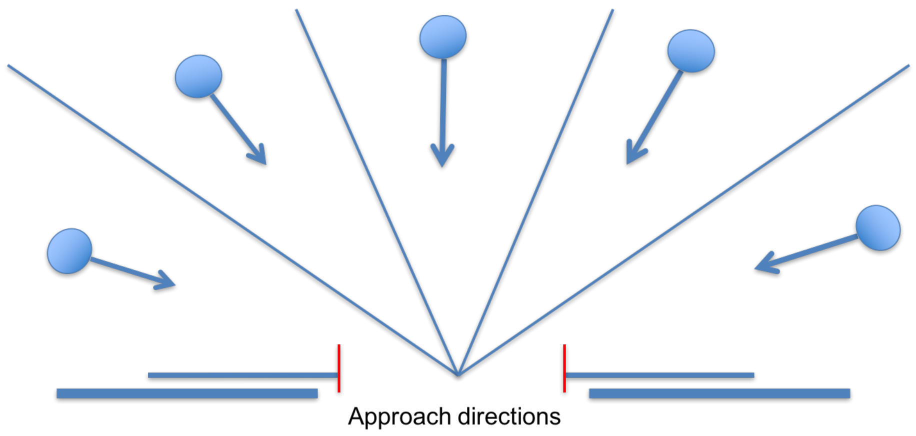


FIGURE 4.5: Approach directions when the pedestrians are following a straight path. The trajectory in the middle is labeled SM and the other trajectories are labeled SA.

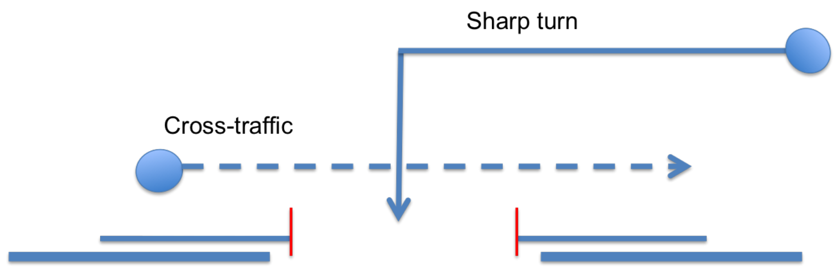


FIGURE 4.6: Cross-traffic is defined as motion approximately parallel to the door and perpendicular to the door opening. It is used to describe a situation where the pedestrian walks past the door with no intent of entry. The "Sharp turn" action of a pedestrian is difficult to predict due to its similarity with cross-traffic

The first categories, SM and SA, are aimed at capturing predictive behavior with pedestrians moving at different velocities. The CT category was used to see how well the system can identify pedestrians which are not headed for the door. When pedestrians turn towards the door there is little time to find a good velocity estimate, this is captured in the TU category. Some of the videos feature occlusions which is normal in the dynamic environment in front of a door.

The data set consists of 84 annotated clips distributed over the categories above. 32 extra clips were annotated specifically for training i.e. tuning of the trackers, detectors and controller.

4.8.2 Annotation

Two kinds of annotations were necessary; bounding box annotation and open time annotation. Both annotations are aimed to be used to evaluate the systems performance.

Open Time Annotation

The open time annotation consists of determining the shape of the region of interest in front of the door and in determining when in the video sequence the door should be open. A pedestrian can make one of two types of entries, a predictable or an unpredictable. A straight entry is a good example of a predictable entry. This type is annotated so that an open signal should be sent a fixed time interval before the entry is made. This fixed time is set to two seconds and is evaluated in later sections, see fig. 4.7. An example of an unpredictable entry is a sharp turn, see fig. 4.6. The annotation in this case is done so that as soon as there is movement towards the door it should open, see fig. 4.7.

Box Annotation

The ground truth annotation is done with the help of OpenCV and an annotation script in Python.

The user first suggests a person and the KCF-tracker in OpenCV is used in an attempt to follow the person through the video sequence. This is done so that the user does not have to manually annotate each frame. The tracker is not very stable and has trouble with drifting and changing the size of the bounding box. As a consequence the user will have to re-do the bounding box when needed. This can lead to induced velocities for objects and unreliable bounding box features. The most reliable features extracted from ground truth is the bounding box center which is used by all trackers.

4.9 Evaluation

This thesis is focused on the tracking performance which therefor is the primary evaluation point. The normal way of measuring a trackers performance is to measure how well it can keep the identity of pedestrians in a video sequence, this requires a well annotated data set with exact identities and bounding boxes for each person. That kind of data set is available in the public space but not with the angle and behavior which are relevant for this work. In order to remove the need to annotate a data set with exact bounding boxes and person identity the scoring in this

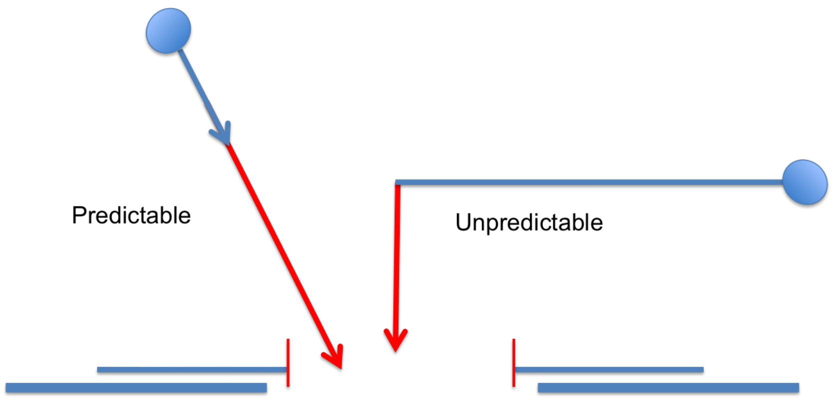


FIGURE 4.7: Entries are divided into unpredictable and predictable. Unpredictable is when the pedestrian turn towards the door unexpectedly. Predictable is when a pedestrian can be assumed to pass through the door and the open time annotation is set to a predetermined time before entry.

work is instead based on how well the entire system can open a door. This is done with the open time annotation in sec. 4.8.2.

Evaluation Metrics

The dataset is annotated with each frame having a true state, open (Positive, P) or closed (Negative, N). The implemented system provides an output signal, open or closed, which is compared to the ground truth of the dataset. The output signal is labeled true if it is correct and false if it is wrong. The possible labels for an output are True Positive (TP), False Negative (FP), True Negative (TN) or False Negative (FN).

The different metrics used to evaluate the performance are listed below.

The Recall - The fraction of the frames labeled open, P, which was correctly classified, TP. Defined as

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{TP}{P}$$

The Precision - The fraction of the total amount of times the system output an open signal, TP + FP, and it was correct, TP. Defined as

$$\text{Precision} = \frac{TP}{TP + FP}$$

The F1 Score - This score is a good way to combine the precision and recall since both often should be optimized. The F1 score is the harmonic mean of the precision and recall. It is defined as

$$F = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

The Accuracy - This is the fraction of correct predictions, TP + TN, divided by the total amount of predictions, P + N. Defined as

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{TP + TN}{P + N}$$

The True Negative Rate, TNR - The fraction of the frames labeled closed, N, which was correctly classified, TN. Defined as

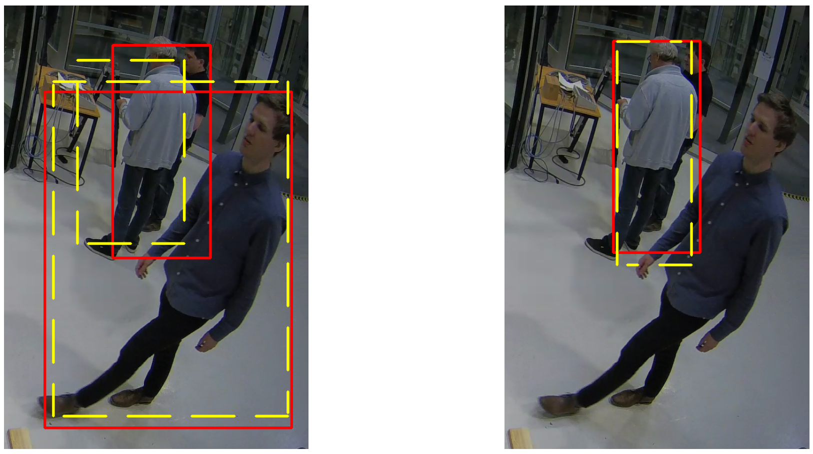
$$\text{TNR} = \frac{TN}{TN + FP} = \frac{TN}{N}$$

Chapter 5

Results

5.1 Pedestrian Detection

The detector that stands out the most is the YOLO detector trained on the MS COCO dataset. By inspecting the videos it is clear that there are very few situations for which the detector fails to find the pedestrians. In fig. 5.1a we can see the performance of both versions of the YOLO detector. The YOLO detector trained on Pascal VOC 2007 and 2012 datasets performed slightly worse and failed to detect the pedestrians in many frames. It specifically struggled with pedestrians located under the camera, see fig. 5.2. The Tiny-YOLO detector trained on Pascal VOC 2007 and 2012 datasets had problems with both detection and localization of objects i.e. there was a lot of variation in bounding box size which confused the trackers. The Tiny-YOLO version trained on the COCO dataset performed similarly the YOLO version trained on the VOC dataset.



(A) Detection performance of the YOLO detector trained on the MS COCO dataset in red and the VOC version in yellow (dashed). The performance between the two are similar with YOLO-COCO providing slightly more accurate bounding boxes.

(B) Detection performance of the ACF detector in red and DPM in yellow (dashed). The pedestrian in the front is not detected by either whereas the pedestrian in the back is detected by both.

FIGURE 5.1: A subimage with the detections from both YOLO detectors, the ACF and DPM based detectors.

The ACF and DPM based detectors had very similar performances but both where outperformed by the YOLO detectors (excluding the Tiny-YOLO detector trained on VOC). For upright pedestrians the detectors performed well but they struggled with pedestrians far from the image center with an angle in perceived pose, see fig. 5.1b. This issue was partly remedied by performing detections on different rotations of the image as described in sec. 4.4.1 and sec. 4.4.2. For pedestrians below the camera the detectors did not perform well, see fig. 5.3.

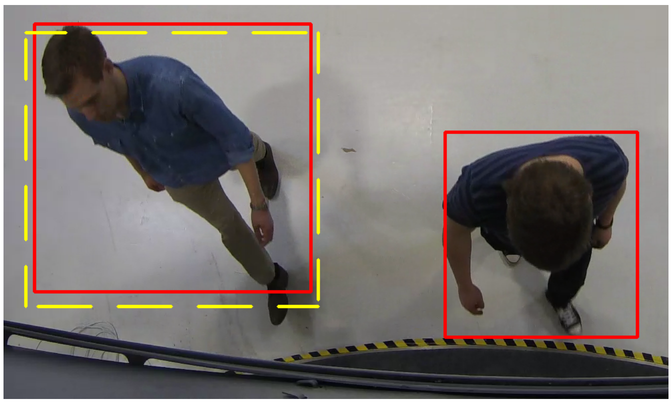


FIGURE 5.2: Detection performance of the YOLO detector trained on the COCO dataset in red and trained on the VOC dataset in yellow. YOLO-COCO finds both pedestrians but YOLO-VOC only finds the left pedestrian. The detected pedestrians are well localized for both detectors.

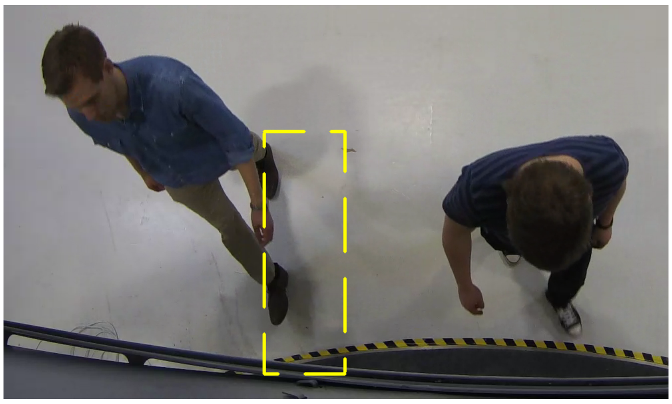


FIGURE 5.3: Detection performance of ACF in red and DPM in yellow (dashed). ACF fails to find both pedestrians. DPM detects the left pedestrian but the bounding box is not well localized.

The performance of the system using different input is presented in tbl. 5.1. The system uses the Hungarian-Kalman tracker and is evaluated on the test set.

The DPM and ACF based detectors run at approximately 10 fps when detecting in a single angle. YOLO runs at approximately 0.1 fps on a MacBook Pro 2012. However, the YOLO detector is optimized to run on GPU where 20 fps was attained on Amazon EC2 (Nvidia K80 12 GiB) instance. The Tiny-YOLO version attained an fps of 0.5 fps on a MacBook Pro 2012 and 70 fps on the Amazon EC2.

TABLE 5.1: Table with the performance of the system using the Hungarian-Kalman tracker with input from the different detectors. The prediction time is two seconds and the table is sorted by the F1 score.

Detector	Accuracy	TNR	Recall	Precision	F1↓
Ground Truth	**0.840**	**0.972**	**0.537**	**0.894**	**0.671**
YOLO (COCO)	0.832	0.964	0.530	0.865	0.657
YOLO (VOC)	0.816	0.963	0.478	0.850	0.612
Tiny-YOLO (COCO)	0.808	0.965	0.448	0.848	0.587
Tiny-YOLO (VOC)	0.792	0.979	0.361	0.883	0.513
DPM based	0.764	0.931	0.380	0.707	0.495
ACF based	0.764	0.951	0.335	0.748	0.463

5.2 Moving Object Detection

Clustering reduces the noise in the optical flow vectors and the mean vectors of a cluster gives a responsive output, see fig. 5.4. This was used as an input to the Hungarian-Kalman tracker and required a lot less filtering compared to other trackers in this work.

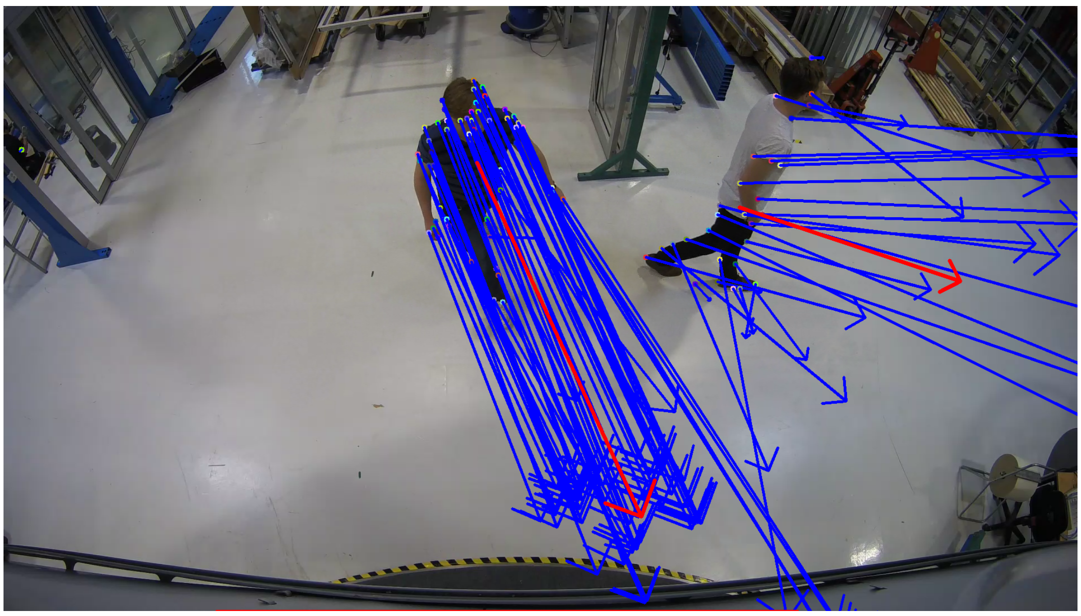


FIGURE 5.4: Results from motion detection. The blue vectors are optical flow vectors. The red vector is the result of clustering the vectors. Note that the pedestrian on the right does not have as many vectors assigned to him. This is because the mask from the background subtraction failed to find larger parts of his body.

This approach can only distinguish between objects if they are spatially separated or if they are heading in different directions. This was obvious from studies of videos where pedestrians are walking side by side. Quantitative results of using moving object detection are shown in tbl. 5.2.

TABLE 5.2: Table with the performance of the system using the Hungarian-Kalman tracker with input from the moving object detector. The prediction time is two seconds.

Detector	Accuracy	TNR	Recall	Precision	F1↓
Moving Object based	0.860	0.984	0.575	0.938	0.713
Ground Truth	**0.840**	**0.972**	**0.537**	**0.894**	**0.671**

5.3 Pedestrian Tracking

Visual examples of the results from applying tracking algorithms to the detection output is illustrated in fig. 5.5. The filters are sometimes slow to estimate the target's initial velocity as well as when a target changes direction. As mentioned in sec. 4.3.1 the measured velocity is in image coordinate space. This resulted in objects far from the image center getting inaccurate velocity estimates.

From manual inspection we could not determine if the increased complexity in the GM-PHD tracker as compared to the Hungarian-Kalman tracker improved tracking performance. More results are presented in sec. 5.4. The tracker with input from the moving object detector performs much better during turns and have better velocity estimations.

The graph based tracker discussed in sec. 3.3.5 was not used in the final tests. Implementation problems and the required causal adaptations reduced the tracker's performance to the point where it could no longer be considered.

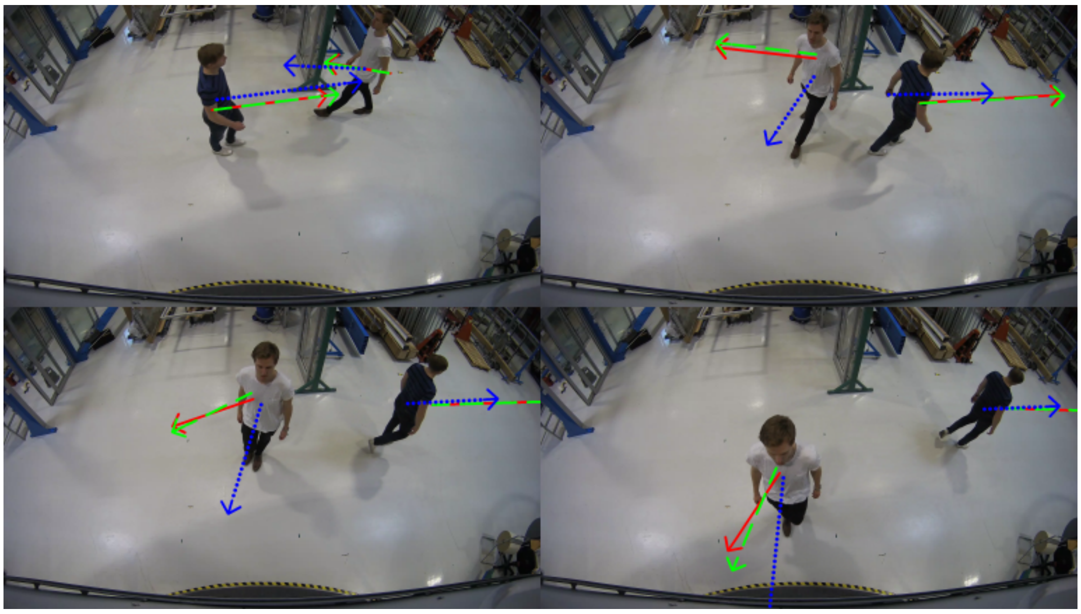


FIGURE 5.5: Output position and two second prediction from the trackers. GM-PHD in red and Hungarian-Kalman in green (dashed) using the ground truth detection data as inputs. The blue (dotted) line is the output from the Hungarian-Kalman tracker using the moving object detector as input. Note the fast response of the tracker associated with the blue (dotted) arrow.

5.4 System Performance

As expected the ground truth has the best performance but YOLO trained on MS COCO follows closely. Results of controlling the door using the different tracking algorithms can be found in tbl. 5.3. The YOLO detector trained on MS COCO was selected as a good detector for the evaluation. As described in sec. 4.8.2 all annotation is based around a two second opening time. Different prediction times were tested to account for the inaccurate velocity estimates. Results from using the ground truth detections are shown in appendix B.

The recall, precision and F1 score for the Hungarian-Kalman tracker using the YOLO-COCO detections is shown in fig. 5.6. In this figure we can see that the optimal prediction time is around eight seconds. The same tests for the moving object detector show six seconds to be the optimal prediction time. These prediction times are used to evaluate the different categories individually.

Manual inspection of the videos for both two and eight seconds prediction time show that the system is slow to open for pedestrians. The issue is larger when two seconds prediction time is used. There is also an issue of the system not keeping the door open when the pedestrians exits the frame through the door.

Performance of the Hungarian-Kalman tracker with input from the moving object detector is found in tbl. C.1. Note that the F1 score is relatively high and stable for different prediction times as compared to the results in tbl. 5.3.

TABLE 5.3: The performance of the different trackers with input from YOLO-COCO for complete test set with P.t denoting prediction time. The table is sorted by the F1 score.

Tracker	P.t (s)	Accuracy	TNR	Recall	Precision	F1↓
GM-PHD filter	8	0.890	0.959	0.732	0.886	0.802
Hungarian-Kalman	8	0.878	0.920	0.783	0.810	0.796
Hungarian-Kalman	6	0.884	0.942	0.749	0.850	0.796
GM-PHD filter	10	0.884	0.945	0.745	0.855	0.796
GM-PHD filter	6	0.888	0.971	0.699	0.913	0.792
Hungarian-Kalman	10	0.868	0.898	0.800	0.773	0.786
Hungarian-Kalman	4	0.876	0.958	0.687	0.877	0.771
GM-PHD filter	4	0.876	0.985	0.627	0.948	0.755
Hungarian-Kalman	2	0.832	0.964	0.530	0.865	0.657
GM-PHD filter	2	0.834	0.991	0.473	0.958	0.634

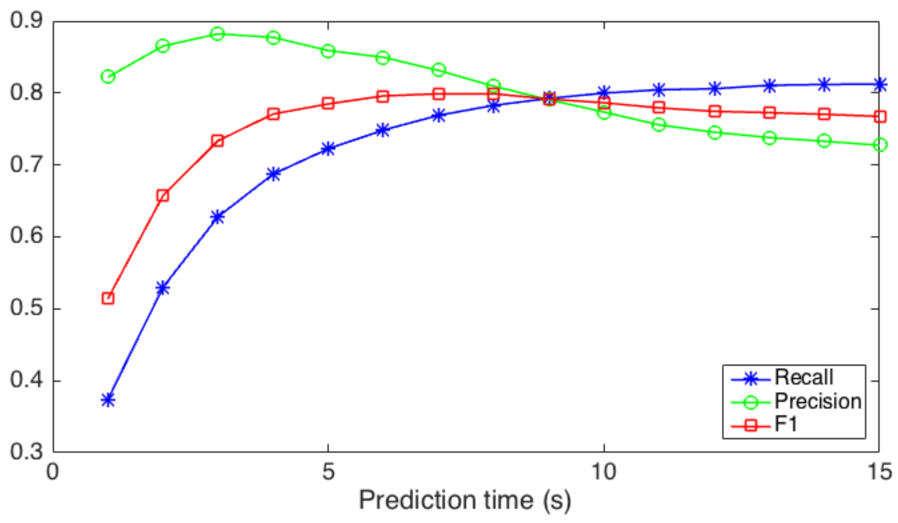


FIGURE 5.6: The recall, precision and F1 score plotted for different prediction times. The tracker used is the Hungarian-Kalman tracker and YOLO-COCO detections are used as input.

The performance of the trackers using detections from YOLO-COCO for different categories are presented in tbl. 5.4 to 5.7. The tables are sorted by the F1 score when applicable. As stated in sec 4.9, the F1 score is a good indicator of a balanced recall and precision. Results for the same categories using the moving object detector are shown in tbl. C.2 - C.5.

When observing the "straight movement towards the door" category we can see that the trackers using YOLO-COCO detections performed better than the tracker using the moving object detector, see tbl. 5.4 and C.3.

The score for the "sharp turn" category using the moving object detections is very impressive when comparing tbl. C.2 to tbl. 5.6.

In tbl. 5.7 we can see that since the TNR metric is very high both trackers can disregard cross-traffic efficiently.

TABLE 5.4: The performance of the different trackers with input from YOLO-COCO for the straight entry (SM) category with P.t denoting prediction time. The table is sorted by the F1 score.

Tracker	P.t (s)	TNR	Accuracy	Recall	Precision	F1↓
GM-PHD	8	0.900	0.931	0.844	0.870	0.857
Hungarian-Kalman	8	0.885	0.883	0.888	0.807	0.845
Hungarian-Kalman	2	0.844	0.973	0.609	0.926	0.735
GM-PHD	2	0.838	0.998	0.547	0.993	0.705

TABLE 5.5: The performance of the different trackers with input from YOLO-COCO for the straight movement from an angle (SA) category with P.t denoting prediction time. The table is sorted by the F1 score.

Tracker	P.t (s)	TNR	Accuracy	Recall	Precision	F1↓
Hungarian-Kalman	8	0.826	0.891	0.747	0.849	0.795
GM-PHD	8	0.836	0.962	0.683	0.936	0.790
Hungarian-Kalman	2	0.720	0.917	0.479	0.825	0.606
GM-PHD	2	0.728	0.980	0.420	0.946	0.582

TABLE 5.6: The performance of the different trackers with input from YOLO-COCO for the sharp turn (TU) category with P.t denoting prediction time. The table is sorted by the F1 score.

Tracker	P.t (s)	TNR	Accuracy	Recall	Precision	F1↓
GM-PHD	8	0.901	0.985	0.596	0.914	0.722
Hungarian-Kalman	8	0.884	0.958	0.617	0.800	0.697
Hungarian-Kalman	2	0.875	0.981	0.487	0.875	0.626
GM-PHD	2	0.878	0.993	0.455	0.950	0.616

TABLE 5.7: The performance of the different trackers with input from YOLO-COCO for the cross-traffic (CT) category with P.t denoting prediction time. Some metrics are not applicable and are labeled n/a.

Tracker	P.t (s)	TNR	Accuracy	Recall	Precision	F1
Hungarian-Kalman	8	0.965	0.965	n/a	0.000	n/a
GM-PHD	8	0.969	0.969	n/a	0.000	n/a
Hungarian-Kalman	2	0.989	0.989	n/a	0.000	n/a
GM-PHD	2	0.991	0.991	n/a	0.000	n/a

Chapter 6

Discussion

6.1 Detector Evaluation

The pedestrian detector that performed best when looking at the F1 score was the YOLO detector trained on the MS COCO dataset. From both qualitative and quantitative perspective it was a superior detector, see tbl. 5.1. It was hard to find a reasonable situation in which it did not find the pedestrians. Furthermore, the localization was even better than the ground truth developed in this work. However, this ground truth does have detections for every frame and the detections have very stable center points. This probably explains why the ground truth still has better accuracy, recall and F1 score than YOLO using a two second prediction.

Moving object detection performs best of all detectors, see tbl. 5.1 and tbl. 5.2. It has the highest accuracy, recall and F1 score. By inspecting the system it seems that the velocity estimation is more responsive than when using the pedestrian detectors. This is a very interesting result especially since the method was not initially considered but added during the project for its special characteristics. It is again worth to note that this method does not classify objects as pedestrians.

For the ACF and DPM based detectors a major factor in the low recall listed in tbl. 5.1 is that they do not detect pedestrians close to the camera. This is because both detectors are trained for detecting upright pedestrians and do not generalize well to the top view of a pedestrian. This is consistent with our results. If a larger dataset for top view pedestrians was to be created a new detector could be trained for this particular purpose. This is likely the best cause of action if either of these detectors are to be used in a production scenario. These should then also be trained on rotated images and datasets collected from the same viewing angle but many different environments. The advantage of these detectors is that they are quite fast and of fairly low complexity and we do not think our results are representative of their full potential.

6.2 Tracker Evaluation

The sometimes slow velocity estimation when pedestrians change direction, see fig. 5.5, is a result of the tuning of the trackers. There is a balance between the filters ability to handle noise and their responsiveness. By adding velocity as a measurable input we were able to tune the filters to be more responsive without increasing the impact of noise.

The inaccurate velocity estimates when the target is far from the camera center is a result of the image coordinates not corresponding well to the world coordinates. This comes as no surprise and was discussed in sec. 4.3.1. However, the impact of this was far greater than expected. We estimate that the predicted velocity in this work is only between 25-75% of the real velocity. These results are further discussed in sec. 6.3. The direction of the velocity estimates were stable for all trackers evaluated. Again, as a result of the tuning.

When inspecting the trackers using the YOLO-COCO detector we were not able to distinguish any crucial performance differences. However, there were differences when adding velocity as a measurement by using the moving object detector. This resulted in a significantly increased responsiveness with regards to direction changes. The speed estimation did not see the same improvement.

There was no indication that the more complex GM-PHD tracker performed better than the Hungarian-Kalman one with regards to tracking individual pedestrians. This could indicate that a less complex solution is adequate for the stated problem. However, it could also show that our tests did not fully capture the full complexity of the problem.

The graph based tracker did not reach the same performance as the other trackers. The initial tracklet generation was implemented successfully and tracked movement well on a public data set. When extending the algorithm to associate tracklets into tracks, the complexity of the implementation caused the tracker to associate detections incorrectly. We think that this is partly because of the causal adaptation, i.e. reducing the temporal window. The purpose of the initial algorithm and its structure, i.e. linking larger and larger tracklets, was lost. Also, the core of the original algorithm is to perform global data association, as discussed in sec. 3.3.5, in which handling occlusions and keeping track of pedestrians walking side-by-side is essential. These problems are less relevant in this work and the increased complexity adds little to its goal. We think that the failed implementation of this theory in this work does not conclude that it is not applicable for this type of problem.

6.3 System Evaluation

As discussed in sec. 5.4 the system activates the door too late and closes it too early. Both problems affect the recall score. The slow responsiveness to open the door is probably linked to the fact that the estimated speed is too low leading to the prediction being inaccurate. We can see, in tbl. 5.3, that doing a four, six, eight or ten second prediction increases performance significantly with regards to recall at the cost of reduced precision. In fig. 5.6 we can see that the optimal prediction time for the trackers using the YOLO-COCO input is around eight seconds. This seems to correspond quite well to the qualitative results which show the predicted velocity to be 25-75% smaller than the true velocity. We can also conclude that the direction estimation is quite good even though the speed estimation is very bad. This also shows that the system will benefit from a static increase in prediction time to compensate for the bad speed estimate.

The other part of the problem, closing to early, can be observed when stepping through each sequence manually for the different system setups. It was found that many frames were classified incorrectly towards the end of the video sequence. This is also verified in the results shown in tbl. 5.3 where a longer prediction time still results in a recall far from one. This is most likely related to the fact that the detector cannot properly detect pedestrians directly underneath the camera. Even when using the ground truth annotations with 8 seconds prediction time the system drops a few frames at the end, on the "straight movement" category, since the bounding box annotations and the open time annotations does not match entirely, see tbl. B.3.

The recall metric can be improved by a few actions; using a better annotated dataset, using a better over head detector or by keeping the door open for a fixed time after it has been opened. The last option is how most automatic door systems functions today.

In hindsight we believe that an additional scoring metric is to measure when the first open signal is sent and compare to the ground truth. This would allow us to observe a more specific behavior in the system.

6.3.1 Tracker Comparison

We can see in tbl. 5.3 that the Hungarian-Kalman and GM-PHD trackers have a similar performance using the detections from the YOLO-COCO detector. These results are taken from an average between all categories of video sequences. Since the performance is so similar it can be suspected that the difference stems from the tuning of the trackers.

When breaking down the different categories for the two trackers we cannot find any significant performance differences. The Hungarian-Kalman tracker usually has a higher recall while the GM-PHD has a higher precision. The resulting F1 score is very similar. This further shows that the added complexity of the GM-PHD tracker does not increase performance in a significant way.

Finally we can see that both trackers have good direction estimates since both handle cross-traffic exceptionally well, see tbl. 5.7. One video sequence is responsible for most of the misclassified frames. In this sequence a pedestrian walks extremely close to the door and camera distortions along with detector noise predicts the wrong state.

6.3.2 Velocity Estimates

As previously discussed the low recall in some tests was considered to be due to the inaccurate velocity estimates. The center of the bounding box does not move with constant velocity even when the pedestrian does in real world coordinates. We suggest that this is in large part due to the pixel-velocity estimates and one could therefore try to model movement along the ground plane. The feet of the pedestrians could be located and from there one could extract the real world position. This could be possible since the feet should be visible as long as the pedestrians are not standing under the camera. The camera distortions for our wide-angled camera is however quite severe in the edges and modeling a plane for this camera setup will be challenging. The center of the bounding box does not always correspond to the same part of the body as the view of the pedestrian changes. This most likely also affects the velocity estimates. It is possible that using multiple cameras with less distortion would reduce this problem.

We can see in tbl. 5.5 and B.4 that the recall in the "straight movement from an angle" case is quite low. After investigating this we found that the recall is reduced by the score from the high velocity video sequences in this category. This shows yet again that the camera can observe little movement in the sides of the image which leads to velocity estimates which do not represent a pedestrian's running movement. It is also a hard task to open the door two seconds before the pedestrian enters since the pedestrian is only partly visible when the door is supposed to be opened. The error is amplified by the high velocity a running pedestrian can achieve, which could be remedied by using a faster door.

Aside from the presented problems with camera distortions and coordinate planes there is a problem when a pedestrian changes velocity. This means that the trackers are slow to react to turns and changes in speed. They also have difficulties with finding the correct speed of newly tracked pedestrians. This is partly remedied with the added velocity information from the moving object detector. We can see this very clearly when comparing the "sharp turn" category in tbl. 5.6 and tbl. C.2.

6.4 Conclusions

Our results indicate that the automatic door solutions used today can benefit from the usage of computer vision. A major advantage of our system compared to current systems is the ability to handle cross-traffic in a robust manner. The processing costs for all pedestrian detectors is very high, especially for the YOLO detector which outperforms both the ACF and DPM based detectors. In order to run the algorithm in real time, a high-end GPU is required. Considering that at least one camera on each side of the door is required the processing cost will be further increased. This is a large disadvantage of the system implemented in this work. The results in this work does not conclude that the ACF and DPM based detectors cannot be used for this application. However, they would both need to be trained specifically for this purpose. Moving object detection is a faster method than pedestrian classification detection methods. It does not provide any classification but can possibly be used together with other methods. All evaluated detection methods have higher processing costs than sensors used today. By evaluation we found the less complex Hungarian-Kalman tracker to be the most prominent for this application due to the ease of tuning. We do believe that with the usage of this system, one can decrease the amount of false openings and increase the comfort for its users. The increased comfort comes from the system having the ability to open a door earlier when a pedestrian approaches. This affects pedestrians interactions with the door as the door then adapts to the pedestrian, as opposed to the pedestrian slowing down to wait for the door.

Chapter 7

Future Work

Distortions in the camera used affected our velocity estimates considerably. It would be interesting to design a system with multiple cameras with smaller viewing angle to avoid these distortions. Mapping the real ground plane to the observed image, even with distortions, could provide better velocity estimations.

The relative low computation cost of the ACF and DPM based detectors compared to the YOLO detectors means that multiple instances of the detectors can be operated at the same time. A possible system would use ACF or DPM based detectors trained for pedestrians in a certain viewing angle. This would allow the system to be computationally fast with a high detection accuracy.

Under the assumption that a pedestrian reacts to the position of the door and how fast it is moving one can model the interaction as a system. This system would use the approaching pedestrian as a measurement and the door position as a control signal. This would mean that one can use control theory to improve performance of the system by e.g. reducing the width the door needs to open.

Another interesting approach is to train the different tuning parameters from data gathered by the system itself. If the complete system would be able to determine if a pedestrian has passed or not it can evaluate its own behavior. By this we mean that we can find e.g. the optimal prediction time and the optimal time to keep a door open after an open signal is received. One could also train the system to react differently depending on the surrounding of the door e.g. typical pedestrian flow depending on time of day. Another benefit is that the system can adapt to the camera distortion.

Appendix A

Histogram of Oriented Gradients (HOG)

When using different types of computer vision algorithms a corner stone is the type of features one can extract from the image in question. A feature is a piece of information about an image or data sequence which is used in a computation. The detection of humans often require a lot of different features since humans come in many sizes, shapes, wearing different clothes and are seen in different lighting. A popular set of features is the Histogram of Oriented Gradients which, as shown by Dalal and Triggs [36], can be gainfully used in pedestrian detection [1] [3]. One reason for why HOG is a good feature extractor is because it uses edge information i.e. areas with high frequency content. This is interesting since it is often edges in a image which provides the necessary information required to understand what one is looking at.

The first step in calculating the HOG is finding the pointwise derivatives in both horizontal and vertical directions. Different variations such as the usage of different derivative approximations, e.g. Sobel mask, or smoothing the image can also be done in this step. The image is then divided into cells where each pixel within the cells will add a function of the amplitude of its gradient into a histogram channel. This yields cells with a histogram of the amount of pixels which point in the different directions.

The cells then need to be normalized to account for differences throughout the image. This is done by creating overlapping blocks of cells and normalizing over them by the use of e.g. the L^1- or L^2-norm. The final descriptor is the result after this normalization, see fig. A.1 for an example image.

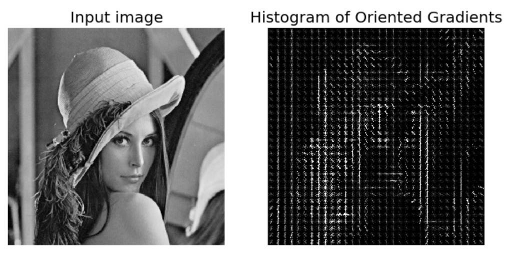


FIGURE A.1: An example image of a HOG descriptor with input image [43].

Appendix B

Results for Ground Truth Detections

TABLE B.1: The performance of the different trackers with input from the ground truth for the complete test set with P.t denoting prediction time. The table is sorted by the F1 score.

Tracker	P.t (s)	Accuracy	TNR	Recall	Precision	F1↓
Hungarian-Kalman	8	0.889	0.933	0.787	0.836	0.811
Hungarian-Kalman	6	0.890	0.951	0.752	0.869	0.806
Hungarian-Kalman	10	0.883	0.920	0.799	0.813	0.806
GM-PHD filter	8	0.885	0.951	0.733	0.867	0.794
GM-PHD filter	10	0.882	0.941	0.746	0.846	0.793
GM-PHD filter	6	0.884	0.962	0.705	0.891	0.787
Hungarian-Kalman	4	0.883	0.965	0.694	0.897	0.782
GM-PHD filter	4	0.873	0.973	0.643	0.911	0.754
Hungarian-Kalman	2	0.840	0.972	0.537	0.894	0.671
GM-PHD filter	2	0.832	0.976	0.500	0.899	0.643

TABLE B.2: The performance of the different trackers with input from the ground truth for the Sharp turn (TU) category with P.t denoting prediction time. The table is sorted by the F1 score.

Tracker	P.t (s)	TNR	Accuracy	Recall	Precision	F1↓
Hungarian-Kalman	8	0.900	0.976	0.623	0.875	0.728
GM-PHD	8	0.892	0.976	0.584	0.869	0.699
Hungarian-Kalman	2	0.874	0.983	0.478	0.885	0.621
GM-PHD	2	0.869	0.980	0.464	0.862	0.604

TABLE B.3: The performance of the different trackers with input from the ground truth for the Straight entry (SM) category with P.t denoting prediction time. The table is sorted by the F1 score.

Tracker	P.t (s)	TNR	Accuracy	Recall	Precision	F1↓
Hungarian-Kalman	8	0.892	0.865	0.942	0.793	0.861
GM-PHD	8	0.894	0.914	0.858	0.845	0.852
Hungarian-Kalman	2	0.861	0.972	0.659	0.929	0.771
GM-PHD	2	0.845	0.981	0.597	0.946	0.732

TABLE B.4: The performance of the different trackers with input from the ground truth for the Straight movement from an angle (SA) category with P.t denoting prediction time. The table is sorted by the F1 score.

Tracker	P.t (s)	TNR	Accuracy	Recall	Precision	F1↓
Hungarian-Kalman	8	0.836	0.938	0.711	0.904	0.796
GM-PHD	8	0.826	0.948	0.677	0.915	0.778
Hungarian-Kalman	2	0.728	0.951	0.456	0.884	0.601
GM-PHD	2	0.722	0.958	0.434	0.895	0.585

TABLE B.5: The performance of the different trackers with input from the ground truth for the Cross-traffic (CT) category with P.t denoting prediction time. Some metrics are not applicable and are labeled n/a.

Tracker	P.t (s)	TNR	Accuracy	Recall	Precision	F1
Hungarian-Kalman	8	0.976	0.976	n/a	0.000	n/a
GM-PHD	8	0.980	0.980	n/a	0.000	n/a
Hungarian-Kalman	2	0.986	0.986	n/a	0.000	n/a
GM-PHD	2	0.984	0.984	n/a	0.000	n/a

Appendix C

Results for Moving Object Detection

TABLE C.1: The performance of the different trackers with input from the moving object detections for the complete test set with P.t denoting prediction time. The table is sorted by the F1 score.

Tracker	P.t (s)	Accuracy	TNR	Recall	Precision	F1↓
Hungarian-Kalman w. vel.	6	0.894	0.936	0.795	0.845	0.819
Hungarian-Kalman w. vel.	4	0.896	0.967	0.735	0.905	0.811
Hungarian-Kalman w. vel.	8	0.880	0.909	0.814	0.795	0.804
Hungarian-Kalman w. vel.	10	0.866	0.886	0.821	0.759	0.788
Hungarian-Kalman w. vel.	2	0.860	0.984	0.575	0.938	0.713

TABLE C.2: The performance of the different trackers with input from the moving object detections for the Sharp turn (TU) category with P.t denoting prediction time. The table is sorted by the F1 score.

Tracker	P.t (s)	TNR	Accuracy	Recall	Precision	F1↓
Hungarian-Kalman w. vel.	6	0.942	0.976	0.820	0.902	0.859
Hungarian-Kalman w. vel.	2	0.905	0.986	0.610	0.924	0.735

TABLE C.3: The performance of the different trackers with input from the moving object detections for the Straight entry (SM) category with P.t denoting prediction time. The table is sorted by the F1 score.

Tracker	P.t (s)	TNR	Accuracy	Recall	Precision	F1↓
Hungarian-Kalman w. vel.	6	0.870	0.890	0.833	0.807	0.820
Hungarian-Kalman w. vel.	2	0.856	0.983	0.624	0.952	0.754

TABLE C.4: The performance of the different trackers with input from the moving object detections for the Straight movement from an angle (SA) category with P.t denoting prediction time. The table is sorted by the F1 score.

Tracker	P.t (s)	TNR	Accuracy	Recall	Precision	F1↓
Hungarian-Kalman w. vel.	6	0.856	0.937	0.758	0.907	0.826
Hungarian-Kalman w. vel.	2	0.773	0.976	0.525	0.948	0.676

TABLE C.5: The performance of the different trackers with input from the moving object detections for the Cross-traffic (CT) category with P.t denoting prediction time. Some metrics are not applicable and are labeled n/a.

Tracker	P.t (s)	TNR	Accuracy	Recall	Precision	F1
Hungarian-Kalman w. vel.	6	0.960	0.960	n/a	0.000	n/a
Hungarian-Kalman w. vel.	2	0.990	0.990	n/a	0.000	n/a

Bibliography

[1] P. Dollár et al. “Pedestrian Detection: A Benchmark”. In: *CVPR*. 2009.

[2] X. Du et al. “Fused DNN: A Deep Neural Network Fusion Approach to Fast and Robust Pedestrian Detection”. In: *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. 2017, pp. 953–961. DOI: 10.1109/WACV.2017.111.

[3] P. Felzenszwalb, D. McAllester, and D. Ramanan. “A discriminatively trained, multiscale, deformable part model”. In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. 2008, pp. 1–8. DOI: 10.1109/CVPR.2008.4587597.

[4] P. Dollár et al. “Fast Feature Pyramids for Object Detection”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.8 (2014), pp. 1532–1545. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2014.2300479.

[5] Yoav Freund and Robert E. Schapire. *A Decision-Theoretic Generalization of on-Line Learning and an Application to Boosting*. 1997.

[6] *Scalepyramid*. [Online; accessed June 5, 2017]. 2016. URL: http://iipimage.sourceforge.net/documentation/images/.

[7] Chun-Nam John Yu and Thorsten Joachims. “Learning Structural SVMs with Latent Variables”. In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML ’09. Montreal, Quebec, Canada: ACM, 2009, pp. 1169–1176. ISBN: 978-1-60558-516-1. DOI: 10.1145/1553374.1553523. URL: http://doi.acm.org/10.1145/1553374.1553523.

[8] Pedro Felzenszwalb. *dpmpic*. [Online; accessed June 6, 2017]. 2016. URL: http://deliveryimages.acm.org/10.1145/2500000/2494532/figs/f2.jpg.

[9] Andrej Karpathy. *3-layer neural network*. [Online; accessed May 24, 2017]. 2017. URL: http://cs231n.github.io/assets/nn1/neural_net2.jpeg.

[10] Andrej Karpathy. *Artificial neuron*. [Online; accessed May 24, 2017]. 2017. URL: http://cs231n.github.io/assets/nn1/neuron_model.jpeg.

[11] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[12] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, 2015, pp. 448–456. URL: http://proceedings.mlr.press/v37/ioffe15.html.

[13] Ignacio Icke. *Overfitting*. [Online; accessed May 24, 2017]. 2008. URL: https://en.wikipedia.org/wiki/Overfitting#/media/File:Overfitting.svg.

[14] K. He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[15] C. Szegedy et al. "Going deeper with convolutions". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 1–9. DOI: 10.1109/CVPR.2015.7298594.

[16] R. Girshick. "Fast R-CNN". In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1440–1448. DOI: 10.1109/ICCV.2015.169.

[17] S. Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.6 (2017), pp. 1137–1149. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2016.2577031.

[18] J. Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91.

[19] Joseph Redmon and Ali Farhadi. "YOLO9000: Better, Faster, Stronger". In: *CoRR* abs/1612.08242 (2016). URL: http://arxiv.org/abs/1612.08242.

[20] OpenCV. *Background subtraction using MOG2*. [Online; accessed May 24, 2017]. 2017. URL: http://docs.opencv.org/trunk/resmog2.jpg.

[21] Z. Zivkovic. "Improved adaptive Gaussian mixture model for background subtraction". In: *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004*. Vol. 2. 2004, 28–31 Vol.2. DOI: 10.1109/ICPR.2004.1333992.

[22] Bruce D. Lucas and Takeo Kanade. "An Iterative Image Registration Technique with an Application to Stereo Vision". In: *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*. IJCAI'81. Vancouver, BC, Canada: Morgan Kaufmann Publishers Inc., 1981, pp. 674–679. URL: http://dl.acm.org/citation.cfm?id=1623264.1623280.

[23] Ba Tuong Vo. *Random finite sets in multi-object filtering*. Citeseer, 2008.

[24] Simo Srkk. *Bayesian Filtering and Smoothing*. New York, NY, USA: Cambridge University Press, 2013. ISBN: 1107619289, 9781107619289.

[25] H. W. Kuhn and Bryn Yaw. "The Hungarian method for the assignment problem". In: *Naval Res. Logist. Quart* (1955), pp. 83–97.

[26] Jack Edmonds and Richard M. Karp. "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems". In: *J. ACM* 19.2 (Apr. 1972), pp. 248–264. ISSN: 0004-5411. DOI: 10.1145/321694.321699. URL: http://doi.acm.org/10.1145/321694.321699.

[27] Li Zhang, Yuan Li, and R. Nevatia. "Global data association for multi-object tracking using network flows". In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. 2008, pp. 1–8. DOI: 10.1109/CVPR.2008.4587584.

[28] A. Dehghan, S. M. Assari, and M. Shah. "GMMCP tracker: Globally optimal Generalized Maximum Multi Clique problem for multiple object tracking". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 4091–4099. DOI: 10.1109/CVPR.2015.7299036.

[29] Ergys Ristani and Carlo Tomasi. “Tracking Multiple People Online and in Real Time”. In: *Asian Conference on Computer Vision*. Springer. 2014, pp. 444–459.

[30] B. N. Vo and W. K. Ma. “The Gaussian Mixture Probability Hypothesis Density Filter”. In: *IEEE Transactions on Signal Processing* 54.11 (2006), pp. 4091–4104. ISSN: 1053-587X. DOI: 10.1109/TSP.2006.881190.

[31] Ronald Mahler. “Random set theory for target tracking and identification”. In: *Multisensor Data Fusion*. CRC press, 2001.

[32] R. P. S. Mahler. “Multitarget Bayes filtering via first-order multitarget moments”. In: *IEEE Transactions on Aerospace and Electronic Systems* 39.4 (2003), pp. 1152–1178. ISSN: 0018-9251. DOI: 10.1109/TAES.2003.1261119.

[33] David M. Beazley. “SWIG: An Easy to Use Tool for Integrating Scripting Languages with C and C++”. In: *Proceedings of the 4th Conference on USENIX Tcl/Tk Workshop, 1996 - Volume 4*. TCLTK’96. Monterey, California: USENIX Association, 1996, pp. 15–15. URL: http://dl.acm.org/citation.cfm?id=1267498.1267513.

[34] G. Bradski. “OpenCv”. In: *Dr. Dobb’s Journal of Software Tools* (2000).

[35] Inc. Gurobi Optimization. *Gurobi Optimizer Reference Manual*. 2016. URL: http://www.gurobi.com.

[36] N. Dalal and B. Triggs. “Histograms of oriented gradients for human detection”. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*. Vol. 1. 2005, 886–893 vol. 1. DOI: 10.1109/CVPR.2005.177.

[37] Tsung-Yi Lin et al. “Microsoft COCO: Common Objects in Context”. In: *CoRR* abs/1405.0312 (2014). URL: http://arxiv.org/abs/1405.0312.

[38] M. Everingham et al. *The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results*. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[39] M. Everingham et al. *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results*. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[40] Martin Ester et al. “A density-based algorithm for discovering clusters in large spatial databases with noise”. In: AAAI Press, 1996, pp. 226–231.

[41] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[42] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. “Exploring network structure, dynamics, and function using NetworkX”. In: *Proceedings of the 7th Python in Science Conference (SciPy2008)*. Pasadena, CA USA, Aug. 2008, pp. 11–15.

[43] Stefan van der Walt. *Hogpicture*. [Online; accessed May 29, 2017]. 2016. URL: http://sharky93.github.io/docs/gallery/auto_examples/plot_hog.html.

Master's Theses in Mathematical Sciences 2017:E33
ISSN 1404-6342

LUTFMA-3324-2017

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/