UNIVERSITY OF ALCALA
Polytechnic School

INTERNATIONAL EXCHANGE AGREEMENT
with
MÄLARDALEN UNIVERSITY
Department of Computer Science and Electronics

Master Thesis

TRACKING MULTIPLE OBJECTS WITH KALMAN FILTERS
PART I

Johanna Broddfelt

February 2006

UNIVERSITY OF ALCALA
Polytechnic School

INTERNATIONAL EXCHANGE AGREEMENT
with
MÄLARDALEN UNIVERSITY
Department of Computer Science and Electronics

Master Thesis

TRACKING MULTIPLE OBJECTS WITH KALMAN FILTERS PART I

Author: JOHANNA BRODDFELT
Alcalá Supervisor: MARTA MARRÓN ROMERA
Home Supervisor: MIKAEL EKSTRÖM

International Program Responsible: ANTONIO GUERRERO BAQUERO

Examiners:

President: Elena López Guillén

Examiner 2: Juan Carlos García García

Examiner 3: Marta Marrón Romera

MARK: .....

DATE: .....

TABLE OF CONTENTS

I. ABSTRACT

II. REPORT

III. USER MANUAL

IV. SPECIFICATIONS

V. PROGRAM CODE

VI. BUDGET

VII. REFERENCES

LIST OF FIGURES:

LIST OF TABLES:

I. ABSTRACT

There are many different solutions for tracking multiple objects and many of these solutions involve probabilistic algorithms, which have been fully tested as the best solution in tracking tasks. In this thesis a multiple tracking algorithm based on the Kalman Filter and the Probabilistic Data Association Filter is developed. The algorithm is part of an obstacle avoidance system in an autonomous robot. The measurements used as input to the tracking algorithm come from a stereo-vision system that detects objects in the robot's environment. The robustness and adaptability of the tracking algorithm is increased by the use of a validation/removal algorithm. The algorithm is capable of initiating tracks, accounting for false reports, and removing tracks, accounting for missing reports.

1. Introduction

This master thesis is focused in the area of robotics and probabilistic algorithms. The objective of the work described here is to track multiple objects in an image with different Kalman Filters (KF). One KF is used for tracking each object.

There are two parts of the project. One of them focuses on the development of a simulator platform (Part I) and the other one on a real time platform [17] (Part II). In real time application, the implementation focuses on achieving a small execution time. In this part, where the implementation is done on a simulator platform, the execution time is not as important.

The correct estimation of the position of multiple objects in a scene is important in many different applications such as mobile robots, white radar or identification of people in a scene. This thesis is centered on the case of mobile robots. In this case it is necessary for the robot to detect and determine the position and velocity, both modulus and direction, of the objects in the scene so that it can choose the right path and avoid collisions.

The input comes from a data file that contains the xyz-coordinates of image points that originate from different objects in the scene. The number of objects can change with time. The input 3D points are obtained with a stereo-vision system, which has already been implemented in another project [12].

The association of the points to different objects is done with another probabilistic algorithm, the Probabilistic Data Association Filter (PDAF). A difficult part of the association is that the number of measures obtained from an object may vary from one to many. In many previous works it is assumed that one object can only generate one measurement [11]. The output of the developed tracking algorithm is the position, the velocity and the number of objects in each image.

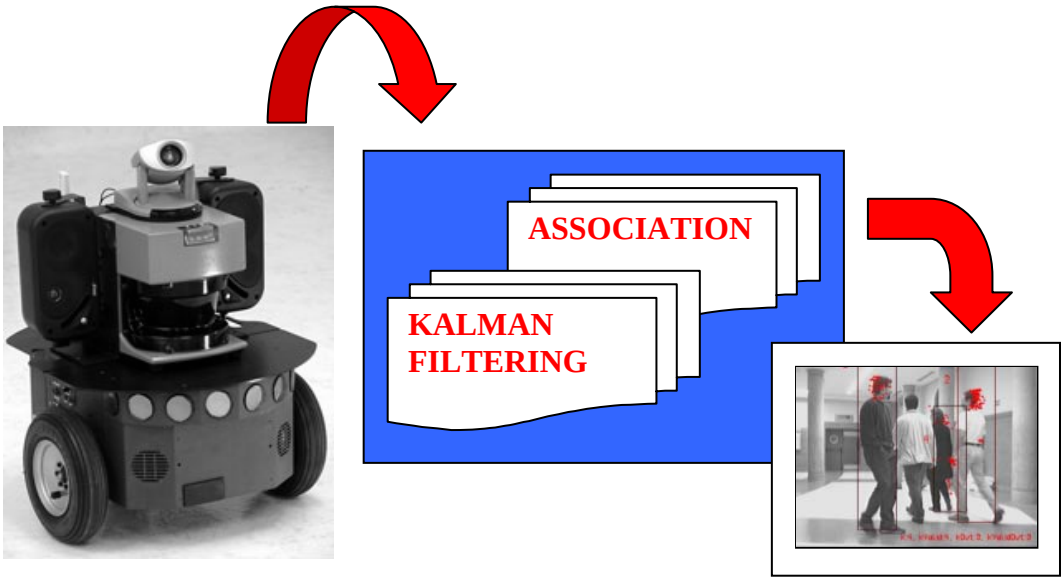

Figure 1.1: General description of the project.

1.1 Outline

To make the understanding of the realised work easier, the report contains seven chapters which can be subdivided into four groups:

1. **Objectives:** The objectives of this master thesis are described in Chapter 2. The difficulty is not only to track the objects, but also to associate the data to different objects. This includes a number of problems such as validation, occlusion and crossing.

2. **Theoretical basis of the development:** Chapter 3 gives a theoretical description of the tracking algorithms used; the Kalman filter and the association algorithm. Chapter 4 explains the input data, the output and the image transformation necessary to visualize the results. This chapter also describes the system model used by the Kalman filter in this thesis.

3. **Implementation:** How the theoretical information is applied in this thesis is described Chapter 5. The implantation was done in three steps, first with one single object, then with a fixed number of objects and finally with a variable number of objects. Solutions on how to solve the problems with association described in the objectives are also presented.

4. **Results:** In Chapter 6 the results of tracking different number of objects are presented. The errors and the accuracy of the tracking algorithm are investigated and then the influences of different parameters are tested. Finally Chapter 7 ends the report with a conclusion.

2. Objectives

The tracking algorithm presented in this thesis deals with the problems of associating measurements and to determine the number of objects and their estimated positions. To do this some objectives must be stated, and these are presented in this section.

- **Tracking objects:** There are many algorithms to choose from when tracking objects. However, given the knowledge that the input data contains noise [12], limits the choice to probabilistic algorithms. In this work the probabilistic Kalman filter is used, since it provides the optimal implementation of the Bayes' filter [1]. One KF is used to track each object. It is possible to use an only estimator for all the objects, but since the number of objects is variable this is more difficult to implement.

- **Data association:** For the tracking to be possible it is necessary to identify which measurements that comes from which object. To develop this association there are multiple choices of algorithms, as discussed in Chapter 3.2. The association algorithm used in this work is the Probabilistic Data Association Filter, PDAF. This algorithm uses the Euclidean distance, a validation gate and a measure probability to do the association.

- **Varying numbers of objects:** The algorithm must be able to handle the fact that the number of objects in the scene can vary. This fact brings problems that need to be taken care of. For example some phenomena might affect a tracker's estimate or decimate the input data. Noise might temporarily create multiple measurements or cause the target-originated measurement to disappear. If a visual disturbance is too severe, the tracker can lose target altogether.

The objective is to solve these problems by analyzing some causes of disturbances and come up with a solution on how to deal with them.

- **Validation:** There may be unexpected measurements - do these measurements originate from newly visible objects or are they just clutter? To avoid that noise is characterized as an object, data has to be associated to the object a certain number of times before it is validated as a real object in the scene.

- **Occlusion:** Predictions may not be supported by measurements – have these objects ceased to exist or are they simply occluded? Occlusion results when another scene element is interposed between the camera and the tracked object, blocking the object's image projection, or a portion of it. This result in incomplete data or no data associated with the tracked object. To avoid that an object is removed even though it is still in the scene, the object is kept in the scene for a certain amount of time even though no new data is associated to it. This is the opposite process to the validation process explained above.

- **Crossing:** A single measurement may match to more than one object – which object should the measurement be assigned to? When two tracked objects cross each other's paths one target–originated measurement may often fall within the other target's overlapping tracking window. This could lead to multiple trackers locked onto the same part. This problem is solved by the algorithm itself since an input parameter to the Kalman filter is the velocity. Another problem appears when two objects move close to each other in the same direction and with the same velocity. To avoid that two objects is seen as one, each object has a validation gate. The size of this gate determines how good the algorithm is on separating measures of different objects.

3. Tracking Algorithms

To be able to carry out the tracking of an object, some kind of information, usually from sensors, has to be received. The implemented system is as mentioned part of an obstacle avoidance system in a robot's vision system. The robot must be able to move autonomously in partially structured environments, and uses a stereo-vision system to recognize objects in its way. The detection of the different objects in the scene has already been done and put into a data file, see previous works [12].

Since the data is noisy and the measures consist of both information and clutter, it is difficult to get an exact position of the objects from the input data only. These inaccuracies, together with other errors, usually have statistical properties that are different from the target ones. This makes it possible to extract the target tracks from the clutter using different probabilistic models. In this chapter the methods and algorithms for achieving this are described and an overview of the tracking task is shown in Figure 3.1.

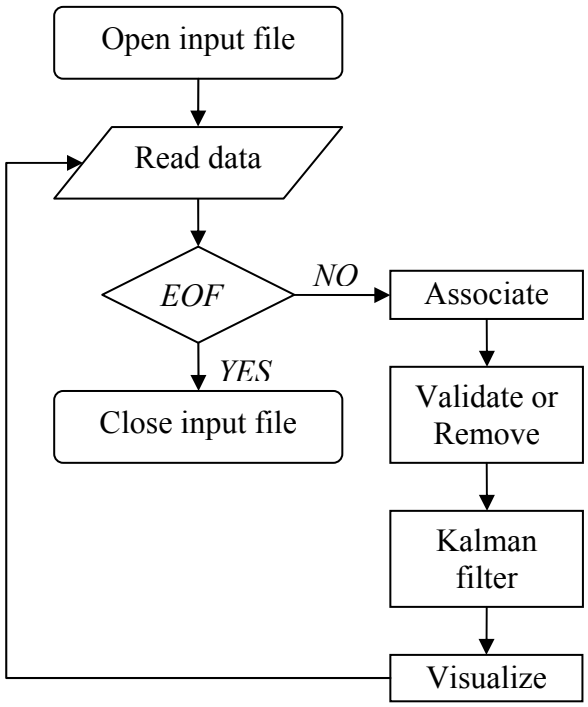

Figure 3.1: Brief overview.

In the association process each measurement is assigned to a target. At each time step, or frame, it has to be decided if the measurements arise from a previously known object, from a new object, or if the measurement is false. This can be done by applying different association algorithms such as the Maximum Likelihood filter (ML), the Nearest Neighbor filter (NN) or the Probabilistic Data Association Filter (PDAF). The last one, the PDAF, will be used in this thesis. Different kinds of association algorithms will be commented later on in this chapter.

As each measure has been associated the states have to be estimated. There are many solutions on how to do this, and a lot of research has been done the last years. Many of these solutions involve some kind of filter, such as the Particle Filter (PF) or the Kalman Filter (KF) which will be used in this thesis. Kalman filtering is an efficient method for tracking when the distribution of measurements is Gaussian. The KF is a discrete process and it is an optimal solution of the Bayes filter when the system is linear and the related noise is Gaussian with zero mean.

3.1 The Kalman Filter

The discrete Kalman filter is a set of mathematical equations that provides an efficient recursive mean for estimation of the state of a discrete process, in a way that minimizes the mean of the squared error, see [1]. The structure of the Kalman filter is very simple as it can be seen in Figure 3.2 below:

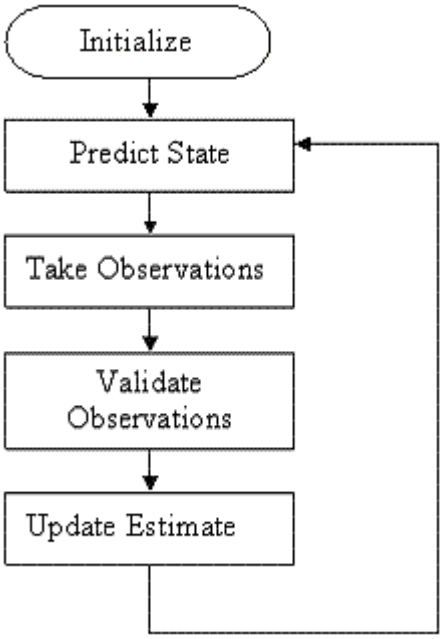

Figure 3.2: Structure of the Kalman filter.

3.1.1 Equations

The Kalman filter combines a motion model with measurements in order to obtain the best target estimate. To use the KF, the process must be expressed by a linear set of equations. Equation 3.1 is called the state or process equation (motion model) and it expresses the evolution of the process' state vector with time:

$$\vec{a}_k = G\vec{a}_{k-1} + H\vec{u}_{k-1} + \vec{w}_{k-1} \qquad <3.1>$$

where G is a $n \times n$ transition matrix that relates the state at the previous time step to the state at the current time step, and n is the number of states. H is a $n \times l$ matrix that relates the state vector to the control input matrix, $\vec{u} \in \Re^l$. $\vec{w}_k$ represents the process noise, that is a zero-mean white Gaussian noise sequence, $p(\vec{w}) \sim N(0, Q)$, whose covariance matrix is *Q(k)*.

The measurement is modeled by the output equation (sensor model) and exposes which is going to be the output of the system from the current state vector:

$$\vec{m}_k = C\vec{a}_k + \vec{o}_k \qquad <3.2>$$

where $\vec{m}_k$ is the measurement vector and C is a $m \times n$ matrix that relates the state vector to the measurement. $\vec{o}_k$ represents the measurement noise that is a zero-mean white Gaussian noise sequence, $p(\vec{o}) \sim N(0, R)$ whose covariance matrix is *R(k)*.

Equations 3.1 and 3.2 can be presented as in the block model below:

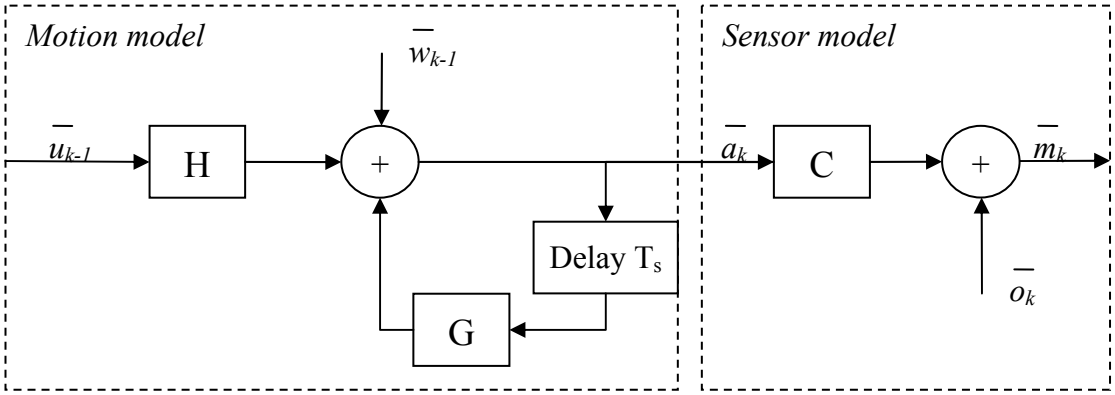

Figure 3.3: Signal-flow-model of the discrete time, linear, dynamical system.

3.1.2 Prediction and Correction

The filter performs the state vector estimation in two phases: **prediction** and **correction** [1]. In the prediction phase the algorithm predicts the value of the state vector using the system model and Bayes' rule, zeroing the process noise $\vec{w}$. Then, in the correction phase, it adjusts the prediction with an actual measurement at that time. Figure 3.4 gives a complete picture of the operation and equations of the KF, the vector sign $(\rightarrow)$ will be left out from now on.

The errors for a *priori* and a *posteriori* estimation can be defined as shown by Equations 3.2 and 3.3 respectively. In the equations $\hat{a}_k^- \in \Re^n$ is defined to be a *priori* state estimate at step *k* given knowledge of the process prior to step *k, k-1,* and $\hat{a}_k \in \Re^n$ is defined to be a *posteriori* state estimate at step *k* given the measurement m_k. The $\wedge$ sign indicates that it is an estimate and the minus sign indicates that it is a *priori* estimation.

$$e_k^- \equiv a_k - \hat{a}_k^- \qquad <3.2>$$

$$e_k \equiv a_k - \hat{a}_k \qquad <3.3>$$

The a *priori* estimate error covariance is expressed by Equation 3.4 and the a *posteriori* estimate error covariance by Equation 3.5, where T denotes the matrix transpose.

$$P_k^- = E\left[e_k^- e_k^{-T}\right] \qquad <3.4>$$

$$P_k = E\left[e_k e_k^T\right] \qquad <3.5>$$

These equations are achieved through the Bayes' filter theory. They are useful when deriving the expressions for the KF. The goal is to find a formula that computes a *posteriori* state estimate, $\hat{a}_k$, as a linear combination of a priori estimate $\hat{a}_k^-$ and a weighted difference between an actual measurement m_k and a prediction of the measurement vector $C\hat{a}_k^-$. This equation (3.10) together with the rest of the prediction and correction equations are shown in Figure 3.4 below.

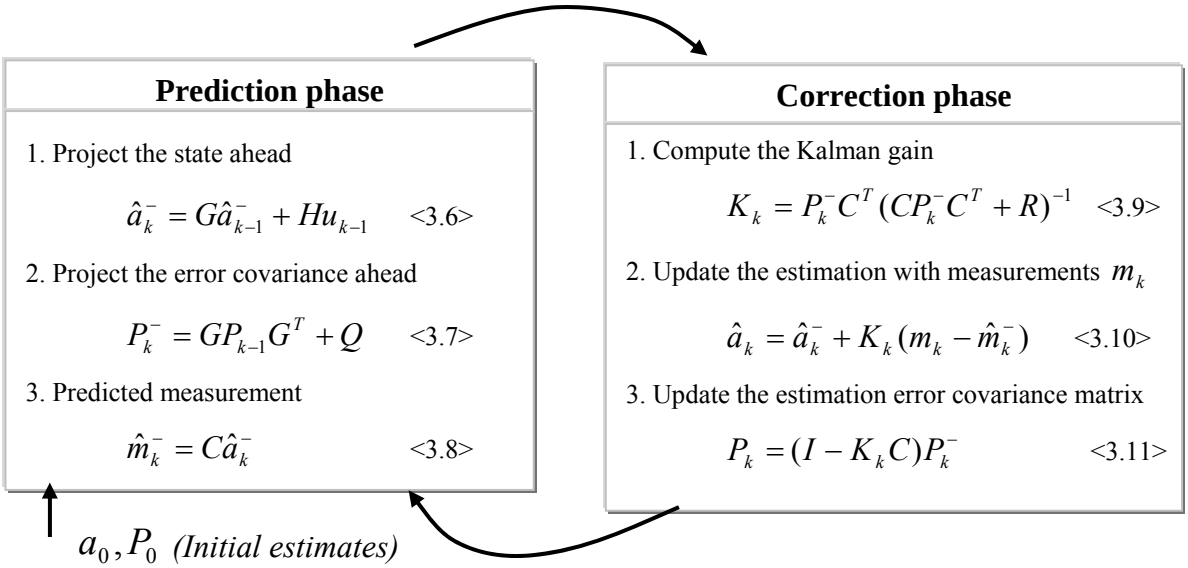

Figure 3.4: A complete picture of the equations of the KF.

K is called the Kalman Filter gain and is a $n \times m$ matrix that minimizes the *posteriori* estimate error covariance in Equation 3.11.

Equation 3.9 uses the predicted measurement covariance matrix S_k:

$$S_k = (CP_k^- C^T + R) \qquad <3.12>$$

The difference between the measurement and the predicted measurement in Equation 3.10 is called *innovation*, or the residual:

$$\hat{r}_k = m_k - \hat{m}_k^- \qquad <3.13>$$

The innovation is weighted by the Kalman filter gain *K*. As it can be seen in Equation 3.9 as the measurement covariance *R* approaches zero, the gain *K* becomes bigger and weights the innovation in Equation 3.10 more heavily. This means that the actual measurement is trusted more and more. On the other hand as the *a priori* estimated error covariance matrix P_k^- approaches zero, K becomes smaller and the predicted measurement is trusted more and more.

3.1.3 Filter Parameters

When the Q and R matrices are constant, the estimation error covariance P_k and the Kalman gain K quickly stabilize and remain constant, as shown in [1]. However, in many applications this is not the case, the measurement error in particular does not remain constant, and sometimes the process noise changes during the filter operation. When it is necessary to take these changes into account Q becomes Q_k and R becomes R_k. In tracking algorithms for example, it is sometimes interesting to reduce the magnitude of Q_k when the object seems to be moving slowly and increase it when the object starts moving faster.

The measurement noise covariance R can usually be calculated during the filter operation. Manipulation of the filter behavior is also possible, by changing the R. For example, if R is made bigger then *K* is reduced and the filter gets slower to respond to the measurements and trusts the predicted states more, see Equations 3.9 and 3.10. And as explained earlier, if R is made smaller the filter responds to the measurements more quickly and the measurements are trusted more.

The state noise covariance matrix Q informs if the process is well known. If it is, the noise covariance can be set to a small value. In cases where the process model is unknown, acceptable estimation results can be accomplished if enough uncertainty is injected via Q.

3.2 The Data Association

As the objective stated the system has to be able to handle multiple and various numbers of objects. In order to deal with this an algorithm for associating the measurement to the different objects is needed. For this kind of association different research groups have developed several algorithms that are classified into two groups: [2]

- Optimal
- Suboptimal

The optimal approach is called Multiple Hypothesis Data Association (MHA). This solution involves all combination of matches between priori estimates and current observations, which are calculated for each time step. At each time step a hypothesis tree is grown, each branch representing a particular combination.

The suboptimal group is further divided into two popular methods known as Nearest Neighbour Data Association (NN) and All Neighbour Data Association.

The All Neighbour approach involves two main implementations known as Probabilistic Data Association Filter (PDAF) and Joint Probabilistic Data Association Filter (JPDAF) [3]. It has been shown [2] that the JPDAF algorithm gives a result that is one of the better. But a disadvantage of using the JPDAF is that a single track can be computationally very expensive.

The NN chooses the nearest measurement within its gate[1] as the most adequate. The PDAF assigns a probability between each object and the measurements within its gate and then uses these for a weighted update. The JPDAF jointly considers all possible data association combinations and calculates their joint probabilities, again for their use in a weighted update stage. [4]

Since the NN chooses the nearest measurement in the estimation process it does not give good results in high clutter density tracking environments, when there is more than one target and when the tracks intersect. An advantage of this method is that it is computationally inexpensive, and regardless of the drawbacks, this method works well with low clutter density and when targets don't interfere with each other. In addition to this the track management is also simple. In Part II [17] a minimization of the execution time is preferred, since the development is done in a real-time platform, and the NN is therefore a good choice. In this thesis, Part I, where the development is done on a simulator platform the PDAF will be used.

1. Gating is explained in Chapter 3.2.1

3.2.1 Gating

In order to speed up the data association step from the optimal solution approach, one strategy is to limit the search region for the measurements. The gating procedure is used to select candidates from the measurements to match with a certain track. The gate, if using with Kalman filters, is designed as follows [5]:

$$V_k = \{z_k : \hat{r}_k^T S_k^{-1} \hat{r}_k < \gamma\} \qquad < 3.14 >$$

Where V_k is the gate, γ is the threshold and $\hat{r}_k$ and S_k come from Equations 3.13 and 3.12. The shape of the gate, if defined like this, is elliptical and centered around the estimated position, as shown in Figure 3.5.

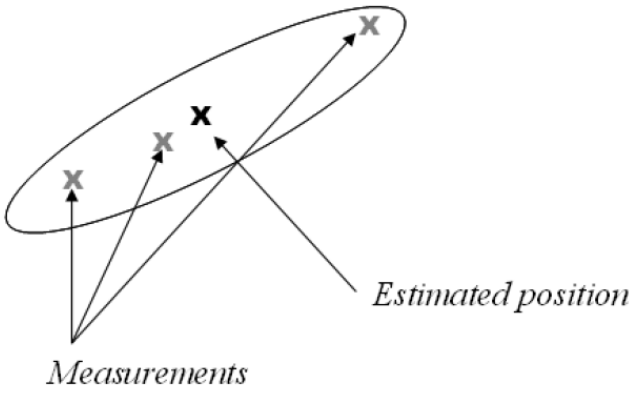

Figure 3.5: Typical ellipsoid gating region.

3.2.2 The Probabilistic Data Association Algorithm

The Probabilistic Data Association Filter (PDAF) uses a Bayesian approach to solve the problem of data association. It proposes a better solution for the association process in situations where there is a single target and no measurements or multiple measurements. However when the tracking problem includes multiple objects the PDAF can difficultly handle the association problem as it will be explained in the following paragraphs.

To select the measurements associated to each target, rather than choosing the data closest to where the object is expected to be, the PDAF weights the influence of various candidate measurements [6].

In the Kalman filter application the PDAF introduces the notation of the *combined* innovation instead of the standard innovation value given by Equation 3.13. The combined innovation, which also is called $\hat{r}_k$, is computed over the *m* measurements detected at a given time step as the weighted sum of the individual innovations, as can be seen in Equation 3.15.

$$\hat{r}_k = \sum_{i=1}^{m} \beta_i r_i \qquad <3.15>$$

Each β_i is the probability of the association event θ_i that the *i*th measurement is originated by the object of interest. These events encompass all possible interpretations of the data so that $\sum_{i=0}^{n} \beta_i = 1$. The association probabilities β_i are derived from an assumed normal Probability Density Function (PDF) on any characteristic related to the correct measurement and a uniform noise model for spurious measurements. [7]

The density function used to obtain β_i uses the idea of a validation gate. The validation gate can be approximated as a tracking window in the form of a circle with a certain radius. This limits the search of measurements to be associated with each target.

As said earlier it is difficult to track multiple objects with the PDAF. This is because one target–originated measurement may often fall within another target's overlapping validation gate. Such persistent interference could lead to multiple trackers locked onto the same target.

The JPDAF solves this problem by sharing information among separate PDAF trackers in order to calculate the association probabilities more accurately. The essential result is an exclusion principle of sorts that prevents two trackers from latching onto the same target [7]. The JPDAF is normally used with Particle Filters. In this thesis the PDAF will be used together with the Kalman Filter.

3.2.3 The Implemented Association Algorithm

To do the association you need to calculate the probability that a measurement belongs to a certain object. This probability can be calculated according to different characteristics. In this case the likelihood β_i is a normalized PDF with zero mean. The PDF is based on the Euclidean distance from each measure to the predicted state vectors. The distance from a measure *m* to a predicted center is calculated as in Equation 3.16 below.

$$d_k^m = \sqrt{\left(x_k^m - \hat{x}_k^-\right)^2 + \left(z_k^m - \hat{z}_k^-\right)^2} \qquad <3.16>$$

If the distance to the closest predicted position is larger than the validation gate radius, *g,* the measure is treated as a candidate for a new track, otherwise it is assigned to the closest existing target. Since the association is depending on the size of the validation gate, this parameter is important when to determine the number of tracks. For example when two people are walking next to each other, then the radius has to be set so that they both can be identified as objects. On the other hand, if the parameter is set too small one object might instead be identified as two.

Since a newly created track does not have a predicted state, the distance from a measure to a new track is calculated to the mean of the measures associated with it. Therefore if one or more of the following measures are associated with a new track, the mean has to be updated when a measure is added.

When all measures have been assigned, the combined innovation of each track, $\hat{r}_k$, is calculated according to Equation 3.15. The combined innovation is used by the Kalman filter to correct the predicted state. The means of the measures associated to each object, which are used for updating the velocity according to Equation 4.2, are also calculated. The structure of the implemented association algorithm can be seen in Figure 3.6 and a more detailed description of the developed association algorithm can be found in Chapter 5.4.

3.2.4 Validation and Removal

As can be seen in the figure below the association algorithm consists of two major parts. In the first part all the measures are associated to different objects. The second part consists of removal or validation of objects.

When a new track is created it is assigned the label *candidate*. This means that it is not considered a *validated* track in the scene until it has been present for a certain amount of time, i.e. a certain number of iterations. The number of iterations you choose as a limit should be adapted to how noisy the input data is.

A track is *removed* when no measures has been associated with it for a certain amount of time. This time, i.e. the number of iterations, must be chosen so that the tracking continues during occlusion.

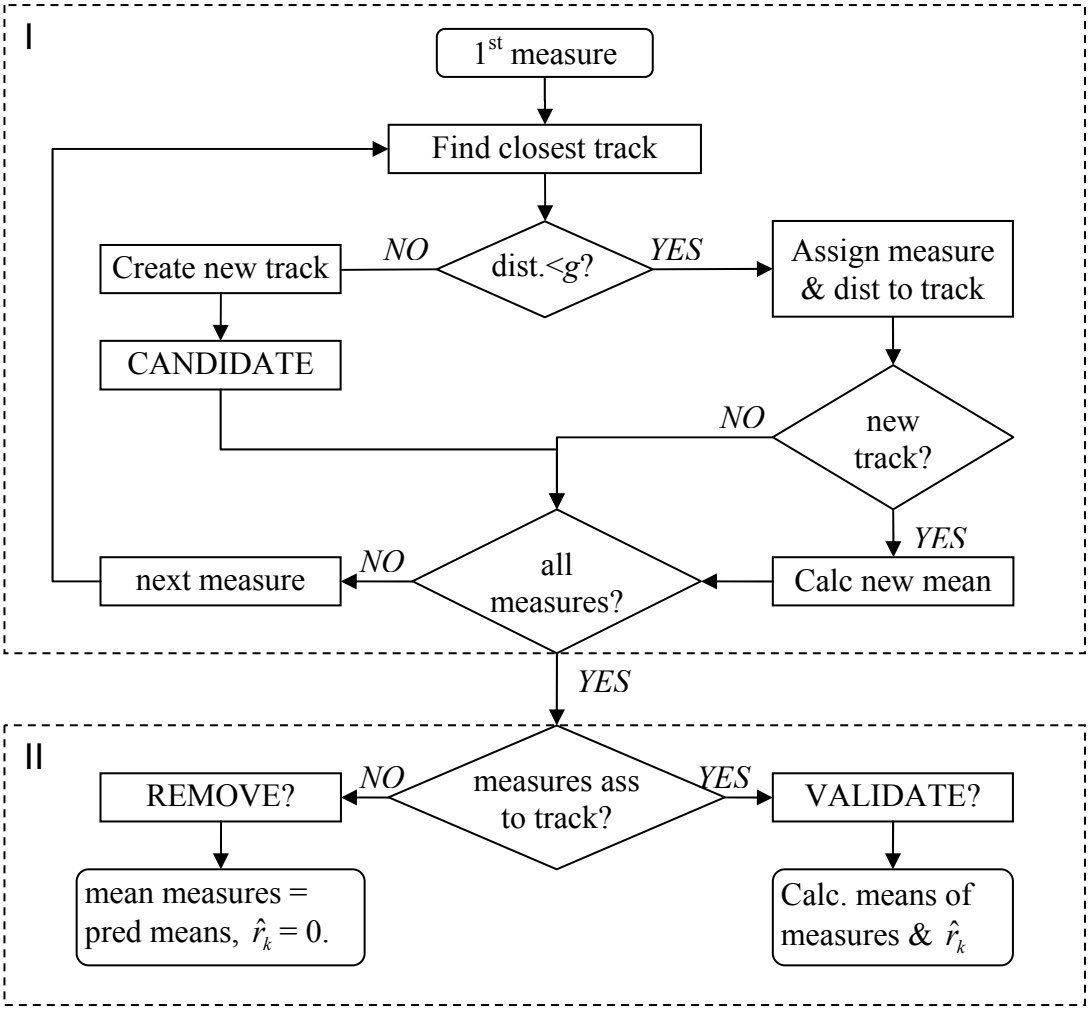

Figure 3.6: Structure of the association algorithm.

4. Models

The format of the input data and the output is described in this chapter. Also to understand the implemented tracking algorithm the system model developed in this thesis is described. The system model defines the state and velocity vectors related to the tracking task. To be able to present the results of the estimated process, it is necessary to know how to transform the 3D data into image coordinates. This is described by the pin-hole model in the end of this chapter.

4.1 The Input Model

The input data used in the tracking and data association processes are in *Cartesian coordinates* (world- or absolute coordinates). This is important for measuring the distance between targets and measurements, and for using state estimation techniques.

The data to be processed in the tracking algorithm comes from a vision system that consists of two cameras. The images from the cameras have already been processed and put in a binary file [12]. This file is organized in the following way:

1. ne - An integer that tells how many points there are in the current frame i.e. the number of XYZ coordinates (see description of the coordinate axis in Figure 4.1).
2. Float X coordinate.
3. Float Z coordinate.
4. Float Y coordinate.

Table 4.1 shows the structure of the input file and Figure 4.1 shows the 3D coordinate system.

Table 4.1: The structure of binary input file.

ne = n	X_1	X_2	. . .	X_n	ne = n	X_1	. . .
	Z_1	Z_2		Z_n		Z_1	
	Y_1	Y_2		Y_n		Y_1	

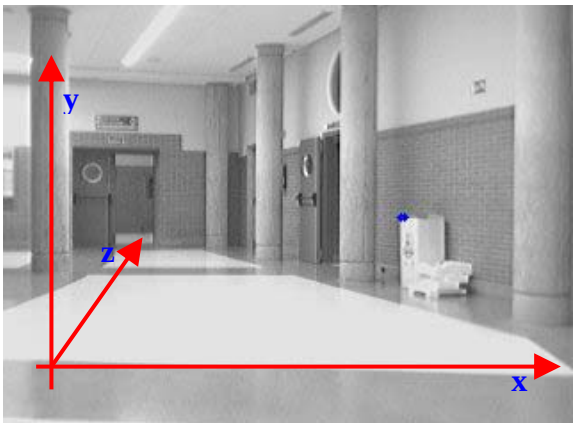

Figure 4.1: The 3D coordinate system.

4.2 The System Model

The system model used in this application, according to the input specifications mentioned before and the estimation requirements commented in the objectives, is described by the equations below. The Kalman filter uses the mean, center of each target, to predict the states, which then are corrected by the mean of the associated measurements.

In the state equation the state vector ($\vec{a}$) consists of the x and z position coordinates. The coordinates are the corrected centers of each object. These coordinates are updated with the help of the velocity of the objects in both x and z directions, as shown by Equation 4.1. T_s is the sample time, 66 ms, of the global discrete estimation process.

$$\hat{a}_k = G\hat{a}_{k-1} + H\hat{v}_k \Leftrightarrow \begin{bmatrix} x_k \\ z_k \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k-1} \\ z_{k-1} \end{bmatrix} + \begin{bmatrix} T_s & 0 \\ 0 & T_s \end{bmatrix} \begin{bmatrix} v_x \\ v_z \end{bmatrix} \quad <4.1>$$

The velocity is calculated in both x and z directions by taking the difference between the mean of the associated measures and the corrected state from the previous time step and then dividing it by the sample time as follows:

$$\hat{v}_k = \frac{m_k - \hat{a}_{k-1}}{T_s} \qquad <4.2>$$

The output equation that relates the state to the measurement is defined by the equation

$$\hat{m}_k = C\hat{a}_k \Leftrightarrow \begin{bmatrix} x_k \\ z_k \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_k \\ z_k \end{bmatrix} \qquad <4.3>$$

A block diagram of the system model is given in Figure 4.2 below.

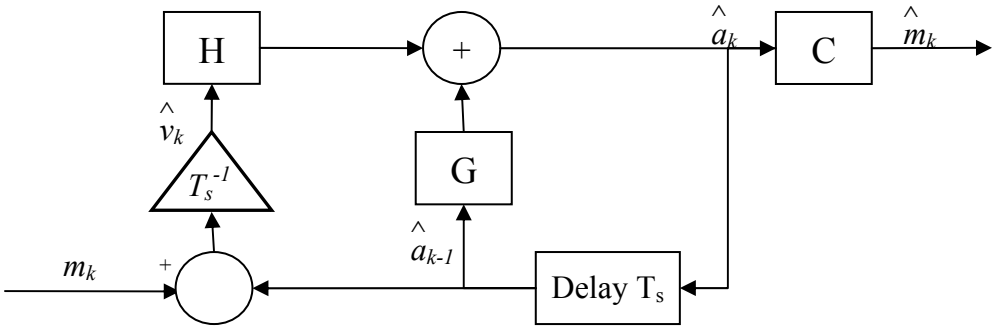

Figure 4.2: System model.

The process noise covariance matrix and the measurement noise covariance matrix are shown in Equations 4.4 and 4.5.

$$Q = \begin{pmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_z^2 \end{pmatrix} \qquad <4.4>$$

$$R = \begin{pmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_z^2 \end{pmatrix} \qquad <4.5>$$

The choice of the covariance matrices is a compromise. In Equation 4.4 σ^2 denote the variance of the process noise and in Equation 4.5 it denotes the variance of the measurement noise.

4.3 The Output Model

All of the objects are moving on the ground and therefore the tracking can be seen as a 2D tracking in the xz-projection plane. The y-coordinates are used for plotting the measures in a 3D-image, where the objects are modelled by a cylinder. The KF models all systems with a Gaussian probability distribution. This distribution and the cylinder model are shown in Figure 4.3.

The center of the cylinder is the corrected state, the mean of the target estimate, which in the figure is represented by the yellow line. The height of the cylinder is fixed, but the y-position depends on the mean of the y-coordinates belonging to each target. The width of the cylinder is an empirically adjusted parameter, the validation gate explained earlier, adapted to the width of the tracked objects.

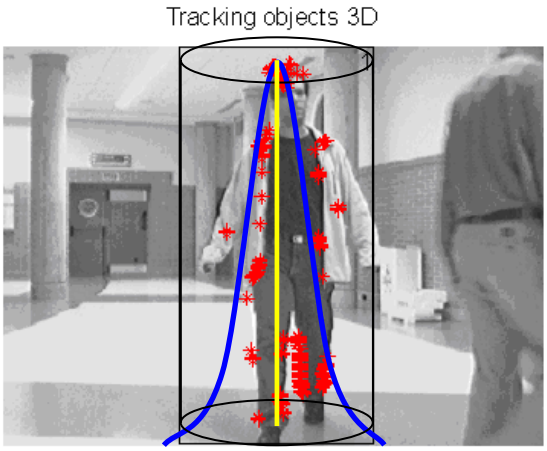

Figure 4.3: Cylinder model.

There is one estimated output from each track and these are visualized in a 2D-projection where the target cylinders are represented by circles, and in a 3D-video where the object cylinders are represented by rectangles. The coordinates of the objects are Cartesian (absolute coordinates in meters or millimeters) and to plot the coordinates in an image, transformation into image pixels is necessary. This is done with the pin-hole model of the camera described in the next chapter.

4.4 The Pin-hole Model

As mentioned earlier the input data are in Cartesian coordinates. This is the traditional system for real world environmental representation. In the ground plane it consists of an x- and y-axis that are perpendicular. To achieve a three-dimensional representation a z-axis, the height, is added perpendicular to the xy-plane. In this thesis the axis-system is changed so that the y-axis represents the height as can be seen in Figure 4.1.

To be able to plot the measures and the tracked state vectors in a video image the absolute coordinates have to be transformed into the *image plane* ones (IP). In order to do this you first need to transform the coordinates into *camera coordinates*.

The different coordinate systems can be seen in Figure 4.4 and are defined as follows:

- The *absolute* or *world coordinate system* (ACS) consists of {*X, Y, Z*} and gives information about the position of the points in 3D space.

- The *camera coordinate system* (CCS) consists of {*X', Y', Z'*}. The origin of the CCS is at *Oc*, and the z-axis coincides with the optical axis of the camera.

- The *image plane coordinate system* (ICS) consists of {*u, v*} and is a 2D system that represents a position in the image plane. The origin *Oi* is the intersection of the optical axis with the IP measured in pixels.

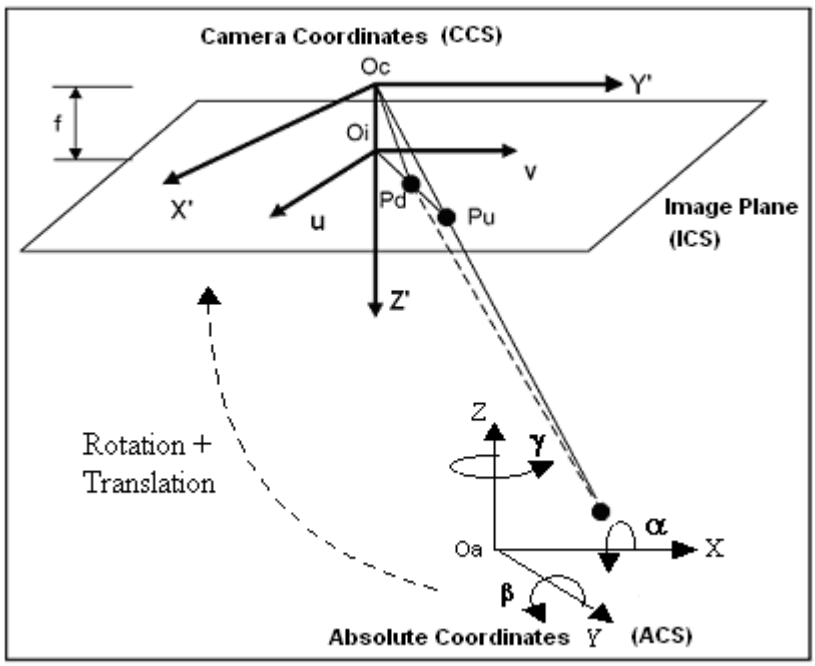

Figure 4.4: Pin-hole model.

The focal length *f* is the distance between the IP and the optical center *Oc*. The distortion-free camera model, or "pin-hole" model, assumes that every point in the ACS is connected to its correspondent image point on the IP in a straight line passing through the focal point of the camera lens, *Oc*. In Figure 4.4 the undistorted image point of the object point, “Pu”, is shown.

The camera has been calibrated so that the CCS and the ACS coincidence. With this calibration, the transformation can be divided into three steps [9]:

1. The first step is to move the origin of the ACS into the origin of CCS.

2. The next step is to rotate the ACS until its axes are coincident with those of the CCS.

3. The final step is to move the IP laterally until there is complete agreement between the two coordinate systems.

These transformations involve some parameters called extrinsic (from ACS to CCS) and intrinsic (from CCS to ICS). The extrinsic parameters include the translation of the origins, the rotation γ around the z-axis and the rotations α and β around the x- and y-axis respectively. The intrinsic parameters include the focal length, *f*, the scaling factors and the image center coordinates [10]. Less important intrinsic parameters such as the distortion coefficients can be consulted in [9].

In the following paragraphs the different matrices needed to do the ACS to ICS transformation are summarized. In the first step, to do the transformation to camera coordinates you need to know the translation matrix **T** that moves the origin of the absolute coordinates into the origin of the camera coordinates:

$$\begin{bmatrix} X'(T) \\ Y'(T) \\ Z'(T) \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \begin{bmatrix} T_X \\ T_Y \\ T_Z \end{bmatrix} \qquad <4.4>$$

In the second step the rotations about the coordinate axes have to be calculated:

$$\mathbf{Z}(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad <4.5>$$

$$\mathbf{X}(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \qquad <4.6>$$

$$\mathbf{Y}(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \qquad <4.7>$$

The sequence of rotations around the ACS coordinate axes can be expressed as $\mathbf{R}(\gamma,\beta,\alpha) = \mathbf{Z}(\gamma)\mathbf{Y}(\beta)\mathbf{X}(\alpha)$ where **R** is a composite rotation, the rotation matrix, in which $\mathbf{Z}(\gamma)$ is applied first, then $\mathbf{Y}(\beta)$ and finally $\mathbf{X}(\alpha)$. The rotation matrix is orthogonal and thus it has the property that $\mathbf{R}^{-1} = \mathbf{R}^{T}$ [9]. The rotation transformation gives the following:

$$\begin{bmatrix} X'(R) \\ Y'(R) \\ Z'(R) \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \qquad <4.8>$$

The generalized displacement (i.e. translation plus rotation) transformation gives the camera coordinates and takes the form:

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} \qquad <4.9>$$

To be able to express the generalized displacement as a product of matrices the matrices must be augmented to 4x4 as follows:

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_X \\ R_{21} & R_{22} & R_{23} & T_Y \\ R_{31} & R_{32} & R_{33} & T_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \qquad <4.10>$$

Finally in, the third step, to extract the image coordinates the CCS coordinates have to be normalized by *Z'* and multiplied by the focal length *f*. That gives the image coordinates in meters or millimeters, depending on the units of *f*. In order to plot the coordinates in the image it is necessary to transform them into pixels. That is done by dividing by the scaling factors s_x and s_y (meters/pixel) [10] and correcting the position by adding the image centers o_x and o_y shown in Figure 4.5.

The equations below show how to project the object coordinates onto the image plane:

$$u = \frac{f}{s_x} \cdot \frac{X'}{Z'} + o_x \qquad v = \frac{f}{s_y} \cdot \frac{Y'}{Z'} + o_y \qquad <4.11>$$

Figure 4.5 shows the image center offset that needs to be corrected for in the transformation into image pixels.

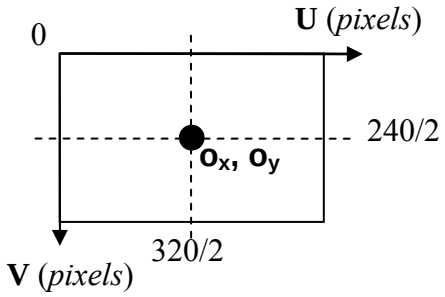

Figure 4.5: Image center offset.

5. Implementation

The association and Kalman filtering described in previous chapters are implemented in Matlab. The input data comes from a *dat* file, see Chapter 4.1. The output from the Kalman filter is visualized in a 2D projection plot and in the input video. The probabilities of finding objects in the scene are also plotted in an *object probability* plot.

The developed Matlab code *main.m*, *initiate.m*, *association.m*, *kman.m, visualize.m, transf.m* and *constants.m* can be found in Section V. An overall view of the global implementation is presented by the flow chart in Figure 5.1. The different parts of the implementation, the input, initiation, association, Kalman filtering and visualization are then explained more in detail. The implementation was done in three steps; first with one single object, then with a constant number of objects and finally with a varying number of objects.

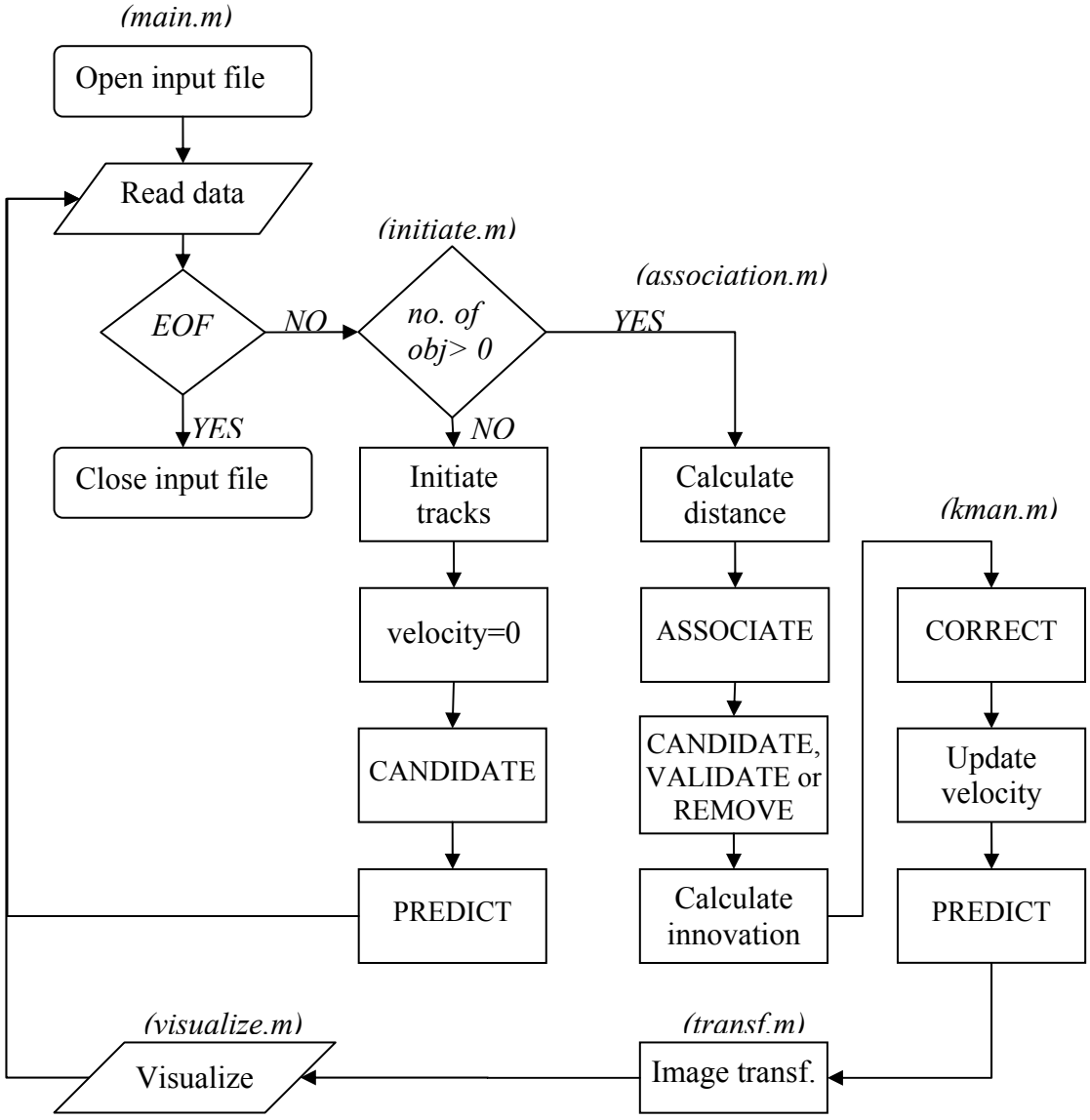

Figure 5.1: Flowchart of implementation.

5.1 Input Data

As explained earlier the images from the vision system have already been processed and the results have been put into a binary *dat* file. The structure of this file is explained in Chapter 4.1. To read the input file the Matlab command *fread* is used. Firstly the integer *ne* is read, which tells how many measures that are obtained. Then the x, y and z-coordinates of each point extracted from the frame are read and saved in a matrix as 32 bits floats. As all frames have been read and processed, the input file is empty and closed.

5.2 Number of Objects

In order to simulate the functionality of the Kalman filter, the tracking task was first developed with one single object. Then the association algorithm was implemented with a fixed number of objects. Finally, the filtering was done with a varying number of objects, which leads to the tracking tasks that was described in the objectives, Chapter 2.

- When tracking one single object, the tracking problems commented in the objectives are not present. If there is only one object, an association algorithm is not necessary; all the data is associated to the object, including noise.

- When the number of tracks is known a priori and remains fixed throughout the motion sequence the measurements are associated to the closest object. This is called Nearest Neighbour association which has the capability of *explicit modeling of measurements,* which is explained in next paragraph. Since the number of objects is constant, tracks should not be terminated during the simulation. If no measurements are assigned to an object it means that it is occluded and the tracking algorithm has to have the capability of *track continuation*.

- When tracking multiple objects where the number of objects is changing over time the association algorithm has to integrate the capabilities of

  Track Initiation: The creation of new tracks when new objects are supposed to enter the field of view and prevent that noise is characterized as a new track.

  Track Termination: The termination of a track when no measurements are associated to the object for an extended and programmable period of time.

  Track Continuation: The continuation of a track over several programmable frames in the absence of measurements associated to it. Thus the tracking algorithm is capable of providing support for temporary occlusion.

  Explicit Modeling of Measurements: A measurement may only be assigned to a single track.

The tracking characteristics described above are going to be explained further in the following chapters.

5.3 Track Initiation

Each time a new track is initiated a new Kalman filter is started. The Kalman filter associated with each track cannot be initiated from a single iteration since it does not provide the velocity information included in the model. There are two solutions to this problem.

1. The first one is to delay track initiation until two consecutive and associated measurements are available to give a reliable estimate of a feature's velocity.

2. The second solution, which is implemented, is to initiate the velocity estimates to zero while simultaneously initializing the corresponding elements of the error covariance matrix to a value that represent the uncertainty in the velocity estimates [11].

To initiate tracks the objects have to be "found" in the scene:

- At the beginning (t=0) it is assumed that there are no objects in the scene. In this situation the first measure that is received is defined as a candidate track.

- From this moment, each time a new measure is received the distances to the already existing tracks are calculated.

- If the distance is bigger than the radius, *g*, of the validation gate, the measure is the beginning of a new candidate track, otherwise the measure is associated to the closest existing track and a new mean is calculated.

- This continues until all the measures in the frame have been associated. The predicted states are then computed by adding the velocity information as shown by Equation 4.1.

If the initial estimate was absolutely correct then P_0 would be zero. But as the initial estimate is based on measurements and the initial velocity is set to zero there is an uncertainty in the estimation and P_0 is assigned the values according to Equation 5.1 (in mm^2):

$$P_0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad <5.1>$$

5.4 Data Association

The association algorithm functionality can be summarized with these two statements:

I) When assigning the measurements each measure may either
   1. Come from a previously known track.
   2. Be the start of a new track, e.g. a previously not detected object that has entered the field of the camera.
   3. Be a spurious measurement, a so called outlier (also called *false alarm*).

II) In addition, at the end of the association process, for those tracks that have not been assigned any measurements there is the possibility of
   4. Deletion of the track. This situation may arise when an object leaves the field of the camera.
   5. Continuation of the track. The missed measurements are perhaps due to either noise or a temporary occlusion caused by the motions of the camera or objects in the scene

The idea of using 3 different states for an object track; candidate, validate and remove were discussed earlier in Chapter 3.2.4. Figure 5.2 shows the different numbered possibilities, 1-5, of the tracking. The implemented association algorithm consists of two parts where the association is done in the first part and the validation or removal is done in the second.

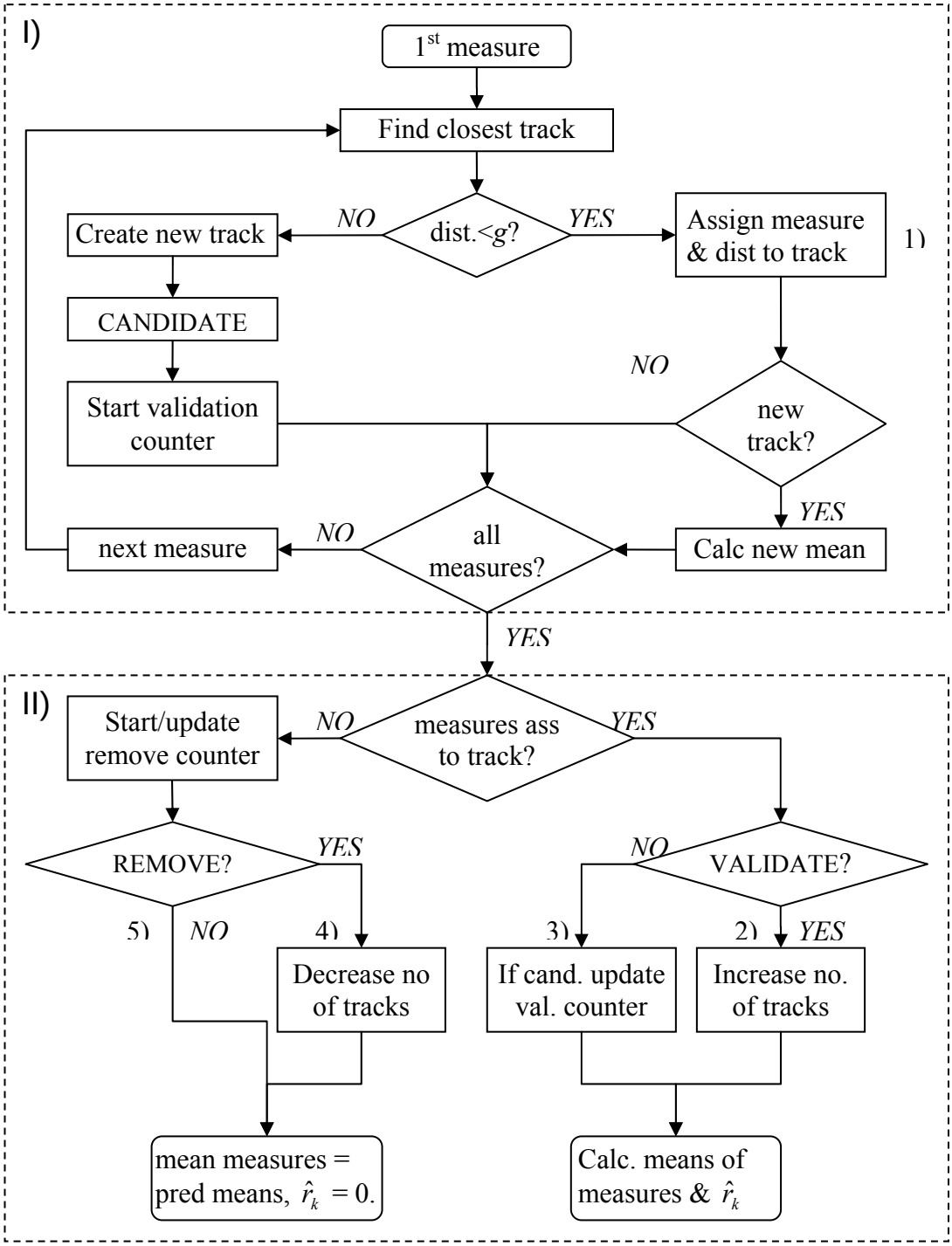

Figure 5.2: The association algorithm.

One property of the association algorithm developed in this thesis is that a measurement may only be assigned to one track. Which track should it be assigned to? The first part of the association algorithm solves this:

- Firstly the Euclidean distances from each measurement to all existing tracks are calculated, see Equation 3.16. Then the algorithm search for the closest track.

- If the number of objects in the scene were constant, the measure would automatically be assigned to the closest track. But with a varying number of objects, should the measurement be assigned to the closest track or treated as a new one? This question is solved by the validation gate – if the measurement is within a radius g of an existing tracks predicted state it belongs to the track, otherwise it is treated as a candidate for a new track.

- When a measure does not belong to a previous known element it gets the label candidate, and the *validation counter* is started. The label candidate means that it can either be the beginning of a new track or just a false alarm.

- If a measure is assigned to a track that has been created in the same iteration a new mean is calculated.

In the second part the tracks may either be validated or removed:

- When a candidate has been present for a certain programmable time, it is validated as a new track and the number of tracks is increased.

- If the candidate is not validated the validation counter is updated.

- When no measurements are associated with a present track it becomes a candidate for removal and the *removal counter* is started.

- When the removal candidate has been absent for a certain programmable time it is removed and the number of tracks is decreased.

Now with the described implementation the first three specifications of the association algorithm have been taken into consideration; *Explicit Modeling of Measurements, Track Initiation* and *Track Termination*. But what about *Track Continuation* - how is the track continued over several frames in the absence of associated measurements? Since no measures are assigned to it, the center of measures simply has to be assumed to be equal to the predicted center, which is a good approximation.

To increase the robustness of the association algorithm a factor of probability is added, i.e. the probability that a measurement belongs to a certain object. This probability is used for calculating the combined innovation $\hat{r}_k$ according to Equation 3.15. If the track is in the state of *Track Continuation* no measures are associated with it, and the combined innovation related to the track is therefore zero.

5.5 Kalman Estimation

As described earlier in chapter 3.1, the KF consists of a prediction step and a correction step. The prediction step predicts the state of the next time step. Therefore, at each time step, the correction of the current state is done first and then the prediction of the next state. To be able to predict the state it is necessary to know the velocity of each target. The filtering function consists of the following steps which also can be seen in Figure 5.3:

1. The first step is to correct the predicted state and error covariance as in *Figure 3.4*, *correction phase*. To increase the robustness, the original innovation that is used by Equation 3.10 to compute the Kalman gain has been replaced by the combined innovation, see Equations 3.13 and 3.15. As discussed earlier if a track does not have measurements associated to it the combined innovation is zero. This means that the state is not corrected but just assigned the predicted state.

2. The next step is to update the velocities, which is the model input, according to Equation 4.2. This is done using the difference between the mean of the measures and the corrected state vector from the previous time step. As mentioned earlier, in the case when no measures are assigned to a track, the mean of the measures are assumed to be equal to the predicted state vector. The calculated movement is then divided by the sample time.

3. The final step is to predict the next estimation by using the system model according to Equation 4.1 and the error covariance according to Equation 3.7.

Table 5.1 below shows the functionality of the different time steps of the KF designed for the application of this thesis and Figure 5.3 shows a block diagram of the Kalman filtering model.

Table 5.1: Time steps in KF.

	t=0	t=1	t=2	t=n
1. Correction	$\hat{a}_0 = m_0$	$\hat{a}_1 = \hat{a}_1^- + K\hat{r}_1$	$\hat{a}_2 = \hat{a}_2^- + K\hat{r}_2$	$\hat{a}_n = \hat{a}_n^- + K\hat{r}_n$
2. Velocity	$v_0 = 0$	$v_1 = \frac{m_1 - \hat{a}_0}{T_s}$	$v_2 = \frac{m_2 - \hat{a}_1}{T_s}$	$v_n = \frac{m_n - \hat{a}_{n-1}}{T_s}$
3. Prediction	$\hat{a}_1^- = \hat{a}_0 + v_0$	$\hat{a}_2^- = \hat{a}_1 + v_1$	$\hat{a}_3^- = \hat{a}_2 + v_2$	$\hat{a}_{n+1}^- = \hat{a}_n + v_n$

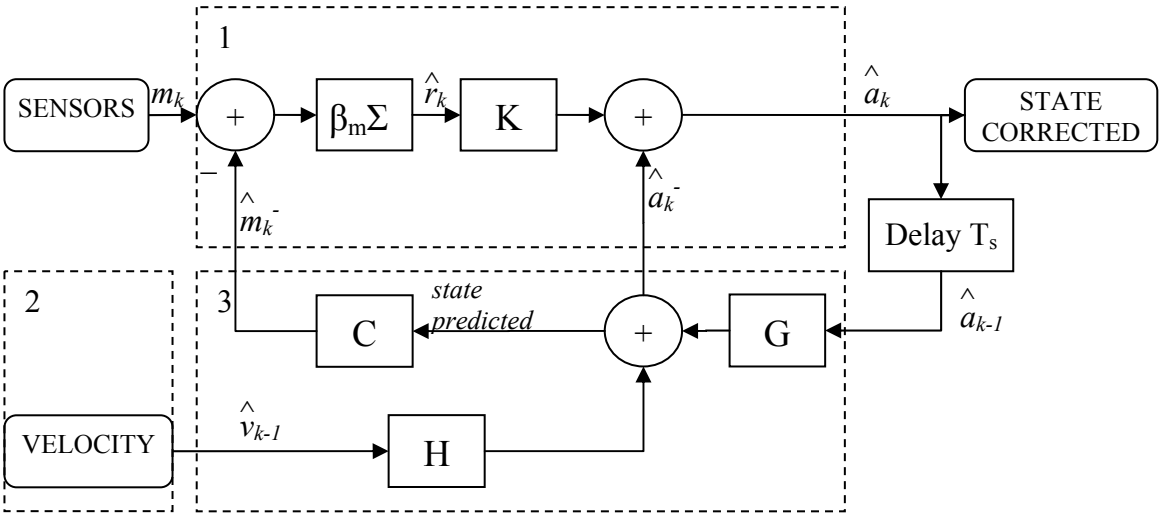

Figure 5.3: Model of Kalman filter.

The values of Q and R used in the implementation can be seen in Equations 5.2 and 5.3. As mentioned earlier the diagonals of the matrices are the x and z variances of the process noise and the measurement noise (in mm^2). The true values of these variances are unknown. Sometimes the measurement noise R can be measured prior to the operation of the filter, but in this case the true point of a measurement's position is unknown. The true value of Q can not be determined since the process can not be directly observed.

The value of R is obtained by dividing the variance of the measurements associated to an object by the validation gate, as it is explained in Part II [17], Chapter 6.1.1. The value of Q is obtained by tuning of the filter.

$$R = \begin{pmatrix} 0.1 & 0 \\ 0 & 0.1 \end{pmatrix} \qquad <5.2>$$

$$Q = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad <5.3>$$

How the tracking result is affected when these values are manipulated is discussed in Chapter 6.3.3.

5.6 Execution Time

The camera's frame rate is 15 fps, giving the system a sample time of $1/15 \approx 66ms$. This means that in a real time implementation, as in Part II [17], the loop execution time should not exceed this expected execution time.

As the implementation in this part is done on a simulator platform, the execution time is not as important as when implementing on a real time platform. A way to compensate for an execution time that is longer than the sample time is to calculate the velocity with the measured execution time. If the measured execution time is shorter than the expected, the system is paused for the remaining of the sample time.

5.7 Visualization

As mentioned earlier the output from each Kalman filter is plotted in both a 2D projection and in a 3D image as in Figures 5.4 and 5.5. In both plots the measures are visualized as red stars. In the 2D image, which can be seen as a projection of the walking plane, each target is represented by a circle. The center of each circle is the corrected state obtained from the Kalman filter and each target is drawn in a different color.

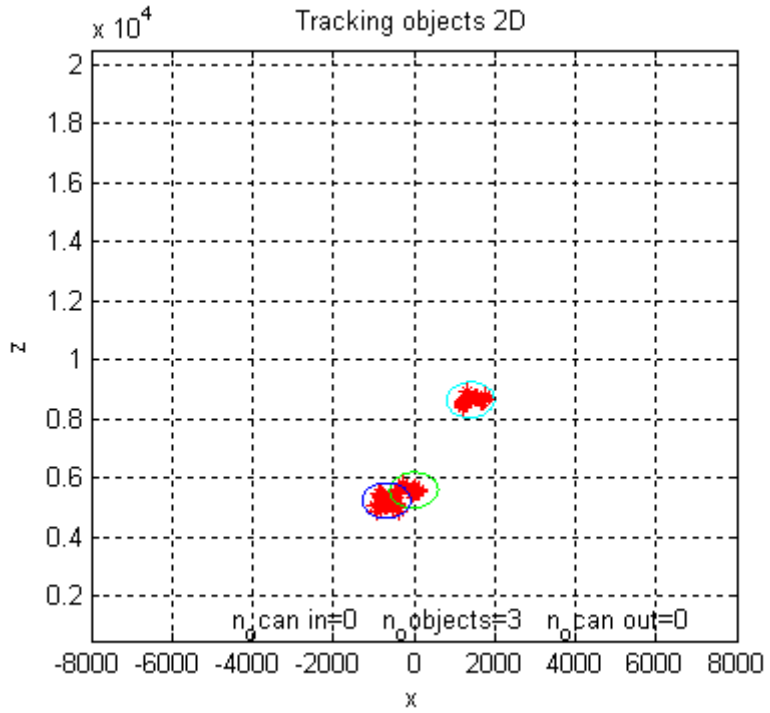

Figure 5.4: 2D plot of tracking with circles.

To show the input video the Matlab command *imshow* is used. To plot the red measures and the rectangles in the image the Cartesian coordinates used in the model must be transformed to image coordinates as explained in Chapter 4.4. The

center of each rectangle is represented by the corrected states of each track and the mean of the associated y-coordinates.

.

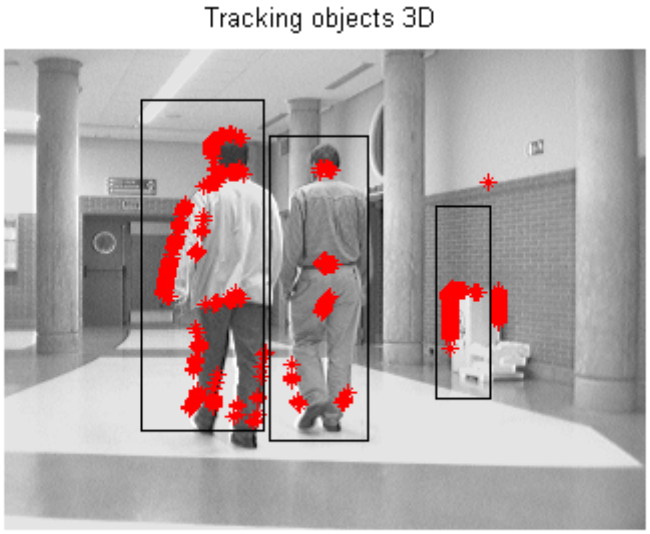

Figure 5.5: 3D image with rectangles.

The size of the input image (u, v) used in the experiments is 320x240 pixels. The extrinsic parameters (rotations around the ACS-axis and translation of the center) are shown in Table 5.2, and the intrinsic parameters (the focal length divided by the scaling factors and the image centers offsets), are shown in Table 5.3:

Table 5.2: Extrinsic parameters.

γ (rad)	α (rad)	B (rad)	T_y (mm)
0.0166	0.019508	-0.014053	970

Table 5.3: Intrinsic Parameters (pixels).

f/s_x	f/s_z	o_x	o_y
430.79014	431.72027	151.26555	117.03242

The *object probability* is also plotted, as Figure 5.6 shows. The figure informs about the normalized probability, obtained from the tracking algorithm, of where to find an object in the scene at each iteration step.

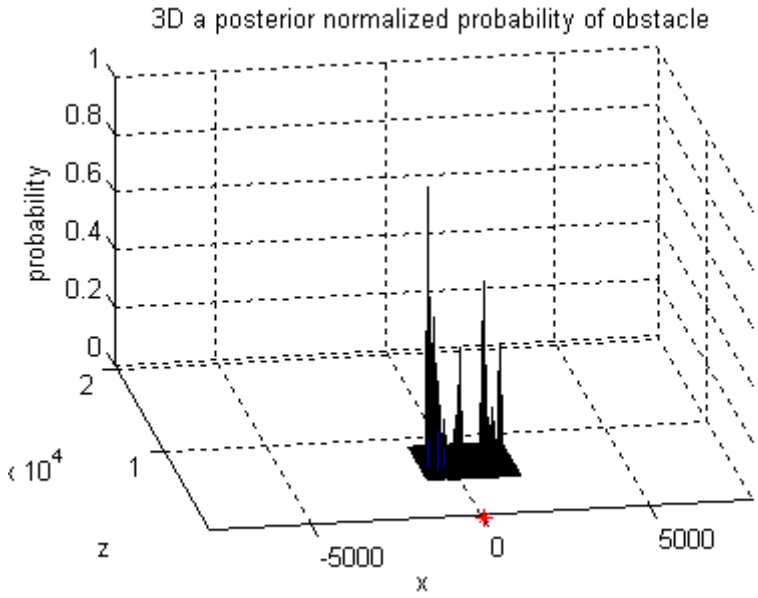

Figure 5.6: Object probability plot.

6. Results

In this chapter the tracking results from the already described theory and implementation are presented. This includes the results when using a single object, a constant number and a varying number of multiple objects.

The tests were done with a fixed R and Q according to Equations 5.2 and 5.3 and P_0 is fixed to the value of Equation 5.1. The limit of the validation counter is set to 4 time steps and the limit of the removal counter is set to 15 time steps. The reason for the difference of the counters' limits is that outliers, spurious measurements, only last a few time steps while an occlusion takes longer time. The validation gate is determined through tuning to the final value of 850 mm.

After the initial tests the accuracy of the tracking algorithm is investigated and then the influences of the different parameters in the paragraph above are tested.

The implemented results will be presented in a 2D xz-projection plot, where the scaling of the axis is in mm, and in a 3D image of the scene.

6.1. Number of Objects

As explained in Chapter 5.2 the implementation of the association algorithm depends on the actual scene; if it consists of one object, a constant number of objects or if the number of objects is changing. The most difficult part is to track a varying number of objects, and therefore a major part of this chapter consists of these results, where different states of the tracking (candidate, validate and removal) are tested in different situations.

6.1.1 Tracking a Single Object

As commented before when tracking one single object no association algorithm is necessary and problems like validation, occlusion and crossing are not present. All the input data is associated to the object, including noise. This could lead to the estimator losing the target as can be seen in Figure 6.1. The estimator is tracking the measures that can be seen in the left plot of Figure 6.1 without problems. Suddenly noise creates a temporary clutter, a so called outlier, and the estimator loses the target as can be seen in the right plot. The reason for this target loss is that the estimator corrects the predicted position with the mean of all the input measures, including noise ones.

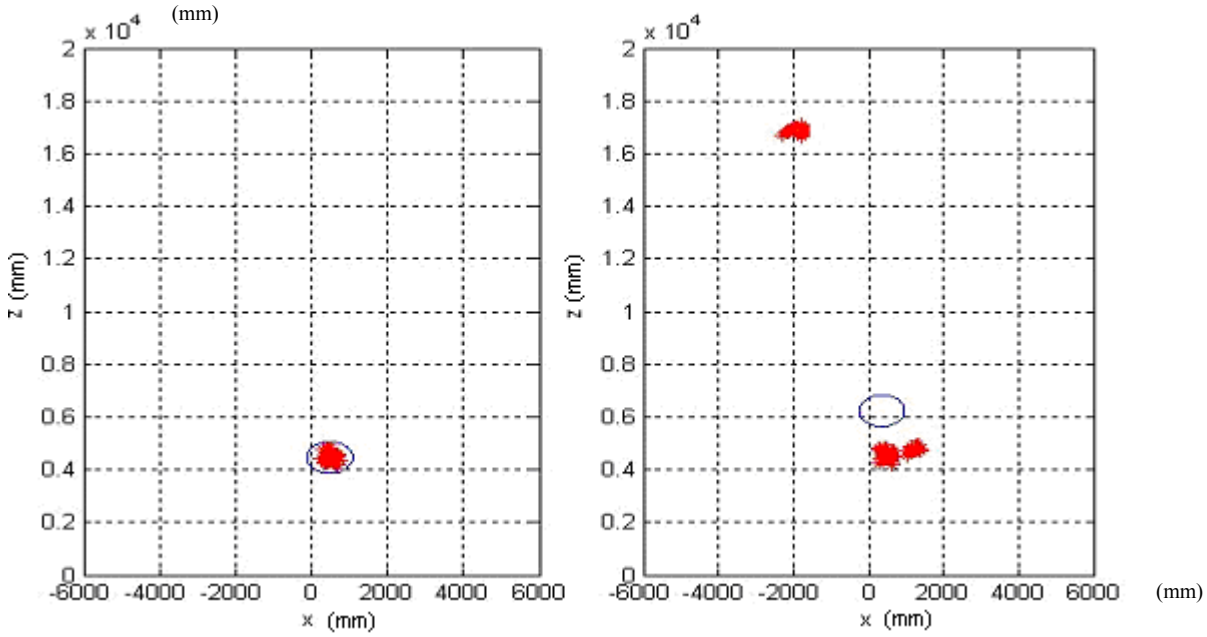

Figure 6.1: Tracking a single object.

6.1.2 Tracking a Constant Number of Multiple Objects

When the number of tracks is known a priori and remains fixed throughout the motion sequence the measurements are associated to the closest object with a NN algorithm. If the initial position of the objects is known prior to the tracking a validation gate is not necessary since no objects have to be found in the scene.

Since the number of objects is constant tracks should not be terminated during the tracking. If no measurements are assigned to an object it means that it is occluded. How the track is continued during occlusion is presented in part c) of the next chapter.

Another problem with the association that has to be taken into consideration is crossing. An example of a problematic crossing can be seen in Figure 6.2 where it is shown that the expected tracks are followed without any problems. The crossing is solved by the algorithm itself since an input parameter to the Kalman filter is the velocity.

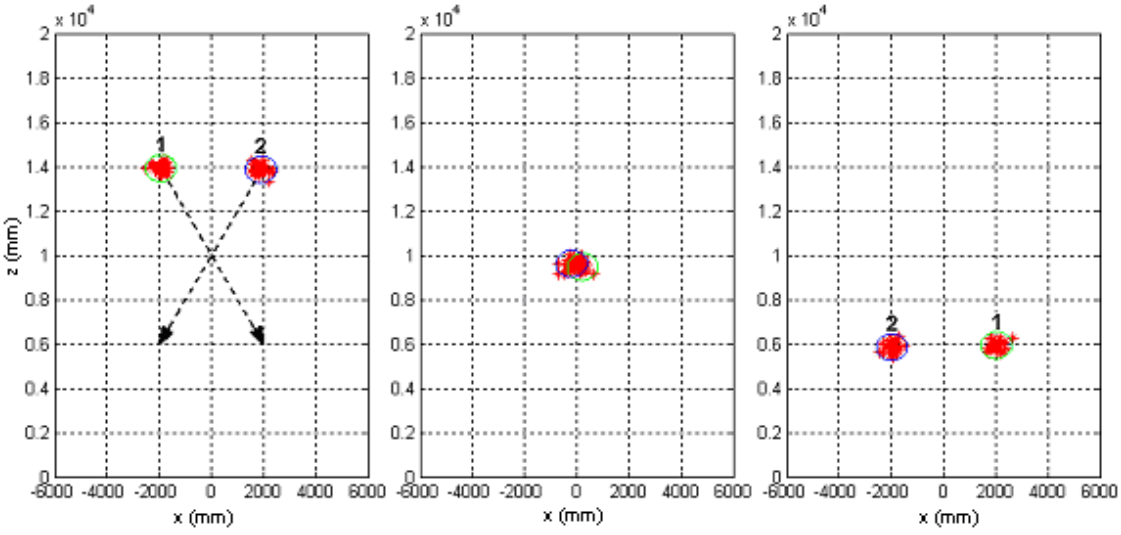

Figure 6.2: Crossing.

This simulation does not use input data from a real scene. The data is generated in Matlab so that two objects cross each other in the point (0, 10000). In a real environment this extreme situation would not occur unless the objects jump over each other. On the other hand, in a real scene, the disadvantage is that objects do not always move in straight lines and with constant velocity. The approximated velocity may then not always be correct. This complicates the association process and the crossing phenomena since an incorrect velocity leads to a bad prediction of the state. If the predicted state is inaccurate the measures might be associated to the wrong tracking estimator.

In other words, in Figure 6.2, if the predicted states would be inaccurate, tracker 1 and 2 might switch target at the crossing so that the positions of tracker 1 and 2 in the right picture are converted.

Usually the initial position of the objects is not known a priori and the objects have to be located in the scene before any tracking can be implemented. To do this a validation gate is necessary - all the measures within a certain radius are associated with the same object. How the tracking initiation is implemented is described in Chapter 5.3. The difficulty is to determine the radius of the gate. The number of objects present in a cluster is determined by the size of the gate as it is shown by the first example in the following chapter.

6.1.3 Tracking a varying Number of Multiple Objects

a) The validation gate radius

An important task when tracking a varying number of multiple objects is to determine the actual number of objects in the scene. As commented in the pervious chapter the number of objects present in the scene depends on the radius of the validation gate. Figure 6.3 below shows the result of an experiment in which the number of objects is being estimated correctly even though two objects move close to each other with the same velocity. The effect of changing the validation gate radius will is presented in Chapter 6.3.2.

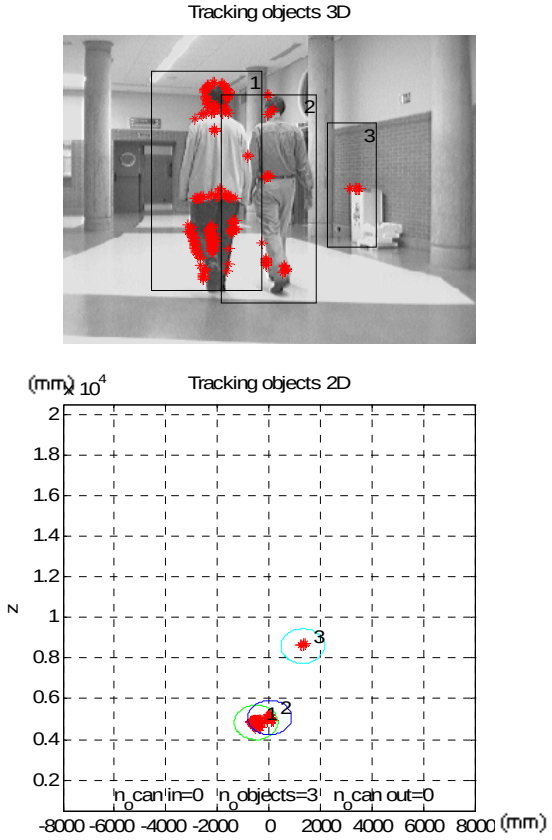

Figure 6.3: Correct number of objects.

b) Validation or false alarm

Another important task is to determine if a measure that is not within the gate of an existing track is the start of a new track or just a false alarm (outlier). This problem is solved by using the validation counter. If a candidate has been present for a certain amount of time its track is validated. Figure 6.4 shows the result of a candidate that is being validated. When the candidate is created the number of candidates for validation, "n_o *can in"*, increases from zero to one. When the object is validated the number of objects, "n_o *objects"*, increase from zero to one and the number of candidates is reset.

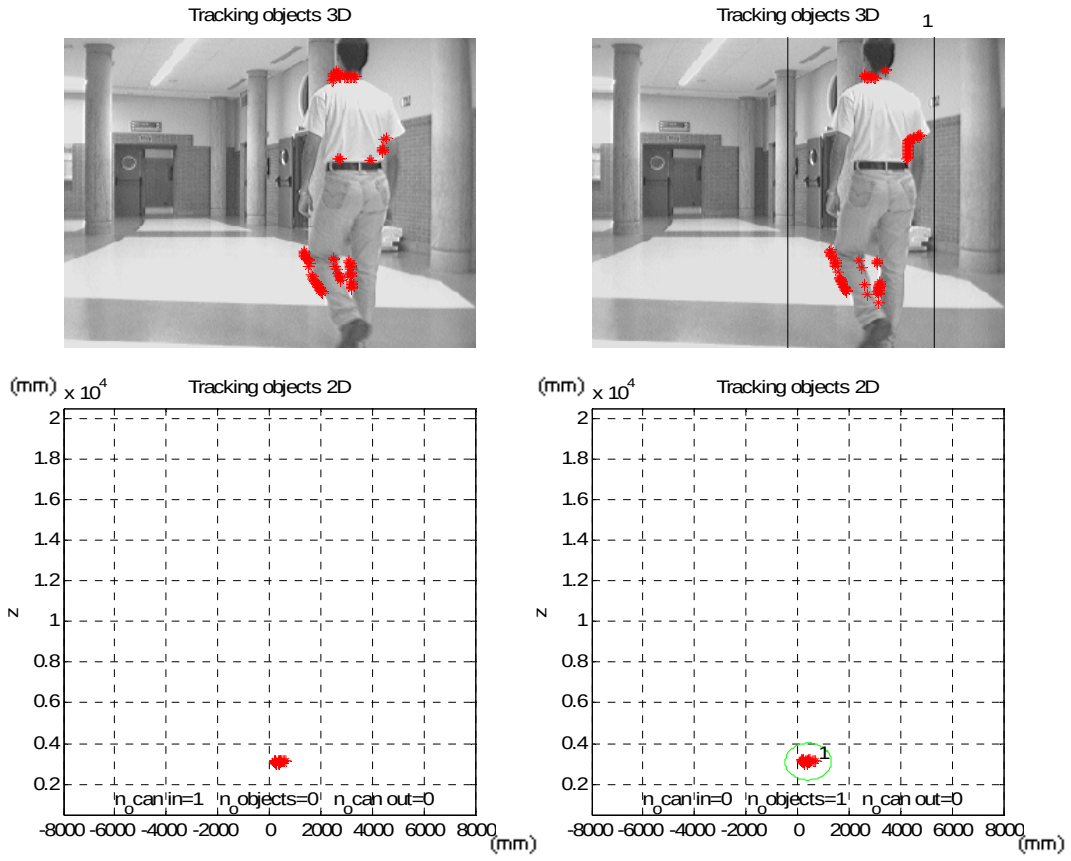

Figure 6.4: Candidate and validate.

Figure 6.5 shows an example of a false alarm; a track is created as a candidate but it is not validated. Here the number of candidates is also increased and then restored when the validation counter ends, but the number of objects remains constant.

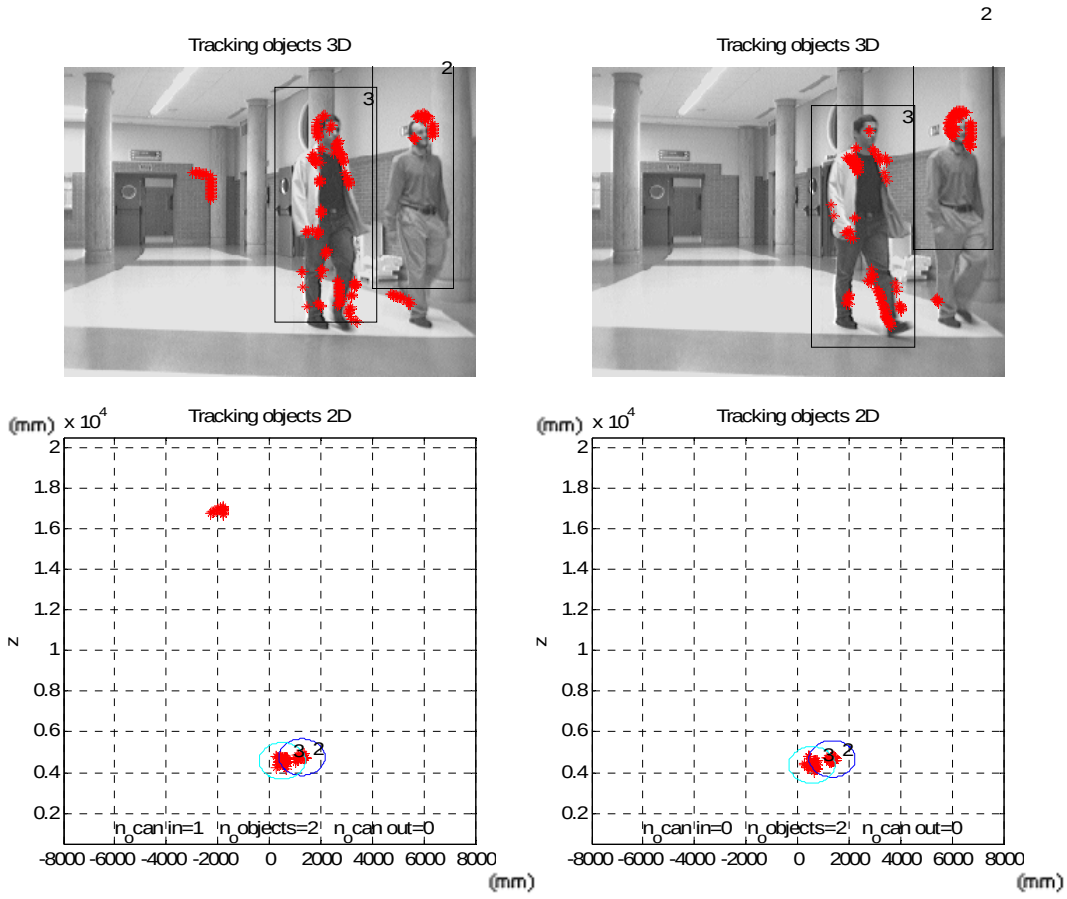

Figure 6.5: False alarm.

c) Removal of continuation of a track

The opposite problem of validation, to remove an object or continue tracking, also has to be solved. Figure 6.6 shows a situation in which two objects get occluded for a while. Instead of removing the tracks immediately the removal counter is started and the tracking of the objects continues.

As can be seen in Figure 6.6, when the objects are occluded no measures from them are received as input to the tracks. The number of candidates for removal, "n_o *can out"*, increases to two while the number of objects, "n_o *objects",* is kept the same during the time.

Then when object nr 1 appears again, as can be seen in Figure 6.7, the number of candidates for removal decreases to one. Object nr 3 is absent for too long and therefore its track is removed when the counter ends, and as a result the number of objects is decreased from two to one.

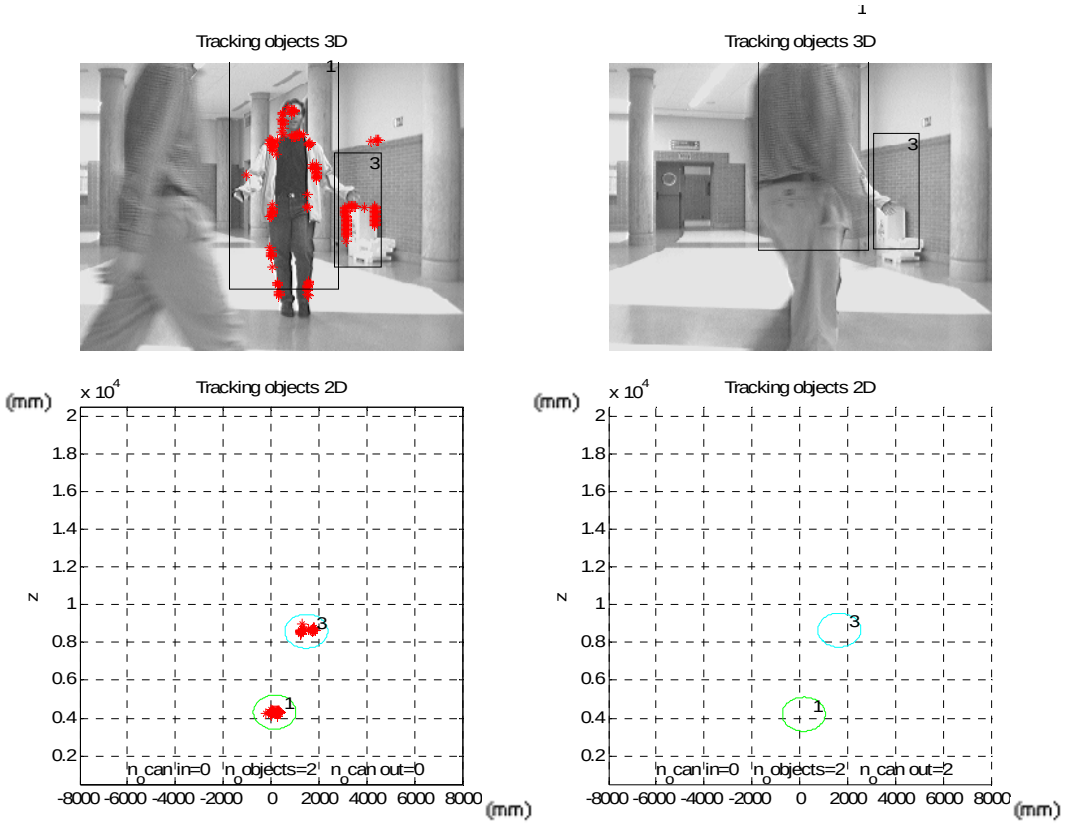

Figure 6.6: Occlusion.

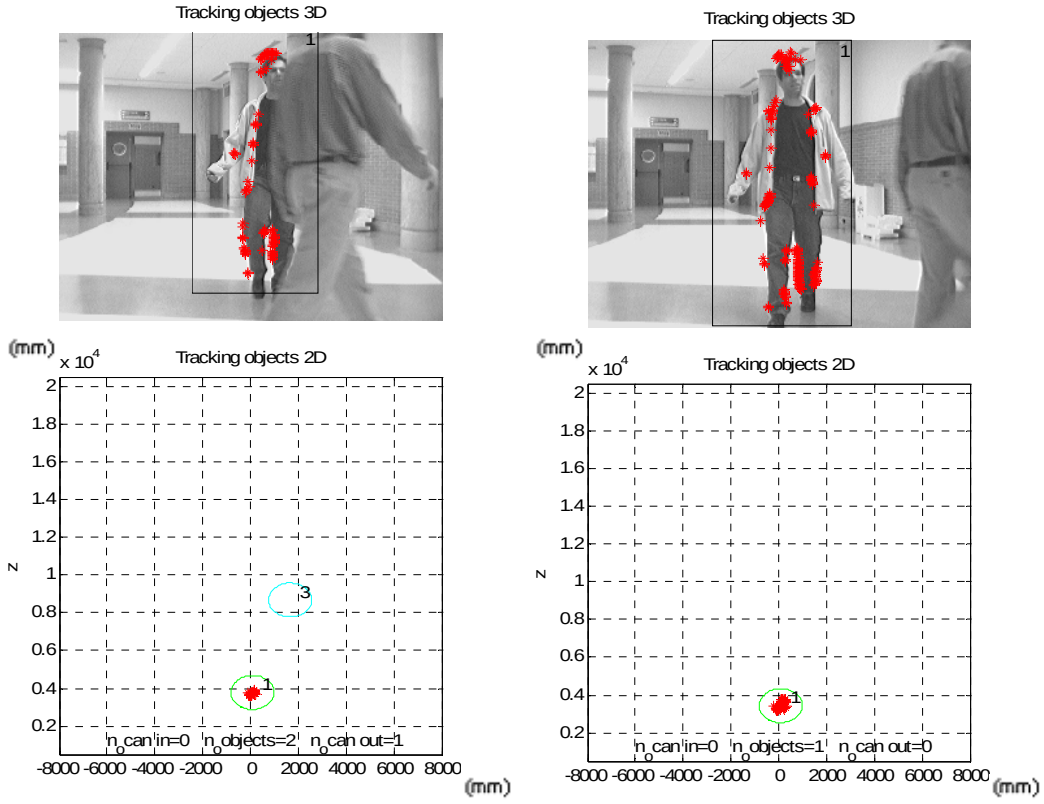

Figure 6.7: Occlusion and removal.

As the presented results show the developed tracking algorithm seems to be working correctly. The number of objects detected in the scene is determined correctly by the validation gate. A candidate is validated when a new target is supposed to enter the scene and treated as a false alarm when outliers appear. The tracking is continued during occlusion and when a target is absent for too long its track is removed

6.2 Accuracy and Errors

As showed in the results above the developed tracking algorithm, including validation and removal of tracks, is working correctly. To determine if the number of detected tracks is accurate the number is compared to the manual background truth, i.e. the number of objects calculated manually in each frame.

Another interesting factor to test is the estimation error that will inform about the accuracy of the developed tracker. Different tests show that almost any initial estimation error covariance matrix P_0 can be used, as the filter *eventually* converges. The only value that can not be used is P_0=0 since the Kalman gain would then initially and always be zero. If K is zero the state is not corrected with the measurements, it only takes the value of the prediction.

The plots in Figures 6.8, 6.9 and 6.10 were obtained from a scene with a single object that is in a comfortable view of the cameras. The objects position in the scene affects the accuracy of the obtained measures. The choice of P_0 determines how long it takes for the filter to converge. As can be seen in Figure 6.10 if an initial value according to Equation 5.1 is used it only takes the filter 4 iterations to converge to the value 0.09161 mm^2. Other parameters that affects the time of convergence will be discussed in Chapter 6.3.3.

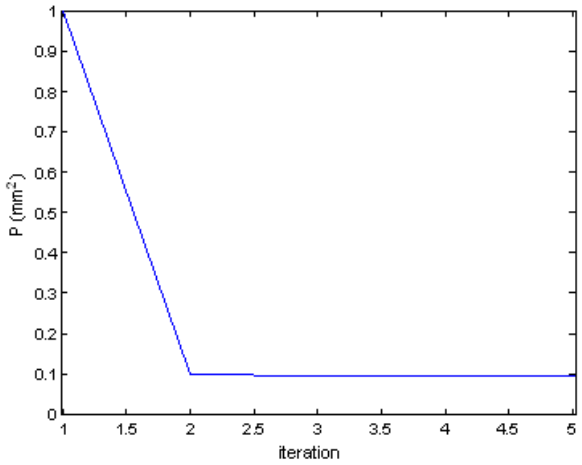

Figure 6.8: Convergence of *P*.

As discussed in Chapter 5.5 the input data only comes from measures and the true points are unknown and therefore the true values of Q and R can not be obtained, only the ideas of them. To measure the accuracy of the estimations some samples should be compared to its correct positions, called the background truth. But since the exact positions of the measurements are unknown the qualities of the estimators

have to be measured with the *manual* background truth. This is done by measuring the distance from the estimated centers to the means of the measurements associated with each track. The manual background truth only gives an idea of how good the accuracy of the estimators is.

Figure 6.9 shows the absolute distance between the estimates and the centers of measures and Figure 6.10 shows the positions of the estimates as blue plus signs and the positions of the centers of the measures as red dots. As can be seen in the figures, the distance between the estimated states and the centers of measures is small, which means that the accuracy of the tracking algorithm is good.

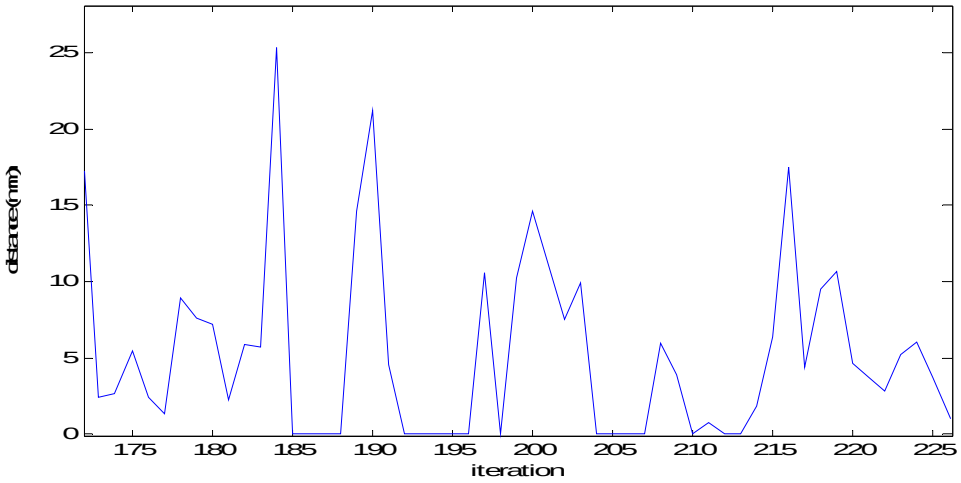

Figure 6.9: Accuracy of estimation compared to manual background truth.

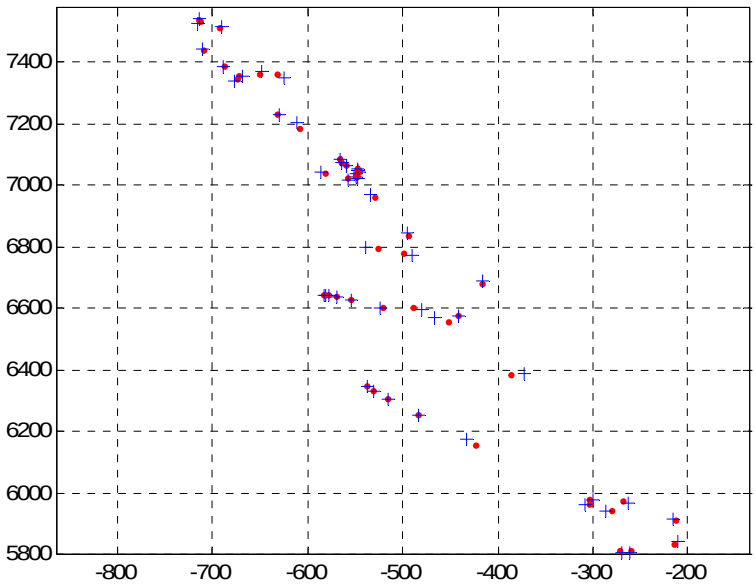

Figure 6.10: Estimates and centers of measures.

6.3 Parameter Influence

Different parameters of the Kalman filter and the association algorithm affect the tracking in different ways. In this chapter some parameters are manipulated in order to see how the tracking is affected. The tested parameters are the removal counter, the validation counter, the validation gate radius and the Kalman parameters Q and R. During the tests all parameters are kept constant except for the one being tested. The values of the parameters are the same as described in the beginning of the result chapter.

6.3.1 Removal and validation Counters

If the limit of the removal counter is too small, objects that are still present in the scene may be removed. One example of this is showed in Figure 6.11. As can be seen the input data associated to object nr 4 is poor and therefore measurements are missing during several iterations. Since measurements are missing the tracked object becomes a candidate for removal.

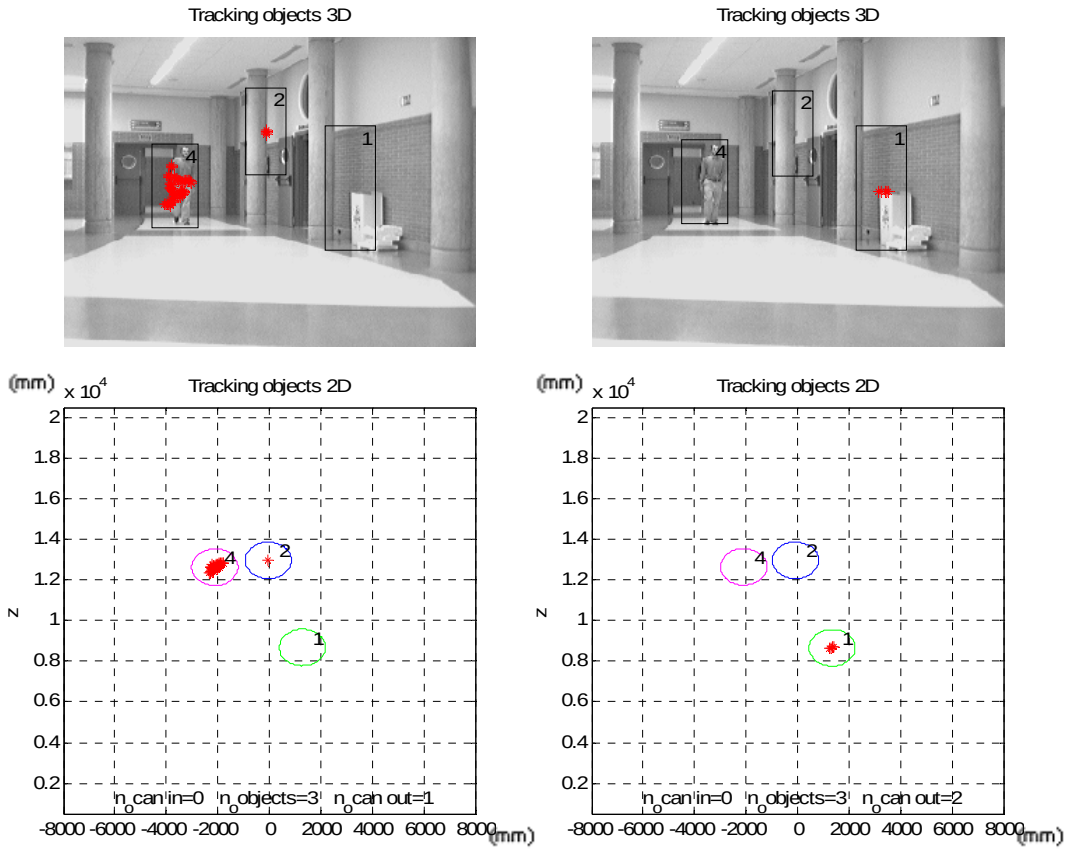

Figure 6.11: Candidate for removal.

In this experiment the limit of the removal counter is set to 10 iterations and in Figure 6.12 the counter reaches the limit. No new measurements have been associated with the target and therefore the track is removed. Then when measurements are associated with the object again a new tracker is started, and the object is applied tracking number 3 instead of the original 4. To avoid this, the number of iterations counted for removal should be increased.

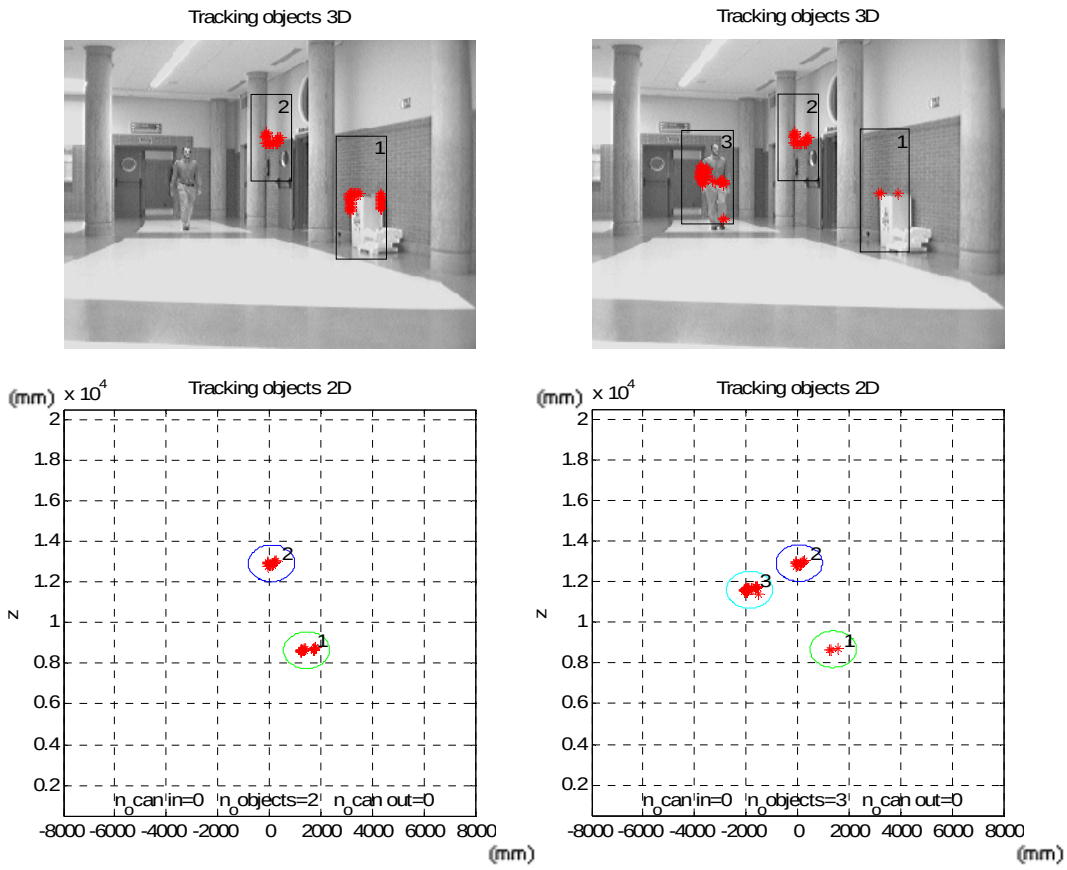

Figure 6.12: Removal of still existing object.

If the counting for removal is made longer it may lead to continued tracking of objects that is not present in the scene. If there are several objects in the scene, this could lead to a great number of objects being tracked, which is time consuming.

For the validation counter, if the counts are too few, clutter that is present over a number of iterations may be validated as an object. And if the number of iterations counted is too many, an object that is present in the scene a short amount of time may never be validated.

6.3.2 Validation Gate Radius

If the size of the validation gate radius is set too big, two objects may be seen as one. The right picture of Figure 6.13 gives an example of this; the radius is increased to 950mm and the two persons are being tracked as one single object. In the right picture the original radius of 850 mm is used and the two persons are being tracked correctly as two individual objects.

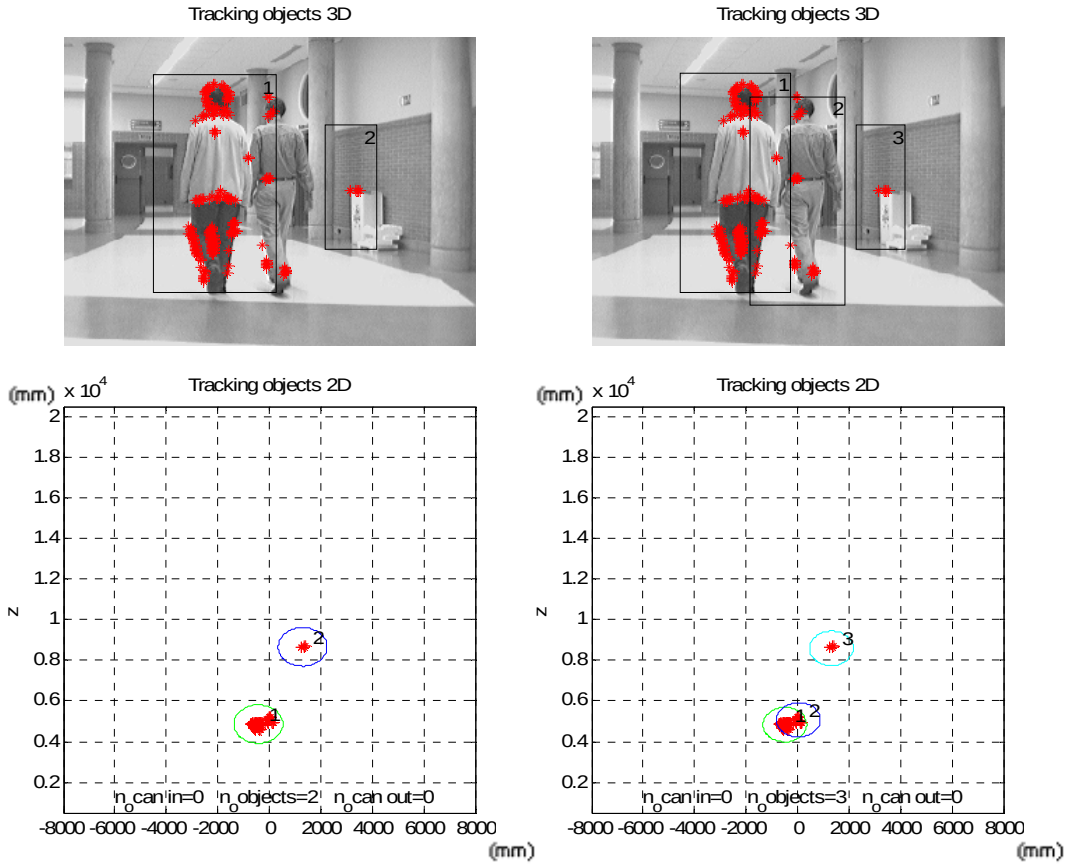

Figure 6.13: Validation gate radius.

The opposite of the example given above, when the radius is set too small the tracking of one single object may be divided into multiple trackers.

6.3.3 Kalman Covariance Matrices

The time it takes for the estimation error covariance to converge was discussed earlier. Other parameters that influence on the time it takes for the KF to converge are the measurement noise covariance matrix, R, and the state noise covariance matrix, Q.

1. When Q is made 100 times smaller than the original value used, the predicted error covariance decreases according to Equation 3.7, and therefore the estimation error covariance also decreases as it depends on the predicted value.

2. When R is made 100 times bigger than the original value used, the Kalman gain, K, decreases as can be seen in Equation 3.9. A small K leads to a big estimation error covariance according to Equation 3.11.

The effects of Q and R on the estimation error covariance can be seen in Table 6.1 and in Figure 6.14 where different values of Q and R affect the number of iterations before the filter converge and the final value of P. The results were obtained from a scene with a single object that is in a comfortable view of the cameras.

Table 6.1: The value of *P* in mm^2 depending on *Q* and *R*.

Iteration	1	2	3	4	5	6	7	8
Q=1 R=0.1	1	0.09524	0.09163	0.09161	0.09161	0.09161	0.09161	0.09161
Q=0.01 R=0.1	1	0.09099	0.05025	0.03760	0.03225	0.02970	0.02842	0.02775
Q=1 R=10	1	0.66667	0.62500	0.61905	0.61818	0.61806	0.61804	0.61803

In Figure 6.14 the black dashed line shows the convergence of P when running the KF with the original values used, Q=1 and R=0.1, the blue line is the result of increasing R 100 times and the red line is the result from decreasing Q 100 times.

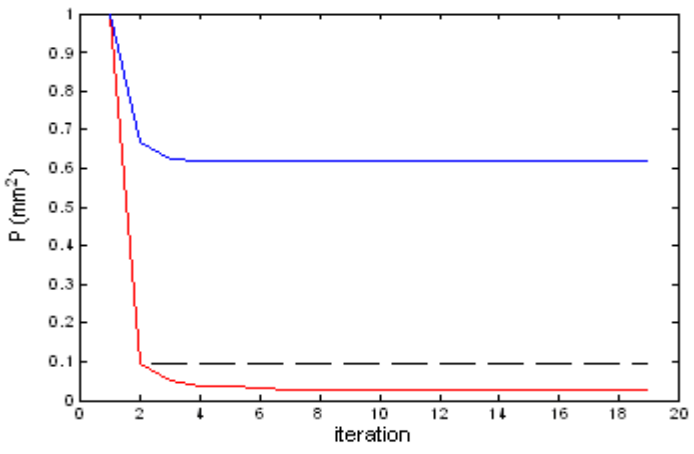

Figure 6.14: The value of *P* depending on *Q* and *R*.

If *R* is increased the measurements are believed to be noisy, *K* becomes smaller and the predicted states are trusted more. The filter becomes slower to respond to the measurements, which results in a reduced estimate variance. On the other hand if *R* is decreased, the measurements are believed to be good, *K* increases and the measurements are trusted more. The filter responds to the measurements more quickly, increasing the estimation variance.

R should be chosen so that the best performance is achieved in terms of balancing responsiveness and estimate variance. The original value of *R* used can be seen in Equation 5.2. To test the theory presented in the previous paragraph *R* was manipulated by increasing and decreasing the original value by a factor of 100. Figure 6.15 shows the tracking when *R* is 100 times greater and Figure 6.16 when *R* is 100 times smaller. The blue plus signs are the estimated states and the red dots are the centers of measures. As can be seen the filter is slower to believe the measurements in Figure 6.15 than in Figure 6.16, resulting in a smaller estimate variance.

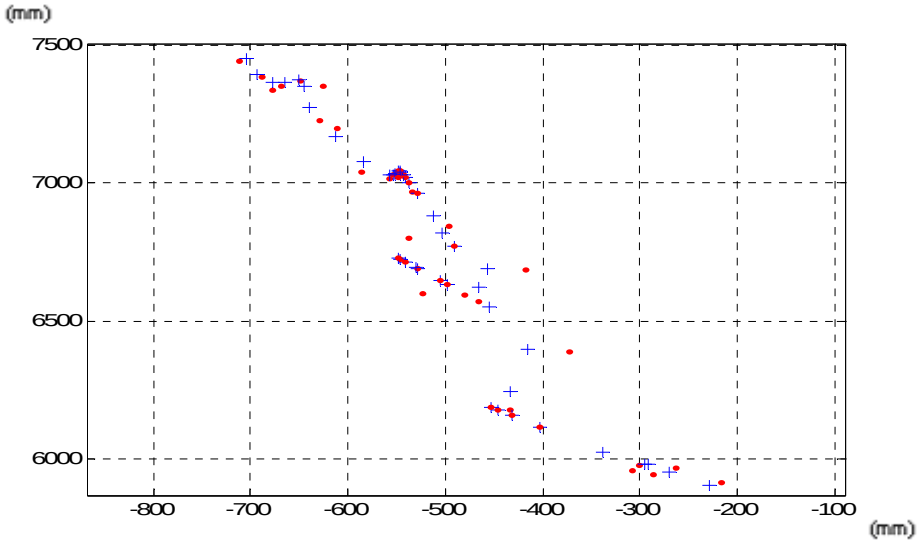

Figure 6.15: Estimates and centers of measures when R is 100 times greater.

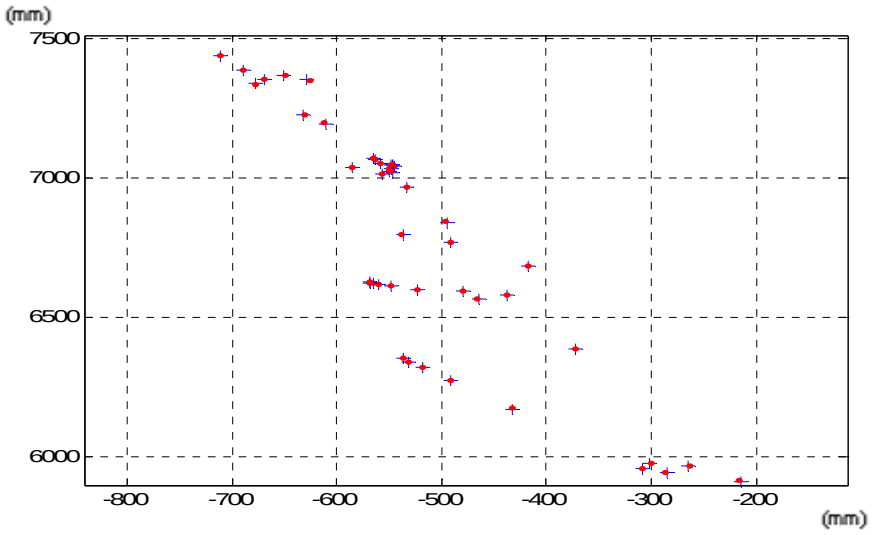

Figure 6.16: Estimates and centers of measures when R is 100 times smaller.

6.4 Object probability

The *object probability* plot informs about the probability of where to find an obstacle in the scene at each time step. The normalized probabilities in Figure 6.17 correspond to the measures in the xz-projection plot in Figure 6.18. As it can be seen in the *object probability* plot the probability is high in the positions where measures exist and zero otherwise.

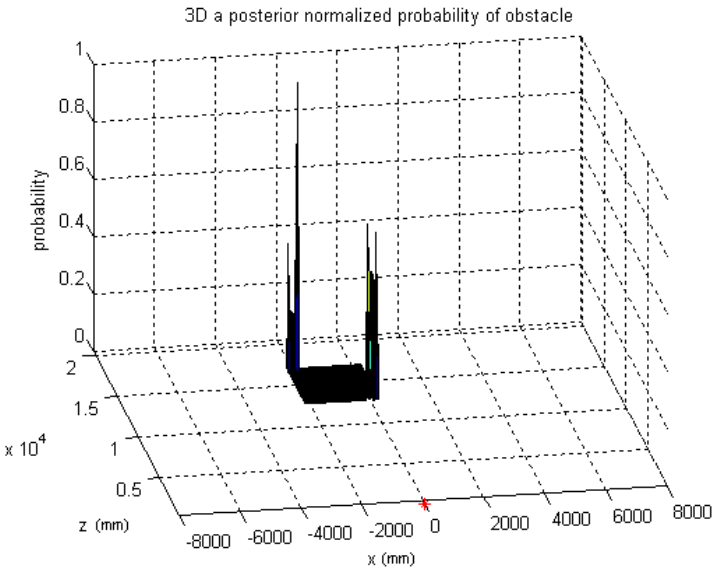

Figure 6.17: Object probability.

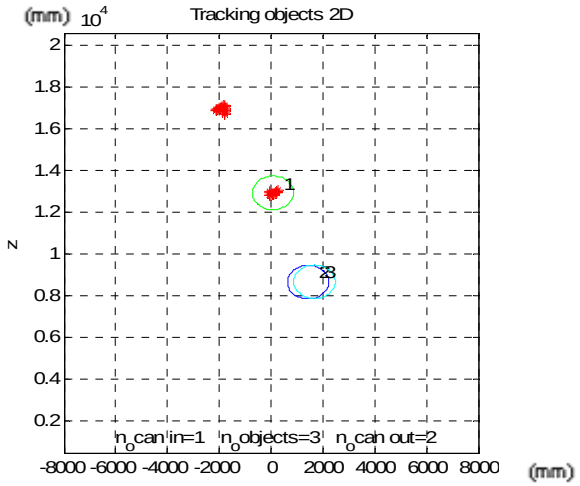

Figure 6.18: xz-projection plot.

7. Conclusions

The presented results shows that the objectives presented in the beginning of the thesis are fulfilled. The tracking of multiple objects is done with one Kalman filter for each target and the PDAF associates the measurements to different tracks by using the Euclidean distance, a validation gate and the measure probability.

Since the real background truth is unavailable the true accuracy of the tracking algorithm can not be obtained. But by comparing the estimated states to the manual background truth, it is showed that the implemented tracking algorithm is working correctly.

The tracking algorithm is capable of tracking a varying number of multiple objects including different solutions to the problems that can occur.

- The initiation of new tracks is working correctly; candidates are validated when new targets enter the scene and treated as false alarms when outliers appear.

- The problem with occlusion is also solved correctly; the tracks are continued during occlusion and when a target is absent for too long its track is removed.

- The crossing phenomenon is solved automatically by the developed Kalman model and the problem that arise when two objects move close to each other is solved by tuning of the validation gate radius.

Still the tracking algorithm is far from perfect and can be improved in many ways which is presented in the following section, future works. The accuracy of the tracking algorithm depends on the value of different parameters and the appearance of the scene. A parameter that is ideal in one situation may not work as well in another environment.

7.1 Future Works

As mentioned in the objectives, Chapter 2, it is possible to use an only estimator for all the objects instead of one Kalman filter for each track. This can be done by increase the size of the state vector so that it fits all objects. This state vector has to be dynamic since the number of objects is changing with time.

Also, since the process is linear and the noise is assumed to be Gaussian, Kalman filtering is used. In other situations another tracking algorithm might have to be used. For example if the process or the measurement relationship to the process is non-linear the Kalman filter can be replaced with an *extended Kalman filter*, EKF. This filter linearizes about the current mean and covariance [1].

The association algorithm can be improved by using a reassociation step. In the developed association algorithm, when a new track is created, only the not yet assigned measurements are compared to this track. This means that previously assigned measures may have been assigned to the wrong track. If a reassociation step is used, the distances from all measurements to all tracks are recalculated and if an already assigned measurement is closer to a new track, it is reassigned. The question is how many times the measures should be reassociated. The accuracy of the association algorithm has to be weighed against the preferred execution time.

Another approach on how to solve the association is presented in [13]. Here the association is done using a *'k-means'* clustering method. This segmentation algorithm increases the robustness of the estimation procedure in different points.

The Kalman filtering can be improved by using *velocity smoothing*, which is implemented in Part II [17]. A critical factor for the prediction step of the Kalman filter is the velocity. The right size and direction for this is crucial for a good approximation. In a real environment the size and direction of the velocity can change rapidly, which makes the tracking more difficult. This problem can be reduced by smoothing the velocity. This means that instead of using the current time steps velocity, the velocities from the last two time steps, or more, are summarized and then divided by the number of summarized velocities.

The parameters discussed in the algorithm (Q, R, P_0, validation gate radius and limits of the validation/removal counters) could be updated dynamically depending on different situations. For example the values in the Q matrix could be updated with change in dynamics and the validation gate radius could change with different sizes of objects.

III. USER MANUAL

In this section all the information needed to test the tracking algorithm developed in this thesis is described. To execute the implemented tracking algorithm some of the specifications in Section IV are necessary: a PC installed with Windows XP and Matlab 6.5.

1. Necessary files

To run the Matlab implementation the following m-files are needed:

- main.m
- constants.m
- initiate.m
- association.m
- kman.m
- visualize.m
- trans.m

The input data is stored in the following *.dat*-files and to show the video, the corresponding *.str*-files are necessary:

- data001.dat, left001.str
- data004.dat, left004.str
- data009.dat, left009.str
- data010.dat, left010.str
- data041.dat, left041.str
- data061.dat, left061.str

2. How to execute the program

The following steps explain how to execute the program. All the files in the previous chapter should be saved in the same folder.

1. Open Matlab 6.5
2. In the field "Current Directory" choose the folder that contains the files.
3. Choose file>>open, and open the main.m file
4. To execute the program choose "Run" or press F5.

3. How to set/change the parameters

To set or change the following parameters locate the desired parameter in its corresponding file:

- The input files can be changed in the *main.m* file by changing the *.dat* and *.str* file that is read by *fopen*.
- The speed of the program can be slowed down by changing the parameter *pause()* in *main.m*
- The Kalman prediction and correction matrices can be found in *constants.m*. The matrices *G* and *H* are represented by *A* and *B,* and the matrix *C* by *H*.
- The process noise matrix can be adjusted in *constants.m* by changing the *Q* and the measurement noise matrix can be adjusted in *kman.m* by changing the *R*.
- The initial value of the estimation error covariance matrix P_0 can be changed by manipulating *P_corrected* in *initiate.m*.
- The time steps counted by the validation and removal counters are set by *VAL* and *DEL* in *constants.m*.
- The size of the validation gate is determined by the parameter *RADIUS* in *constants.m*.

4. Monitoring

The outputs are plotted in Figure No.1 that contains both the video image and the 2D projection plot. The object probability is plotted in Figure No.2. The number of objects and the number of candidates for validation and removal are shown at the bottom of the projection plot in Figure No.1.

5. Definitions of the functions

```
%========================================================================
%   MAIN.M - TRACKING OF MULTIPLE OBJECTS
%
%   function [] = main()
%
%   global FILE_SIZE DEL
%
%   Tracking multiple objects in a scene using Kalman filters and a
%   probabilistic data association filter.
%
%   GLOBAL PARAMETERS
%   FILE_SIZE:          Contains the size of the input video image (width&height)
%   DEL:                Number of time steps for the removal counter
%========================================================================



%========================================================================
%   CONSTANTS.M - DEFINING CONSTANTS
%
%   function [] = constants()
%
%   Defining global constants:
%   Kalman matrixes, coordinate transformation parameters,
%   image parameters, axis coordinates, colors of the tracks,
%   validation and removal counting time
%========================================================================



%========================================================================
%   INITIATE.M - INITIATE TRACKS AND PREDICT STATE
%
%   function [initialized, state_corrected, state_predicted, P_predicted,
FLAG_empty,...
%   FLAG_new, length_, probability, y] = initiate(data, ym, length_)
%
%   global A B Q RADIUS;
%
%   Initiate tracks from measures. The first measure is defined as a candidate
%   track. For the rest of the measures the distances to the existing
%   candidate tracks are computed. If the distance is bigger than the
%   validation gate, RADIUS, a new candidate track is created, otherwise
%   the measure is assigned to the closest candidate.
%
%   Then the state is predicted.
%
%
%   GLOBAL PARAMETERS:
%   A, B:               Kalman matrixes to predict the state
%   Q:                  Process noise matrix
%   RADIUS:             Validation gate radius
%
%   INPUT AND OUTPUT PARAMETERS:
%   initialized:        Flag for telling if tracks have to be initiated
%   state_corrected:    Array of corrected states (x and z)
%   state_predicted:    Array of predicted states (x and z)
%   P_predicted:        The predicted estimation error covariance matrix
%   FLAG_empty:         Removal counter
%   FLAG_new:           Validation counter
%   length_:            Length of array with xz-states
%   probability:        Each measures probability based on the distance
%   y:                  Mean of the y-coordinates associated to each target
%   data:               Containes the input x and z coordinates of the measures
%   ym:                 Contains the input y coordinates of the measures
%========================================================================



%========================================================================
%   ASSOCIATION.M - ASSOCIATION ALGORITHM
%
```

```
%   function [centers_measures, length_, n_objects, state_predicted, FLAG_empty,
%   FLAG_new, probability, y, v] =association(data,state_predicted, length_,
%   n_objects, FLAG_empty, FLAG_new, c3)
%
%   global RADIUS DEL VAL
%
%   Association of measures to tracks. The distance to each existing track
%   is calculated. If it is bigger than the validation gate, RADIUS, a new
%   track is created, otherwise it is assigned to the closest track.
%
%   After association, validation and removal counters are checked and the
%   innovation and centers of measures are calculated.
%
%
%   GLOBAL PARAMETERS:
%   RADIUS:             Validation gate radius
%   DEL:                Number of time steps for the removal counter
%   VAL:                Number of time steps for the validation counter
%
%   INPUT AND OUTPUT PARAMETERS:
%   centers_measures:   Mean of each track's associated measures
%   length_:            Length of array with xz-states
%   n_objects:          Number of tracks
%   state_predicted:    Array of predicted states (x and z)
%   FLAG_empty:         Removal counter
%   FLAG_new:           Validation counter
%   probability:        Each measures probability based on the distance
%   y:                  Mean of the y-coordinates associated to each target
%   v:                  Combined innovation
%   data:               Containes the input x and z coordinates of the measures
%   c3:                 Contains the input y coordinates of the measures
%=========================================================================



%=========================================================================
%   KMAN.M - KALMAN FILTERING
%
%   function [state_predicted, P_predicted, state_corrected,P_corrected] =
%   kman(centers_measures, state_predicted, P_predicted,
length_,state_corrected_old,v)
%
%   global A B H Q T;
%
%   One Kalman filter is used for each target to track. The correction of the
%   states is done first, then the velocities are updated and finally the
%   states of the next time step is predicted.
%
%
%   GLOBAL PARAMETERS:
%   A, B, H:            Kalman matrixes to estimate the state
%   Q:                  Process noise matrix
%   T:                  Sample time
%
%   INPUT AND OUTPUT PARAMETERS:
%   state_predicted:    Array of predicted states (x and z)
%   P_predicted:        The predicted estimation error covariance matrix
%   state_corrected:    Array of corrected states  (x and z)
%   P_corrected:        The corrected estimation error covariance matrix
%   centers_measures:   Mean of each track's associated measures
%   length_:            Length of array with xz-states
%   state_corrected_old:Array of corrected states from previous time step
%   v:                  Combined innovation
%=========================================================================



%=========================================================================
%   VISUALIZE.M - PLOTS THE MEASURES, TRACKS AND OBJECT PROBABILITY
%
%   function [] = visualize(data, state_corrected, FLAG_empty, FLAG_new, length_,
%       probability, n_objects)
%
%   global X_MIN X_MAX Z_MIN Z_MAX Y_MIN Y_MAX COLORS RADIUS DEL;
%
%   Plotting of the measures and the circles of the corrected states in the
%   projection plane. The number of candidates for validation or removal
```

```
%   are calculated and displayed in the same plot together with the actual
%   number of tracks. Another plot shows the normalized probability of
%   finding an object in the scene.
%
%
%   GLOBAL PARAMETERS:
%   X_MIN, X_MAX:       Size of x-axis
%   Z_MIN Z_MAX:        Size of z-axis
%   Y_MIN Y_MAX:        Size of y-axis
%   COLORS:             Colors of circles
%   RADIUS:             Validation gate radius
%   DEL:                Number of time steps for the removal counter
%
%   INPUT PARAMETERS:
%   data:               Containes the input x and z coordinates of the measures
%   state_corrected:    Array of corrected states  (x and z)
%   FLAG_empty:         Removal counter
%   FLAG_new:           Validation counter
%   length_:            Length of array with xz-states
%   probability:        Each measures probability based on the distance
%   n_objects           Number of tracks
%=========================================================================



%=========================================================================
%   TRANSF.M - TRANSFORMATION FROM CARTESIAN COORDINATES TO IMAGE COORDINATES
%
%   function transf(c, I, state_corrected, length, FLAG_empty, FLAG_new, hight)
%
%   global FXL FYL U0L V0L TRAS_Y ROTATION_SCC RADIUS
%
%   Shows the video and transforms the measures into image coordinates. The
%   rectangles that represents the tracks are calculated and then transformed.
%   Then the measures and rectangles are plotted in the video image.
%
%
%   GLOBAL PARAMETERS:
%   FXL, FYL, U0L, V0L: Intrinsic transformation parameters
%   TRAS_Y:             Extrinsic translation matrix
%   ROTATION_SCC:       Extrinsic rotation matrix
%   RADIUS:             Validation gate radius
%
%   INPUT PARAMETERS:
%   c                   Input measures
%   I                   Input video image
%   state_corrected:    Array of corrected states  (x and z)
%   length:             Length of array with xz-states
%   FLAG_empty:         Removal counter
%   FLAG_new:           Validation counter
%   hight:              Means of the y-coordinates
%=========================================================================
```

IV. SPECIFICATIONS

In this section the hardware and software used for the implementation and simulations presented in this project are presented.

1. Hardware

- **Packard Bell Notebook**

Microprocessor	AMD Turion 64
Speed	1,58 GHz
RAM	896 MB RAM
Disk	80 Gb
Monitor	17” LCD

- **Printer Xerox Document Center 340**

Printer type	Laser
Speed	40 ppm
Resolution	600 dpi
Communications	Standard TCP/IP

2. Software

- **Operating Systems**
  Windows XP Professional Version 2002, SP2

- **Office 2003 (English)**
  Microsoft Office Professional Edition 2003, version 11.5604.5606.

- **Matlab**
  Matlab 6.5.0.180913a, Release 13

- **Antivirus Systems**
  Panda Titanium Antivirus 2005, version 4.02.01.
  Norton Antivirus 2005, version 11.0.11.4

V. PROGRAM CODE

1. main.m

```
function [] = main(file)
clear all;
tic;

%CALL FUNCTION TO DEFINE CONSTANTS
constants;

global FILE_SIZE DEL;


%============================INITIALIZATION=================================
%Number of objects.
n_objects=0;
length_=0;
%flag for telling if initial objects have been found in empty scen
initialized=0;
%frame number
frame=1;


%=========================OPEN FILE TO READ================================

pf=fopen('left004.str');
fid=fopen('data004.dat');
ne=fread(fid,1,'int32');

while isempty(ne)==0 && feof(pf)==0,
    I=fread(pf,FILE_SIZE);
    c=fread(fid,[3 ne],'float32');
    data=c(1:2,:)';



    %=====================================================================
    %   IF EMPTY SCENE FIND OBJECTS & PREDICT FIRST POSITION
    %=====================================================================
```

```
    %if frame contains measurements and no object has been found yet
    if ne~=0 && initialized==0;

        [initialized, state_corrected, state_predicted, P_predicted, FLAG_empty,...
                FLAG_new, length_, probability, y]=initiate(data, c(3,:), length_);


    %======================================================================
    %   ASSOCIATION & KALMAN FILTERING
    %======================================================================

    else
        if initialized==1
            y_old=y;


            %==================ASSOCIATION FUNCTION=======================

            [centers_measures, length_, n_objects, state_predicted,...
                    FLAG_empty, FLAG_new, probability, y, v] =...
                association(data, state_predicted, length_, n_objects,...
                    FLAG_empty, FLAG_new, c(3,:));

            n=1;
            for m=1:2:2*length_
                if FLAG_new(n)==2 && FLAG_empty(n)==0
                    state_corrected(1,m:(m+1))=centers_measures(1,m:(m+1));
                end
                n=n+1;
            end


            %====================KALMAN FUNCTION==========================
            if length_ ~=0
                [state_predicted,P_predicted,state_corrected, P_corrected]=...
                    kman(centers_measures, state_predicted,P_predicted,length_,...
                    state_corrected,v);
            end

            for i=1:size(FLAG_empty,2)
                if FLAG_empty(i)<DEL && FLAG_empty(i)~=0
                    y(i)=y_old(i);
                end
            end
        end

    end


    %======IF SCENE EMPY THEN CLEAR & NEXT ITER FIND OBJECTS IN SCENE========

         if n_objects==0 && length_==0
        clear state_corrected;
        clear dist_to_obj;
        clear state_predicted;
        P_predicted=1*eye(2);
        initialized=0;
    else


        %======================================================================
        %   VISUALIZATION OF DATA
        %======================================================================

        visualize(data, state_corrected, FLAG_empty, FLAG_new, length_, ...
            probability, n_objects);

        transf(c, I, state_corrected, length_, FLAG_empty, ...
            FLAG_new, y);
    end


     pause(0.05);
%    MAKE MOVIE
%    film_mov(frame) = getframe;
```

```
    %UPDATE FRAME
    frame=frame+1;
    ne=fread(fid,1,'int32');
end;

%create an avi movie that is saved in the current map
% movie2avi(film_mov, 'video010');

fclose(fid);
fclose(pf);
```

2. constants.m

```
function [] = constants()

%================DEFINITION OF CONSTANTS=========================

%KALMAN PARAMETRS
global A B H Q;
A=eye(2);
B=eye(2);
H=eye(2);
Q=1*eye(2);

%TIME
global T;
T=0.066;

%IMAGE PARAMETERS
global WIDTH HIGHT FILE_SIZE;
WIDTH=320;
HIGHT=240;
FILE_SIZE=[WIDTH HIGHT];

%INTRINSIC PARAMETERS, LEFT CAMERA
global FXL FYL U0L V0L;
FXL = 430.79014; FYL = 431.72027;
U0L = 151.26555; V0L = 117.03242;

%CAMERA ROTATION ANGLES
alpha = 0.94972*3.14159/180;
beta = 0.019508;
phi = -0.014053;

%TRANSLATION OF Y-AXIS
global TRAS_Y;
TRAS_Y=970;

%ROTATION MATRIX
global ROTATION_SCC;
ROTATION_SCC=...
        [cos(beta)*cos(phi) -cos(beta)*sin(phi) sin(beta);...
        sin(alpha)*sin(beta)*cos(phi)+cos(alpha)*sin(phi)...
        sin(alpha)*sin(beta)*sin(phi)+cos(alpha)*cos(phi)...
        -sin(alpha)*cos(beta);...
        -cos(alpha)*sin(beta)*cos(phi)+sin(alpha)*sin(phi)...
        cos(alpha)*sin(beta)*sin(phi)+sin(alpha)*cos(phi)...
        cos(alpha)*cos(beta)];

%AXIS COORDINATES
global X_MIN X_MAX Z_MIN Z_MAX Y_MIN Y_MAX;
X_MIN = -8000;
X_MAX = 8000;
Z_MIN = 500;
Z_MAX = 20500;
Y_MIN = 100;
Y_MAX = 2100;

%DEF FOR THE OBJECTS
global COLORS RADIUS DEL VAL;
COLORS = {'g' 'b' 'c' 'm' 'k' 'y' 'g' 'b' 'c' 'm' 'k' 'y'};
RADIUS=800;
DEL=10;
VAL=4;
```

3. initiate.m

```
function [initialized, state_corrected, state_predicted, P_predicted, FLAG_empty,...
        FLAG_new, length_, probability, y] = initiate(data, ym, length_)

global A B Q RADIUS;

    length_ = length_ + 1;

    %save the measure's position as object center
    centers(length_:length_+1) = data(1,:);

    %data_org contains the measures associated to each object
    for m = [1:length_]
        data_organized{m} = [];
        y_org{m}=[];
    end


    %=========================================================================
    %   ASSIGNMENT
    %=========================================================================

    for m = [1:size(data,1)]

        %calculate the distance from each measure to each object
        n=1;
        for i = [1:2:2*length_]
            dist_to_obj(:,n) =...
                sqrt((data(:,1)-centers(:,i)).^2 +...
                (data(:,2)-centers(:,i+1)).^2);
            n=n+1;
        end

        %find the shortest distance
        minimum = min(dist_to_obj(m,:));
        %find the column that contains the shortest distance
        column = find(dist_to_obj(m,:) == minimum);
        probability(m,1)=exp(-dist_to_obj(m,column)/1000);


        %===============ADD MEASURE TO EXISTING OBJECT=================
        if dist_to_obj(m,column)<RADIUS
            data_organized{column} = [data_organized{column};...
                    data(m,:) dist_to_obj(m,column)];
            y_org{column} = [y_org{column}; ym(m)];
            centers(1,2*column-1:2*column) =...
                mean(data_organized{column}(:,1:2),1);

        %=================DEFINE MEASURE AS NEW OBJECT===============
        else    %otherwise define new object
            length_ = length_ + 1;
            %saves the measure's position as object center
            centers(2*length_-1:2*length_) = data(m,:);
            data_organized{length_} = [data(m,:) dist_to_obj(m,column)];
            y_org{length_} = ym(m);
        end
    end

    %calculate mean of y-coordinates assigned to each object
    for n=1:length_
            y(n)=mean(y_org{n});
    end


    %======================INITIALIZATION================================

    velocities=zeros(2,length_); %initial velocities is set to zero
    state_corrected=centers;
    initialized=1; %init obj has been found >> go to Kalman filtering
    FLAG_empty=zeros(1, length_); %flag that inicates if remove obj
    FLAG_new=2*ones(1,length_); %flag that indicates if obj validated
    P_corrected=1*eye(2); %initial error_cov
```

```
%=========================================================================
%   PREDICTION t=0
%=========================================================================

n=1;
for m=1:2:2*length_
    X_corrected(1,1)=state_corrected(m);
    X_corrected(2,1)=state_corrected(m+1);
    V=velocities(:,n);

    X_predicted=A*[X_corrected]+B*V;
    P_predicted=A*P_corrected*A'+Q;

    state_predicted(m) = X_predicted(1,1);
    state_predicted(m+1) = X_predicted(2,1);
    n=n+1;
end
```

4. association.m

```
function [centers_measures, length_, n_objects, state_predicted, FLAG_empty,
FLAG_new, probability, y, v] =...
    association(data,state_predicted, length_, n_objects, FLAG_empty, FLAG_new, c3)

global RADIUS DEL VAL


%=========================================================================
%   CALCULATION OF DISTANCE
%=========================================================================

for m = [1:length_]
    data_organized{m} = [];
    y_org{m}=[];
end
probability=[];
y=[];

if isempty(data)==0

    n=1;
    for m = [1:2:2*length_]
        distance(:,n) =...
            sqrt((data(:,1)-state_predicted(:,m)).^2 +...
            (data(:,2)-state_predicted(:,m+1)).^2);
        n=n+1;
    end


    %=========================================================================
    %   ASSIGNMENT
    %=========================================================================


    for m = [1:size(data,1)]

        %finds the shortest distance
        minimum = min(distance(m,:));
        %finds the column that contains the shortest distance
        column =find(distance(m,:) == minimum);
        %calculates the probability that a measure belongs to a certain object
        probability(m,1)=exp(-distance(m,column)/1000);


        %==============ADD MEASURE TO EXISTING OBJECT=========================
        %if distance is close to an existing object then associate
        if distance(m,column)<RADIUS
            data_organized{column} = [data_organized{column}; data(m,:)...
                    distance(m,column)/1000];
            y_org{column} = [y_org{column}; c3(m)];
            %if objects was created in this iteration calc new mean & distance
            if FLAG_new(column)==1
                state_predicted(2*column-1:2*column)=...
                    mean(data_organized{column}(:,1:2),1);
                distance(:,column)=sqrt((data(:,1)-state_predicted(:,2*column-1)).^2
+...
                    (data(:,2)-state_predicted(:,2*column)).^2);
            end


        %==================DEFINE MEASURE AS NEW OBJECT=======================
        else
            %find empty place "i" in "data_organized"
            i=1;
            while FLAG_empty(i)<DEL
                i=i+1;
                if i>length_ break
                end
            end

            data_organized{i}=[];
```

```
            data_organized{i}=[data(m,:) distance(m,column)/1000];
            y_org{i}=[];
            y_org{i} = [c3(m)];
            FLAG_new(i)=1;
            FLAG_empty(i)=0;
            state_predicted(2*i-1:2*i)=data(m,:);
            %if data placed in new element in data_organized then increase length_
            %if data placed in existing empty place, keep length_
            if i>length_
                length_ = length_ + 1;
            end
            distance(:,i) =sqrt((data(:,1)-data(m,1)).^2 + (data(:,2)-
data(m,2)).^2);
        end
    end

end


%=========================================================================
%   CALCULATE CENTERS OF ORGANIZED DATA & INNOVATION
%   & VALIDATE or REMOVE OBJECTS
%=========================================================================

n=1;
for m = [1:2:2*length_]

    %IF NO MEASURES ASSOCIATED WITH OBJECT
    if isempty(data_organized{n})==1
        if FLAG_empty(n)<(DEL+1)
            FLAG_empty(n)=FLAG_empty(n)+1;
        end

        %decrease number of objects (only if validated & gone for 10 frames)
        if FLAG_empty(n)==DEL
            if FLAG_new(n)==0
                n_objects=n_objects-1;
            end
            data_organized{n}=[];
            state_predicted(1,m:m+1)=[0 0];
            FLAG_new(n)=0;
        end

        centers_measures(1,m:(m+1)) = state_predicted(1,m:(m+1));
        v(1:2,n)=[0;0]; %no measures, gives probability=0, innovation=0


    %IF MEASURES ASSOCIATED WITH OBJECT
    else %calc center of measures
        centers_measures(1,m:(m+1)) = mean(data_organized{n}(:,1:2),1);
        y(n)=mean(y_org{n});
        FLAG_empty(n)=0;

        %if candidate for 4 frames then validate and inc. no objects
        if FLAG_new(n)==VAL
            n_objects = n_objects + 1;
            FLAG_new(n)=0;
        end

        %keeps track on the number of frames present but not valid
        if FLAG_new(n)~=0
            FLAG_new(n)=FLAG_new(n)+1;
        end

        %combined innovation
        data_organized{n}(:,3)=normpdf((data_organized{n}(:,3)),0,1)/...
            (sum(normpdf((data_organized{n}(:,3)),0,1)));
        for m=1:length(data_organized{n}(:,3))
            v_m(1,m)=data_organized{n}(m,3)*...
                (data_organized{n}(m,1)-state_predicted(2*n-1));
            v_m(2,m)=data_organized{n}(m,3)*...
                (data_organized{n}(m,2)-state_predicted(2*n));
        end

        v(1:2,n)=[sum(v_m(1,:));sum(v_m(2,:))];
    end
```

```
    n = n + 1;
end


%decrease length_ if object at the end hasn't existed last 10 iterations
for i=length_:-1:1
    if FLAG_empty(i)>=DEL && i==length_
        length_=length_-1;
        FLAG_empty=FLAG_empty(1:i-1);
        FLAG_new=FLAG_new(1:i-1);
    end
end
```

5. kman.m

```
function [state_predicted, P_predicted, state_corrected, P_corrected] =...
   kman(centers_measures, state_predicted, P_predicted, length_,
state_corrected_old, v)

global A B H Q T;


%=========================================================================
%   CORRECTION
%=========================================================================
n=1;
% P_corrected=1*eye(2);
for m=1:2:2*(length_)
        X_predicted(1,1)=state_predicted(m);
        X_predicted(2,1)=state_predicted(m+1);
        R=0.1*eye(2);

        K=P_predicted*H'/(H*P_predicted*H'+R);
        X_corrected = X_predicted + K*v(:,n);
        P_corrected=(eye(2)-K*H)*P_predicted;
        state_corrected(m) = X_corrected(1,1);
        state_corrected(m+1) = X_corrected(2,1);
        n=n+1;
end


%=========================================================================
%   UPDATE VELOCITIES
%=========================================================================
velocities_x=[]; velocities_z=[]; velocities=[];

n=1;
t=toc;
while t<0.066
         t=toc;
end
for m=1:2:2*length_
            velocities_x(n)= (centers_measures(m)-state_corrected_old(m))/t;
            velocities_z(n)= (centers_measures(m+1)-state_corrected_old(m+1))/t;
            velocities=[velocities_x; velocities_z];
            n=n+1;
end
tic;

%=========================================================================
%   PREDICTION
%=========================================================================
clear state_predicted;
clear P_predicted;

n=1;
for m=1:2:2*length_
    X_corrected(1,1)=state_corrected(m);
    X_corrected(2,1)=state_corrected(m+1);
    V=velocities(:,n);

    X_predicted=A*[X_corrected]+B*V*T;
    P_predicted=A*P_corrected*A'+Q;
    state_predicted(m) = X_predicted(1,1);
    state_predicted(m+1) = X_predicted(2,1);
    n=n+1;
end
```

6. visualize.m

```
function [] = visualize(data, state_corrected, FLAG_empty, FLAG_new, length_,
probability, n_objects)

global X_MIN X_MAX Z_MIN Z_MAX Y_MIN Y_MAX COLORS RADIUS DEL;


%============================CANDIDATES=====================================
can_in=0;
can_out=0;

for i=1:length_
    if FLAG_new(i)~=0 && FLAG_empty(i)==0
        can_in=can_in+1;
    end
    if FLAG_empty(i)>0 && FLAG_empty(i)<DEL && FLAG_new(i)==0
        can_out=can_out+1;
    end
end
str=sprintf('n_ocan in=%d   n_oobjects=%d   n_ocan out=%d',...
    can_in, n_objects,  can_out);



%============================================================================
%   2D-PLOT
%============================================================================

%============PLOT THE MEASURES==============
figure(1);
clf(1);
subplot('position',[0.12 0.03 0.84 0.47])
plot(data(:,1),data(:,2),'r*');
axis([X_MIN X_MAX Z_MIN Z_MAX]);
xlabel('x'); ylabel('z');
title('Tracking objects 2D');
grid;
hold on

%===========PLOT THE CIRCLES================
k = 1;
theta = linspace(0,2*pi,100);
for m = [1:2:2*length_]
    if (state_corrected(m)&&state_corrected(m+1))~=0 &&...
            FLAG_empty(k)<DEL && FLAG_new(k)==0
        x = RADIUS*cos(theta)+state_corrected(m);   % generate x-coordinate
        z = RADIUS*sin(theta)+state_corrected(m+1); % generate y-coordinate
        plot(x,z, COLORS{k});
        number=sprintf('%d',(m+1)/2);
        text(state_corrected(m)+400,state_corrected(m+1)+400,number);
    end
    k = k + 1;
end


%============================================================================
%   PROBABILITY PLOT
%============================================================================

[hx,vx]=hist(data(:,1),50);
[hz,vz]=hist(data(:,2),50);

if ~isempty(data)
    figure(2);
    clf(2);
    vz(1)=min(data(:,2)); vz(end)=max(data(:,2)); vz(end+1)=Inf;
    vx(1)=min(data(:,1)); vx(end)=max(data(:,1)); vx(end+1)=Inf;
    density=zeros(50,50);
    resta=[];
    for i=1:50
        indexx=find((data(:,1)>=vx(i))&(data(:,1)<min([vx(i+1) max(vx)])));
        if ~isempty(indexx)
```

```
            for j=1:length(indexx)
                resta=abs(vz-data(indexx(j),2));
                [foo,indexz]=min(resta);
                density(i,indexz)=density(i,indexz)+probability(indexx(j));
            end;
        end;
    end;
    foo=max(max(density));
    if foo~=0
        density=density/foo;
        surf(vx(1:end-1), vz(1:end-1), density');
        xlabel('x'); ylabel('z'); zlabel('probability');
        title('3D a posterior normalized probability of obstacle')
        hold on;
        plot3(0,0,0,'r*');
        view([-10 30]);
        axis([X_MIN,X_MAX,Z_MIN,Z_MAX,0,1]);
        hold off;
    else
        disp('w total is zero');
    end;
end
```

7. transf.m

```
function transf(c, I, state_corrected, length, FLAG_empty, FLAG_new, hight)

global FXL FYL U0L V0L TRAS_Y ROTATION_SCC RADIUS


y=TRAS_Y-c(3,:);
c(3,:)=c(2,:);
c(2,:)=y;

%==================SHOW THE FILM=====================
figure(1);
subplot('position',[0.12 0.5 0.84 0.5])
imshow(uint8((I')));
hold on;


%==============TRANSFORM THE MEASURES================
SCCmatr=ROTATION_SCC*c;
U=round(FXL*(SCCmatr(1,:)./SCCmatr(3,:)) + U0L);
V=round(FYL*(SCCmatr(2,:)./SCCmatr(3,:)) + V0L);

%===============PLOT THE MEASURES====================
plot(U,V,'r*');
axis([0 320 0 240]);
title('Tracking objects 3D');


%========CREATE THE CORNERS OF THE RECTANGLES========
n=1;
for m=1:2:2*length
    rect1(1,n) = state_corrected(m)-RADIUS/2;
            rect2(1,n) = state_corrected(m)+RADIUS/2;
    rect1(3,n) = state_corrected(m+1)-RADIUS/2;
            rect2(3,n) = state_corrected(m+1)+RADIUS/2;
    rect1(2,n) = -hight(n);
            rect2(2,n) = 2*TRAS_Y-hight(n);
    n=n+1;
end

%=========TRANSFORM THE CORNERS OF THE RECTANGLES====
SCC_mean=ROTATION_SCC*rect1;
U_mean=round(FXL*(SCC_mean(1,:)./SCC_mean(3,:)) +U0L);
V_mean=240-round(FYL*(SCC_mean(2,:)./SCC_mean(3,:)) +V0L);

SCC_1=ROTATION_SCC*rect1;
U1=round(FXL*(SCC_1(1,:)./SCC_1(3,:)) +U0L);
V1=round(FYL*(SCC_1(2,:)./SCC_1(3,:)) +V0L);

SCC_2=ROTATION_SCC*rect2;
U2=round(FXL*(SCC_2(1,:)./SCC_2(3,:)) +U0L);
V2=round(FYL*(SCC_2(2,:)./SCC_2(3,:)) +V0L);

U_width=U2-U1;
V_width=V1-V2;

%==============PLOT THE RECTANGLES==================
n=1;
for m=1:2:2*length
    if (state_corrected(m)&&state_corrected(m+1))~=0 && FLAG_empty(n)<10 &&
FLAG_new(n)==0
        rectangle('Position',[U1(n),V1(n),abs(U_width(n)),abs(V_width(n))])
        number=sprintf('%d',(m+1)/2);
        text(U2(n)-10,V1(n)+10,number);
    end
    n=n+1;
end
```

VI. BUDGET

In this section different costs for the project are described and summed together.

1. Cost for laboratory equipment

Items	Quantity	Total
PC	1	920 €
Matlab 6.5	1	600 €
Windows XP	1	200 €
Microsoft Office	1	400 €
Printer	1	100 €
Office Material	-	50 €

Total cost for laboratory equipment…….:	**2270.00 €**

2. Cost for manual work

Function	Number of hours	€/h	Total
Engineering	800	60.00	48000 €
Writing	100	12.00	1200 €

Total cost for manual work…….:	**49200 €**

3. Total cost for project implementation

Concept	Total
Cost for laboratory equipment	2270.00 €
Cost for manual work	49200.00 €

Total cost for project implementation…….:	**51470.00 €**

4. Industrial benefit

For industrial benefits, 20 % of the total cost for project implementation is applied.

Total cost of industrial benefits…….:	**10294.00 €**

5. Budget for fulfillment of the contract

Concept	Total
Cost for project implementation	51470.00 €
Cost for industrial benefits	10294.00 €

Total cost for fulfillment of the contract…….:	**61764.00 €**

6. Writing honorary

For the writing honorary you calculate 7% of the total cost for project implementation.

Writing honorary…….:	**3602.90 €**

7. Total amount of the budget

Concept	Total
Cost of project	61764.00 €
Writing honorary	3602.90 €
Total	65366.90 €
16% of VAT	10458.70 €

TOTAL AMOUNT…….:	**75825.60 €**

VII. REFERENCES

[1] G. Welch, G. Bishop, "An Introduction to the Kalman Filter", ACM, Inc. 2001
http://www.cs.unc.edu/~welch/

[2] Visual Tracking: http://www.cis.fordham.edu/~lyons/rcv/hmk.html

[3] L. Y. Pao, R. M. Powers, "A comparison of Several Different Approaches for Target Tracking With Clutter", Department of Colorado at Boulder, USA, 2002

[4] C. Bibby, "Fast Detection and tracking of Multiple Visual Targets", University of Oxford, 2005

[5] B. Palacios Serra, "Robust Feature Point Extraction and Tracking for Augmented Reality", Master Thesis, Êcole Polytechnique Fédérale de Lausanne, 2005

[6] C. Rasmussen, G. D. Hager, "Probabilistic Data Association Methods for Tracking Multiple and Compound Visual Objects", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 23 no.6, 2001, (pp. 560-576)

[7] C. Rasmussen, G.D. Hager, "Joint Probabilistic Techniques for Tracking Multi-Part Objects", Center for Computational Vision & Control, Yale University, 1998

[8] J. M. Motta, R. S. McMaster,"Experimental validation of a 3-D vision-based measurement system applied to robot calibration", J. Braz. Soc. Mech. Sci. v.24 n.3, Rio de Janeiro, 2002

[9] E.R. Davies, "Machine Vision, Theory, Algorithms, Practicalities", 3rd ed, Elsevier Inc., 2005, (pp. 595-623)

[10] Sebastian Thrun, "Lecture 2, Lenses and Calibration", CS223B Computer Vision, Stanford University, 2005 http://robots.stanford.edu/cs223b05/notes/

[11] I. J. Cox, S. L. Hingorani, "An Efficient Implementation of Reid's Multiple Hypothesis Tracking Algorithm and Its Evaluation for the Purpose of Visual Tracking", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 18 no.2, 1996, (pp. 138-150)

[12] M. Marrón, J.C. García, M.A. Sotelo, D. Fernández, D. Pizarro. ""XPFCP": An extended Particle Filter for tracking multiple and dynamic objects in complex environments", Proceedings of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS05), ISBN: 0-7803-8913-1, Edmonton, August 2005, (pp. 234-239)

[13] M. Marrón, J. C. García, M. A. Sotelo, E. J. Bueno, "Clustering methods for 3D vision data and its application in a probabilistic estimator for tracking multiple objects", Proceedings of the Thirty-First Annual Conference of the IEEE Industrial Electronics Society (IECON05), ISBN: 0-7803-9252-3, Raleigh, November 2005, (pp. 2017-2022)

[14] F. Gustafsson, L. Ljung, M. Millnert, "Signalbehandling", Studentlitteratur, Lund, ISBN: 91-44-01500-3, 2000, (pp. 273-309)

[15] C. J. Needham, R. D. Boyle, "Tracking multiple sports players through occlusion, congestion and scale", School of Computing, University of Leeds

[16] D. B. Reid, "An Algorithm for Tracking Multiple Targets", Transactions on Automatic Control, vol. AC-24, no.6, 1979, (pp. 843-854)

[17] M. Lindeborg, "Tracking Multiple Objects With Kalman Filters – Part II", Master Thesis, Universidad de Alcalá, 2006