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 II

Mikael Lindeborg

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 II

Author:	MIKAEL LINDEBORG
Alcalá Supervisor:	MARTA MARRÓN ROMERA
Home Supervisor:	MIKAEL EKSTRÖM
International Program Responsable:	ANTONIO GUERRERO BAQUERO

Examiners:

President:	Elena López Guillén
Examiner 2:	Juan Carlos García García
Examiner 3:	Marta Marrón Romera

MARK: ....

DATE: ....

Table of Contents

List of Figures

List of Tables

I. Abstract

In this report the implementation of a multiple object tracking algorithm is described. The algorithm is part of the obstacle avoidance system in an autonomous robot. The measurement vector used to achieve the tracking task comes from a stereo-vision system that detects objects in the robot's environment [1]. The algorithm uses the probabilistic Kalman filter (KF) to estimate the position and movement of different objects in the scene. One filter is used for each object to track. An algorithm for associating the data in the measurement vector to different objects is described. A validation process that the tracking algorithm uses to reduce the noise included in the measurement vector is also described.

II. Report

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.

There are two parts of the developed project. One of them focuses on the development of a simulator platform (Part I) and the other one on a real time platform (Part II). This part focuses on the real time application and will use some of the results that are found in the simulations in part I. Since the work is made for real time execution the implementation of this part focuses on achieving a small execution time.

The input data vector contains the XYZ-coordinates of image points that come from different objects in the scene, and the number of objects in the images can change with time. The input 3D points are obtained with a stereo-vision system, which has already been implemented in another project [1]. One KF are used to track each object. The output of the tracking algorithm to develop is the position, velocity and the number of objects in each image.

Since the input data from the image acquisition are discrete and the final implementation of the tracking system is developed in a computer, algorithms and signals discussed in this thesis are discrete.

The most difficult part of the project is to do the association between the measurement vector and the different estimators. This is because the 3D-points are not sorted in the input vector and the number of measurements for each object differs. In order to track the objects the 3D points must be organized. Probabilistic algorithms dealing with this problem have been presented and in Part I some of these are described.

The application for the algorithm presented in this thesis, is a part of the obstacle avoidance module. This module is used in a robot that moves autonomously in populated indoor environments.

The problem of finding a good estimate of the position of mobile objects in an environment is a problem that has great importance in autonomous robots. Knowledge about the position of moving objects can be used to improve the behavior of the system, especially if the robot is located in populated environments [2]. This ability allows the robot to adapt its velocity to the people one and helps the robot to avoid collisions in situations where the robot crosses the path of a human.

2 Objective

To achieve the work explained in the introduction some objectives must be stated, and these are presented in this section.

- **Tracking of multiple objects** - There are many algorithms to choose from when tracking objects. However, given the knowledge that the input data contains noise [1], reduces the options by the deterministic algorithms; systems containing noise requires probabilistic algorithms. In this work the probabilistic Kalman filters are used, since they provide the optimal implementation of the Bayes' filter [3]. 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 come from which object. To develop this association there are multiple choices of algorithms, as discussed in part I of the project. In this part an implementation of the "Nearest Neighbor Data Association Algorithm" is used. This algorithm uses the Euclidean distance as a measure for associating; the 3D points are assigned to the closet estimator.

- **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 arises more problems that need to be taken care of;

  - **Validation** – If the measurements indicate that a new object has just appeared a new estimator is created for it. To avoid that measurement noise are characterized as an object, data has to be associated to the object a certain number of iterations consecutively before it is validated as a real object.

  - **Occlusion** - Occlusion is when a 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 iterations even though no data is associated to it during this time.

  - **Crossing** - When two tracked objects cross each others 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 KF is the direction shown by the velocity.

- **Execution time** - The cameras used in the stereo-vision system, have a frame rate of 15 fps [1], this means that the sample time of the measurement acquisition system are ≈67 ms. This development must take this specification into account, and in cases where the execution time for the tracking process exceeds this time, it has to be able to take care of this.

3 The input and output models

The format of the input data is described in this chapter. To be able to present the results of the estimated process, it is necessary to know how to transform 3D data into the image coordinate system. In this section a description of these questions and how to deal with them are presented.

3.1 The input data

The objects detected in the images by the stereo-vision system produces a varying number of XYZ-coordinates. The generated data are organized in the following way:

1. ny – Information about the number of measurements found in this frame.
2. X coordinate.
3. Z coordinate.
4. Y coordinate.

Table 1 below shows the structure of the stereo-vision output data and figure 3.1 shows a description of the Cartesian coordinate system.

Table 3.1: The structure of the input data.

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

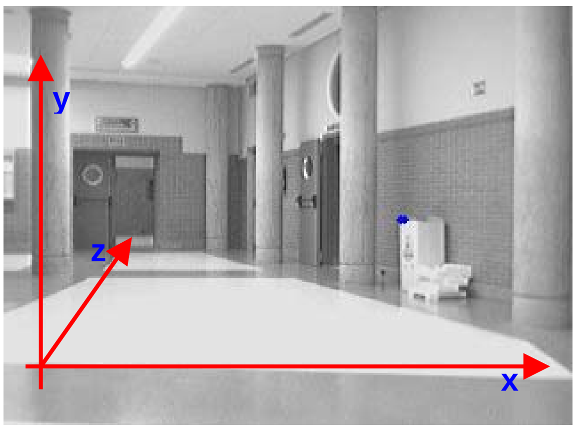

Figure 3.1: The Cartesian coordinate system.

3.2 Image transformation

To acquire information from an image correctly the camera used in the acquisition system has to be calibrated. The calibration is a process finding the relation between an image of the environment taken by a camera and the image itself. This process includes the position of the camera in the environment and the cameras internal characteristics [4]. In this project the **pinhole model** is used, which is a model of an ideal camera. The model is shown in figure 3.2.

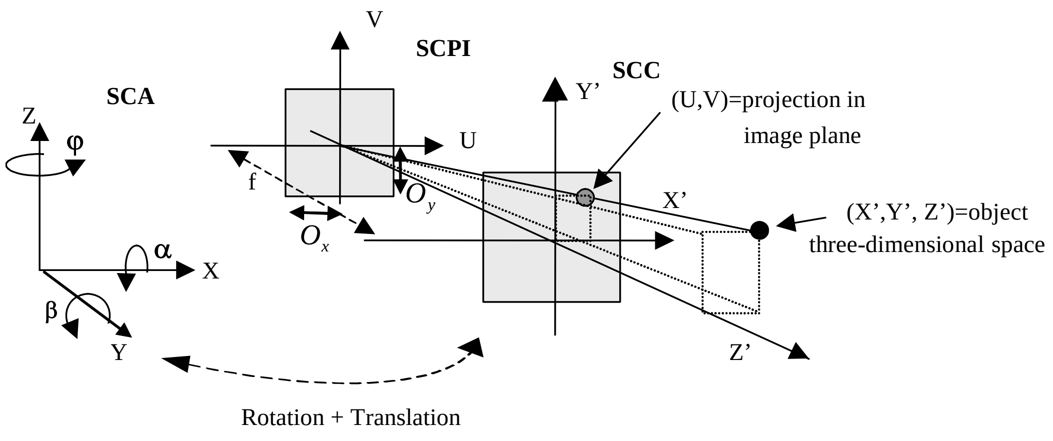
V
SCPI
SCC
SCA
Y'
(U,V)=projection in image plane
Z
φ
U
f
O_y
X'
O_x
(X',Y', Z')=object three-dimensional space
α
X
β
Y
Z'
Rotation + Translation

Figure 3.2: The pinhole model and the related coordinate systems.

In the pinhole model three coordinate systems can be considered to do the translation between 3D and 2D:

- **SCA** (System of Absolute Coordinates) $(X,Y,Z)^T$ - Information about position of the points in 3D space. The origin of this coordinate system will be the reference of the 3D coordinate position of the objects.

- **SCC** (System of Camera Coordinates) $(X',Y',Z')^T$ - A coordinate system whose origin is located in the optic center of the camera. The main restriction of the pinhole model is that the Z'-axis in the SCC coincides with the optical axis.

- **SCPI** (System of Image Plane Coordinates) $(u,v)^T$ - 2D system that represents a position in the image plane. The SCPI coordinates u and v are only a direct projection of the X' respective Y' axis.

The transformation points from the SCA and SCPI can be divided into three steps:

1. Move the origin of the SCA to the origin of SCC.

2. Rotate the SCA until its axes are coinciding with those of the SCC.

3. Move the SCPI laterally until there is complete agreement between the two coordinate systems.

In the following paragraphs the different matrixes needed to do the SCA to SCPI point transformation are summarized. In the first step, the translation matrix T, that moves the origin of the absolute coordinates into the origin of the camera coordinates, is added. See equation 3.1.

Equation 3.1

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

In the second step the rotations around the coordinate axes are performed; α around the X-axis, β around the Y-axis and φ around the Z-axis. Equation 3.2-3.4 shows these rotations.

Equation 3.2

$$\mathbf{Z}(\varphi) = \begin{bmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Equation 3.3

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

Equation 3.4

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

The sequence of rotations around the SCA coordinate axes can be expressed as $\mathbf{R}(\alpha,\beta,\varphi) = \mathbf{X}(\alpha)\mathbf{Y}(\beta)\mathbf{Z}(\varphi)$ where **R** is a composite rotation matrix, in which $\mathbf{Z}(\varphi)$ is applied first, then $\mathbf{Y}(\beta)$ and finally $\mathbf{X}(\alpha)$. The rotation matrix is orthogonal and thus it has the property that $\mathbf{R}^{-1} = \mathbf{R}^{T}$ [4]. The rotation value is expressed by equation 3.5.

Equation 3.5

$$\begin{bmatrix} X'(R) \\ Y'(R) \\ Z'(R) \end{bmatrix} = \begin{bmatrix} \cos\beta\cos\varphi & -\cos\beta\sin\varphi & \sin\beta \\ \sin\alpha\sin\beta\cos\varphi + \cos\alpha\sin\varphi & \sin\alpha\sin\beta\sin\varphi + \cos\alpha\cos\varphi & -\sin\alpha\cos\beta \\ -\cos\alpha\sin\beta\cos\varphi + \sin\alpha\sin\varphi & \cos\alpha\sin\beta\sin\varphi + \sin\alpha\cos\varphi & \cos\alpha\cos\beta \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

The generalized displacement (i.e. translation plus rotation) between SCA and SCC is shown by equation 3.6.

Equation 3.6

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \begin{bmatrix} \cos\beta\cos\varphi & -\cos\beta\sin\varphi & \sin\beta \\ \sin\alpha\sin\beta\cos\varphi + \cos\alpha\sin\varphi & \sin\alpha\sin\beta\sin\varphi + \cos\alpha\cos\varphi & -\sin\alpha\cos\beta \\ -\cos\alpha\sin\beta\cos\varphi + \sin\alpha\sin\varphi & \cos\alpha\sin\beta\sin\varphi + \sin\alpha\cos\varphi & \cos\alpha\cos\beta \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix}$$

To be able to express the generalized displacement as a product of matrices, they must be enlarged to 4x4 as shown by equation 3.7

Equation 3.7

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\beta\cos\varphi & -\cos\beta\sin\varphi & \sin\beta & T_X \\ \sin\alpha\sin\beta\cos\varphi + \cos\alpha\sin\varphi & \sin\alpha\sin\beta\sin\varphi + \cos\alpha\cos\varphi & -\sin\alpha\cos\beta & T_Y \\ -\cos\alpha\sin\beta\cos\varphi + \sin\alpha\sin\varphi & \cos\alpha\sin\beta\sin\varphi + \sin\alpha\cos\varphi & \cos\alpha\cos\beta & T_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

In the third step the SCC coordinates are normalized by Z' and multiplied by the camera focal length (f). In order to plot the coordinates in the image it is necessary to transform them into pixels. That is done by dividing by scaling factors *s_x* and *s_y* [5], and correcting the position of the image plane origin by the offsets *o_x* and *o_y* .The equation 3.8 below shows how to do the second transformation (from SCC to SCPI).

Equation 3.8

$$u = \frac{f}{s_x} \cdot \frac{X'}{Z'} + o_x\,,\; v = \frac{f}{s_x} \cdot \frac{Y'}{Z'} + o_y$$

3.3 The output model

The objects in the scene are considered to move on the floor, this means that the tracking is performed in the XZ-plane only. The KF models all systems with Gaussian probability distribution, so the objects can be characterized by a cylinder centered on their estimated center position. Figure 3.3 shows a description this “cylinder model”.

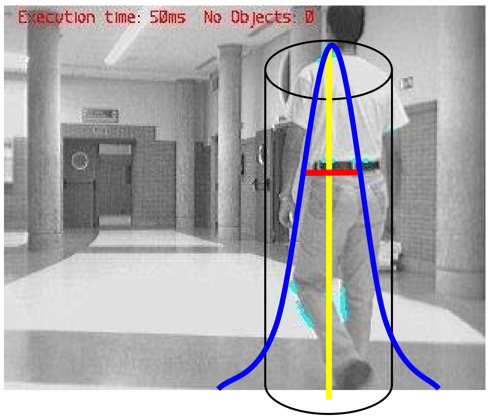

Figure 3.3: The objects are characterized by a cylinder around their estimated center position. The center of the Gaussian is the estimated center (yellow line), the 3 dB line (red) indicates the estimation error covariance (P).

The cylinder is given a constant height since the tracking is performed in the XZ-plane. The radius is a tracking parameter that is specified to achieve different tracking behavior; the radius decides the maximum distance the measurements can be positioned at to be considered to come from the object. This parameter is important; imagine for example if two persons are walking next to each other. Then the radius has to be set so that both can be tracked as objects. If it is set too small one person can instead be identified as two.

The cylinder is shown as a rectangle in the image plane projection, and as a circle in a XZ-projection plane plot. Figure 3.3 shows an example of these plots. For plotting purposes the width of the rectangle does not match exact width of the cylinder.

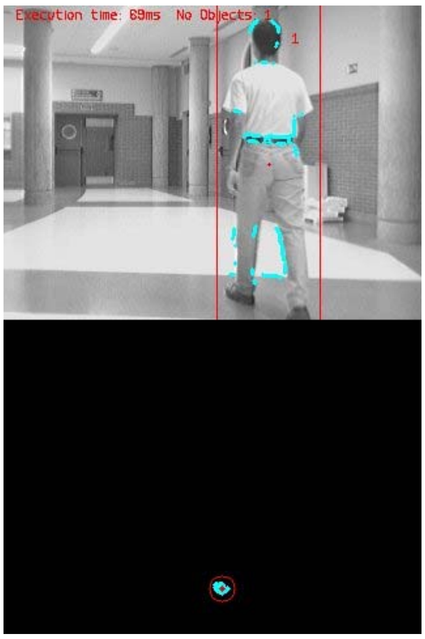

Figure 3.4: An example of an object representation.

4 Tracking algorithms

The tracking algorithm implemented is, as mentioned, part of the obstacle avoidance process in a robot's navigation system. In this chapter the different "sub-processes" of the tracking algorithm is described.

4.1 The discrete Kalman filter

The KF is the optimal implementation of the Bayes' filter [3]. A description of the KF functionality and the way to adjust its parameters are described in this section. The limitations of this filtering method are also mentioned.

4.1.1 The process to be estimated

The KF is a recursive algorithm used to obtain the optimal estimation value of a state vector $\vec{a} \in \Re^n$ [3]. To use the KF, the process can be used with a linear set of equations shown in 4.1 and 4.2. Equation 4.1 is also called the state equation and it expresses the evolution of the process' state vector with time. Equation 4.2 is also called the output equation and describes what is going to be the output of the system taken into account the current state vector.

Equation 4.1

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

Equation 4.2

$$\vec{m}_k = C\vec{a}_k + \vec{o}_k$$

w_k and o_k represent the process respectively the measurement noise. In order to use the KF, the noises must be independent of each other, white and with Gaussian (normal) probability distributions and with zero mean. This is shown in equation 4.3 and 4.4.

Equation 4.3

$$p(\vec{w}) \sim N(0,Q).$$

Equation 4.4

$$p(\vec{o}) \sim N(0,R).$$

In equation 4.1 G is an $n \times n$ matrix that relates the state vector at the previous time step to the one at the current time step, where n is the number of states. H is an $n \times l$ matrix that relates the state vector to the control input matrix, $\vec{u} \in \Re^l$, l is the number of inputs. The $m \times n$ matrix C relates the state vector to the measurement vector, $\vec{m}_k$, where m in $(m \times n)$ is the number of measurements. R and Q are respectively the measurement and state noise covariance matrices. The vector sign $(\rightarrow)$ will be left out in all equations from now on.

As already mentioned above, the KF assumes that both the measurement model and the state model are linear, and that the errors have a Gaussian probability distribution. These inherent assumptions restrict its use. However, the system presented in this report is supposed to be linear and Gaussian system.

4.1.2 Definition of the Kalman filter equations

In the following equation:

- $\hat{a}^- \in \Re^n$ is defined to be *priori* state estimation at step *k* given knowledge of the process prior to step *k; k-1.* The $\wedge$ sign indicates that it is an estimation and the minus sign indicates that it is a *priori* estimation.
- $\hat{a} \in \Re^n$ is defined to be *posteriori* state estimation at step *k* given the measurement m_k.
- The errors for a *priori* and a *posteriori* estimation can then be defined as shown by equation 4.5 respectively 4.6.

Equation 4.5

$$e_k^- \equiv a_k - \hat{a}_k^-$$

Equation 4.6

$$e_k \equiv a_k - \hat{a}_k$$

- The *priori* estimation error covariance is expressed by equation 4.7 and the *posteriori* estimation error covariance is expressed by 4.8, T denotes the matrix transpose.

Equation 4.7

$$P_k^- = E\left[e_k^- e_k^{-T} \right]$$

Equation 4.8

$$P_k = E\left[e_k e_k^T \right]$$

These equations are achieved through the Bayes' filter theory. They are useful when deriving the expressions for the KF as the goal, achieved by the algorithm formulation, 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 prediction of the measurement vector $C\hat{a}_k^-$. The found expression is shown in equation 4.9 below.

Equation 4.9

$$\hat{a}_k = \hat{a}_k^- + K_k(m_k - C\hat{a}_k^-)$$

In the previous expression K is called the KF gain and it is the $n \times m$ matrix that minimizes the *posteriori* error covariance. This minimization is done in the following way:

- Writing the equation 4.9 into the definition of the estimated error (equation 4.5 and 4.6).
- Substituting the acquired expression into the estimate error covariance equation (equation 4.7 and 4.8).
- Minimizing the expression with respect to K.

The resulting KF gain, K, is given by equation 4.10 below.

Equation 4.10

$$K_k = P_k^- C'(CP_k^- C' + R)^{-1}$$

The minimized *posteriori* error covariance can then be obtained, using the Kalman filter gain in equation 4.10 above, resulting in equation 4.11 below.

Equation 4.11

$$P_k = (I - K_k C)P_k^-$$

The KF does the estimation in two phases: **prediction** and **correction**. At the prediction step, the algorithm predicts the value of the state vector and the estimation covariance error using the system model (equation 4.1). Equation 4.12 and 4.13 describes the prediction process.

Equation 4.12

$$\hat{a}_k^- = G\hat{a}_{k-1} + Hu_{k-1}$$

Equation 4.13

$$\hat{P}_k^- = GP_{k-1}G' + Q$$

In the correction phase the KF adjusts the prediction with an actual measurement, using equation 4.9 – 4.11, and Bayes' rule, zeroing the process noise. Equation 4.14 shows Bayes' rule and figure 4.1 below gives a complete picture of the operation and equations of the KF.

Equation 4.14

$$P(A|B)=\frac{P(B|A)P(A)}{P(B)}$$

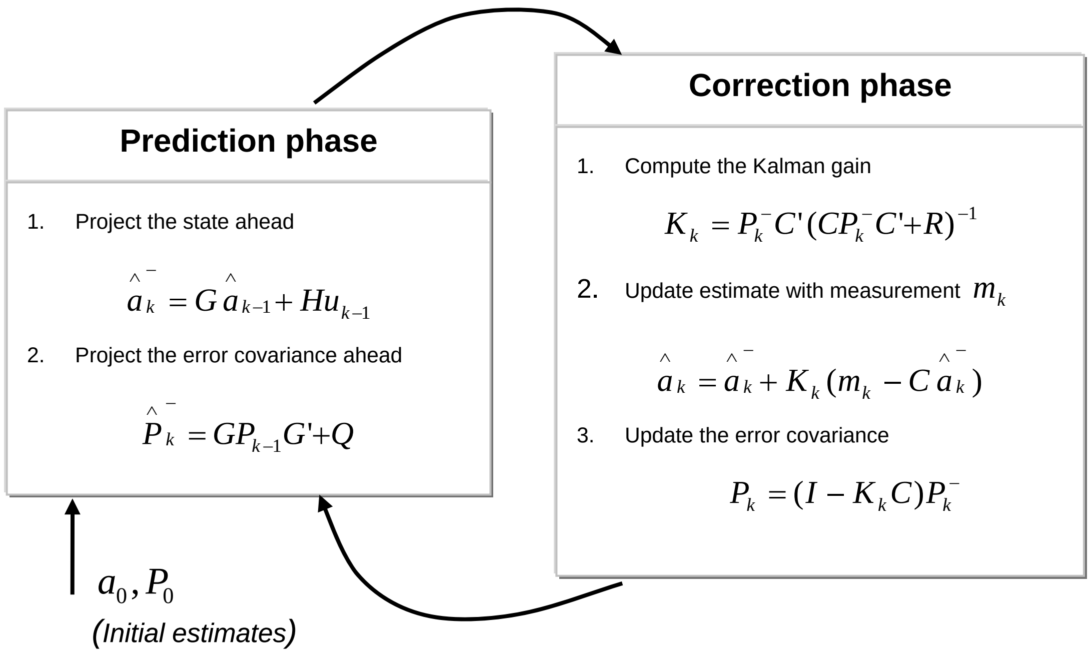
Correction phase
1. Compute the Kalman gain
$K_k = P_k^- C'(CP_k^- C'+R)^{-1}$
2. Update estimate with measurement m_k
$\hat{a}_k = \hat{a}_k^- + K_k(m_k - C\hat{a}_k^-)$
3. Update the error covariance
$P_k = (I - K_k C)P_k^-$
Prediction phase
1. Project the state ahead
$\hat{a}_k^- = G\hat{a}_{k-1} + Hu_{k-1}$
2. Project the error covariance ahead
$\hat{P}_k^- = GP_{k-1}G'+Q$
a_0, P_0
(Initial estimates)

Figure 4.1: A summarize of the Kalman filter equations.

4.1.3 Filter parameters and tuning

When Q and R matrixes are constant, the estimation covariance P_k and the Kalman gain K quickly stabilizes and remains constant [3].

The time it takes the filter to converge has to do with the initial value of the estimation covariance error (P_0). If the initial value is too small it takes more time for the filter to converge [6], than if it is big. On the other hand, if the initial value is set too big the filter will never converge. So, the initial value should be chosen carefully.

Manipulation of the filter behavior is possible through K, by changing the R. For example, if R is made bigger the filter gets slower to respond to the measurements information, resulting in a reduced estimation covariance, since the KF gain, K, decreases. If the R is made smaller the filter responds to the measurements quickly, increasing the estimation covariance since K increases.

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 a unknown, acceptable estimation results can be accomplished if enough uncertainty is injected via the of Q.

In many applications the R and Q do not remain constant. In order to adjust this the matrices R and Q becomes R_k and Q_k. This is a manipulation that can be done to simulate changes in the dynamics and in level if noise. In tracking algorithms for example, the magnitude of Q_k can be reduced when the object seems to be moving slowly and increase the magnitude if the dynamics start changing rapidly.

4.1.4 The system to be implemented

It is necessary further describing of the tracking algorithm developed to define the state vector of the system related to the tracking task.

In the state equation, the state vector (a) consists of the X and Z position of the cylinder center in Cartesian coordinates. These coordinates are updated with the help of the velocity of the objects in both X and Z directions, that are considered the input vector (v) in the system model. The state vector is shown by equation 4.15 below.

Equation 4.15

$$a_k = G\,a_{k-1} + H\,v \Leftrightarrow \begin{bmatrix} x_k \\ z_k \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k-1} \\ z_{k-1} \end{bmatrix} + \begin{bmatrix} T_s & 0 \\ 0 & T_s \end{bmatrix} \begin{bmatrix} v_x \\ v_z \end{bmatrix}$$

T_s is the expected sample time of the global discrete estimation process. The output equation that relates the state vector to the measurement one is defined by equation 4.16 below. The measurements are received in both X and Z Cartesian coordinates as well.

Equation 4.16

$$m_k = C\,a_k \Leftrightarrow \begin{bmatrix} x_k \\ z_k \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_k \\ z_k \end{bmatrix}$$

The velocity in 4.15 is calculated in both X and Z directions according to equation 4.17, using the measured sample time.

Equation 4.17

$$v_k = \frac{m_k - a_{k-1}}{T_s}$$

Figure 4.2 shows the Signal-flow-model of the KF process.

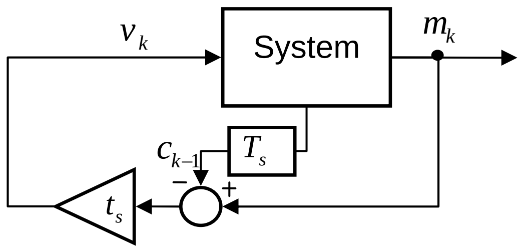

Figure 4.2: Signal-flow-model of the Kalman filter process.

The process noise covariance matrix and the measurement noise covariance matrix are then expressed by the following equations 4.18 and 4.19.

Equation 4.18

$$Q = \begin{pmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_z^2 \end{pmatrix}$$

Equation 4.19

$$R = \begin{pmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_z^2 \end{pmatrix}$$

In equation 4.18 the σ^2 indicates the variance of the process noise and in 4.19 the measurement noise, both in corresponding Cartesian coordinates. These values are discussed in section 6.1.

4.2 Data association

As stated in chapter 2 of this document the tracking system has to be able to handle multiple and various numbers of objects in the scene. In order to deal with this specification an algorithm for associating the measurement to the different objects is needed.

There are several algorithms proposed for tracking multiple objects in the scientific literature. Some of them are discussed in Part I of this project. In this part the Nearest Neighbor Data Association Algorithm (NN), is used. The NN chooses the nearest measurement within its cylinder (for cylinder description see chapter 3.3) as the most adequate. To calculate the Euclidean distance between two points with coordinates (x_1, z_1) and (x_2, z_2) the equation 4.20 below can be used.

Equation 4.20

$$d = \sqrt{(x_2 - x_1)^2 + (z_2 - z_1)^2}$$

Since the NN chooses the nearest measurements 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.

4.2.1 The data association algorithm

In figure 4.2 the association algorithm used in the implementation described in this document is shown, and the following points describe the process:

- The distances from each of the measurements, at each time step, to the priori estimation of each object’s centers are calculated.

- If the current measurement’s shortest distance to the closet of the priori estimation is smaller the predefined cylinder radius the measurement is associated with this estimator. Otherwise a new estimator is created, and then it is necessary to restart the association of measurements (see section 5.1.1 for further description).

- When all the measurements are assigned, the posteriori objects that did not have any associated measurements this iteration are marked to be removed.

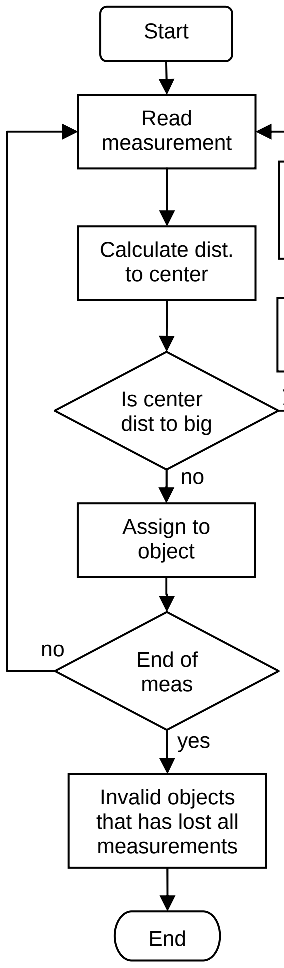
Start
Read measurement
Restart the assignment of measurements
Calculate dist. to center
Start new object
Is center dist to big
yes
no
Assign to object
no
End of meas
yes
Invalid objects that has lost all measurements
End

Figure 4.3: The data association algorithm.

4.2.2 Improving the algorithm

In order to reduce the miss-identification of input noise as objects, the newly created estimators are considered as *candidate objects* a programmable number of iterations. After these iterations, where the object has appeared all the iterations, the objects considered a *validated object*. The number of iterations you choose as a *validation limit* should be adapted to how noisy the input data is.

When a posteriori object is marked to be removed by the association process the estimator is not removed at once. Instead it is set as a *candidate to erase*. Sometimes objects in the scene are occluded or the image acquisition system does not produce measurements from them. This improves the system's performance with these problems. The number of iterations before the estimator is removed is a parameter, *invalidation limit*, that can be varied to achieve different system behavior.

The validation and invalidation limits are tested in section 6.3. However the invalidation limit is usually the bigger one since it is used to handle occlusion, while the validation limit is set to suppress outliers.

5 Implementation

This chapter describes the final implementation of the algorithms discussed theoretically in the previous one. Some improvements that are necessary for the thesis implemented are also presented. Figure 5.1 shows the flowchart of the global algorithm and the different tasks included in it are discussed more thoroughly in the following sections. The source code for the implementation can be found in chapter V.

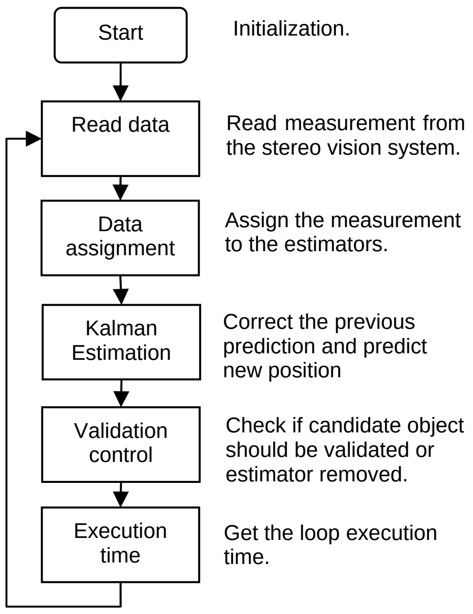
Start
Initialization.
Read data
Read measurement from the stereo vision system.
Data assignment
Assign the measurement to the estimators.
Kalman Estimation
Correct the previous prediction and predict new position
Validation control
Check if candidate object should be validated or estimator removed.
Execution time
Get the loop execution time.

Figure 5.1: The global process.

A struct named *data* has been created to store information about each object. It has the following members:

- *XZY[3]* – The coordinates for the Kalman estimated positions of the object center (a_k).
- *Identify* – Indicates if the struct data value is active. If its value is 1 it is active. If it is -1 the array member is empty and a new object can be stored in this position. 0 indicates that it is a newly created estimator.
- *addCand* – Is updated during the time a new object is in the validation process.
- *remCand* – Is updated during the time an object is in the process to be removed.

5.1 Data association

The data association algorithm implemented in this thesis is shown in figure 5.2, and described in the following steps:

- Each set (XZY) of measurements' distance to the priori objects' centers is calculated (the priori object center is the predicted object center in the previous iteration).
- If this distance to the nearest priori object is not longer than the maximum radius of the cylinder, the measurement is assigned to that priori object.
- In the case where the minimum distance found between the mean and the priori objects is too big it is necessary to start a new estimator. This process is described more thoroughly below in this section.
- After all the measurements have been associated, the remCand struct member is zeroed for those that have been assigned measurements.

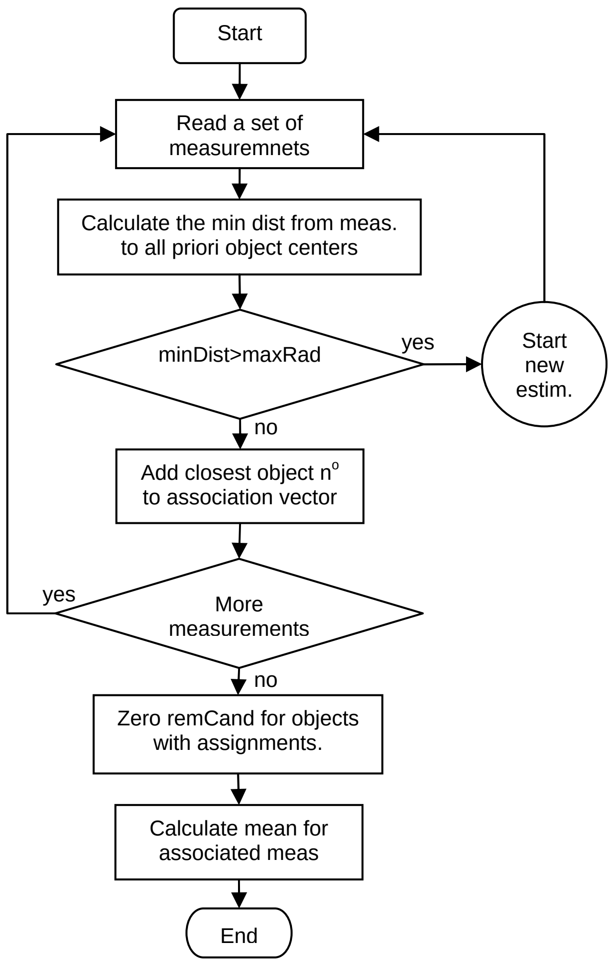
Start
Read a set of measuremnets
Calculate the min dist from meas. to all priori object centers
minDist>maxRad
yes
Start new estim.
no
Add closest object n° to association vector
More measurements
yes
no
Zero remCand for objects with assignments.
Calculate mean for associated meas
End

Figure 5.2: The data association process.

- Finally the mean is calculated for all the measurement associated to each priori object.

The process of starting a new estimator is shown in figure 5.3 and it works as follows:

- The data struct member *addCand* is initialized, to indicate that it is a candidate object before it is validated.
- The first measurement assigned a new object is set to be considered the center of the object during the rest of the association this iteration.
- The global association process is then restarted, because it is necessary to test if there might be measurements already associated with other object that are closer to this new candidate object.

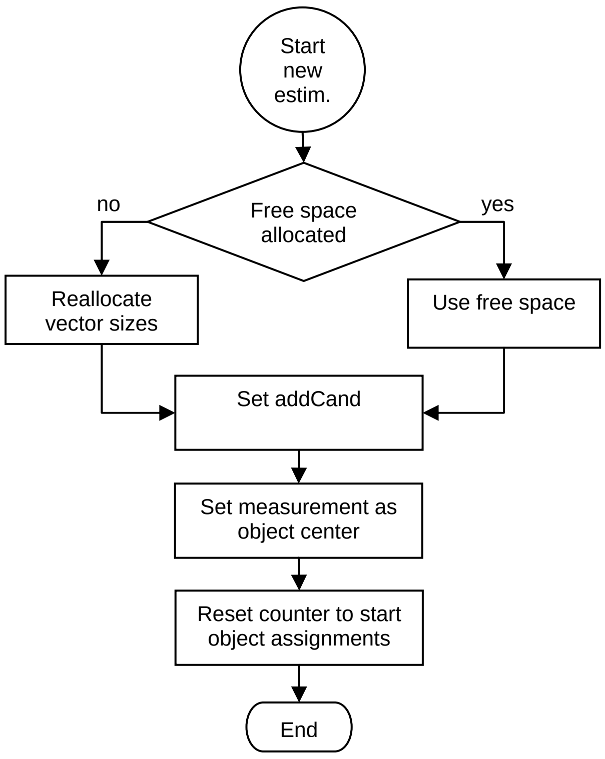

Figure 5.3: The process of creating a new object.

5.2 Kalman estimation

In figure 5.4 the process of the KF is shown and it is described in the following steps:

- If it is the estimators' first iteration, indicated by the struct member *Identify*, the starting values are assigned. The state is set to the mean calculated for the object, the starting velocity is set to zero and the start value of the estimation covariance error matrix are set to unity, see section 6.1.3.

- Then the correction of the priori object position and priori estimation error covariance is performed; correcting the predictions from the previous iteration. The correction step is performed only in those iterations where there are measurements associated for the priori object. Table 5.1 shows how the different KF time steps for each object are performed. The equations shown in this table are the KF equations found in 4.9 and 4.12 and the velocity calculation equation in 4.17.

- The next step is the prediction of a new priori object center and priori estimation covariance error (next iterations object center and estimation covariance error) is performed. If the object struct member *remCand* is non-zero, the velocity that this object struct had the last time it were associated with measurements is used to predict. This helps maintaining the tracking when the object is occluded.

In the Kalman estimation function the data is put into matrices so the KF calculations can be performed more easily.

Table 5.1: The time steps in the Kalman Filter.

	t=0	t=1	t=2		t=n
Correction	$\hat{a}_0 = m_1$	$\hat{a}_1 = a_1^- + K(m_1 - Ca_1^-)$	$\hat{a}_2 = a_2^- + K(m_2 - Ca_2^-)$	…	$\hat{a}_n = a_n^- + K(m_n - Ca_n^-)$
Velocity	$v_0 = 0$	$v_1 = \frac{m_1 - \hat{a}_0}{Ts} = 0$	$v_2 = \frac{m_2 - \hat{a}_1}{Ts}$		$v_n = \frac{m_n - \hat{a}_{n-1}}{Ts}$
Prediction	$a_1^- = m_1$	$a_2^- = \hat{a}_1 + v_1$	$a_3^- = \hat{a}_2 + v_2$		$a_{n+1}^- = \hat{a}_n + v_n$

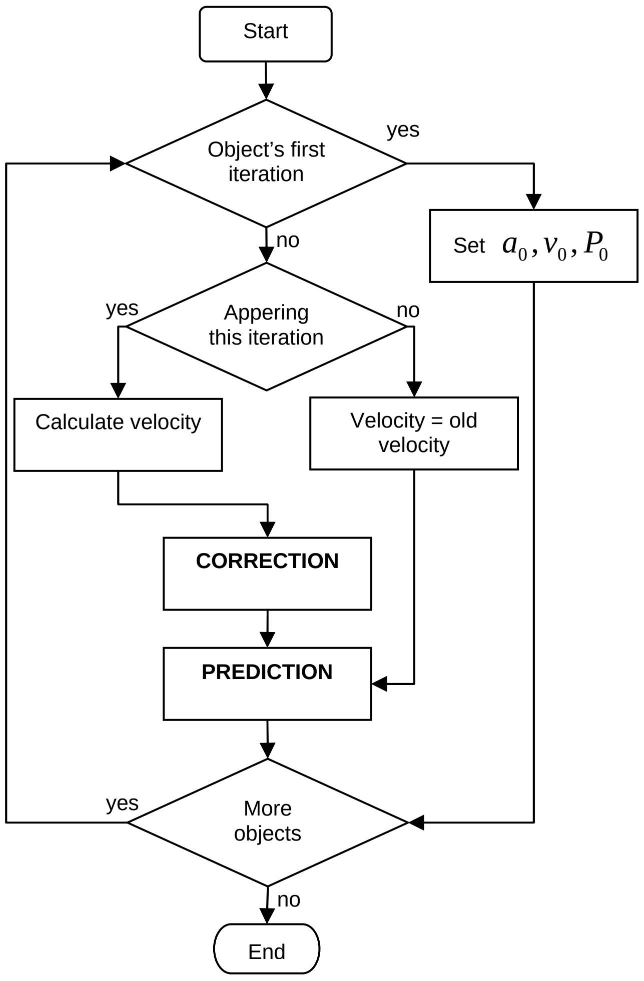

Figure 5.4: The Kalman estimation process.

5.3 Object validation

The object validation process is performed according to figure 5.5 and in the following steps it is described:

- The process starts by checking whether the current object struct is marked as candidate. In that case, it checks the object's *addCand* variable to see if it has been a candidate for as long as the condition for being validated is set to, indicated by the validation limit. In that case the candidate variable (*addCand*) in the related struct is zeroed and the estimator is now a validated object.

- If the object struct is a marked as candidate and the remCand variable is not zero, i.e. no measurements were associated to it this iteration, the estimator is removed. This is indicated by the invalidation limit and is done simply by setting the *Identify* field in the struct to -1.

- The same process as in the point above is performed when measurements for a validated object has been missing for as long as the condition to be removed is fulfilled.

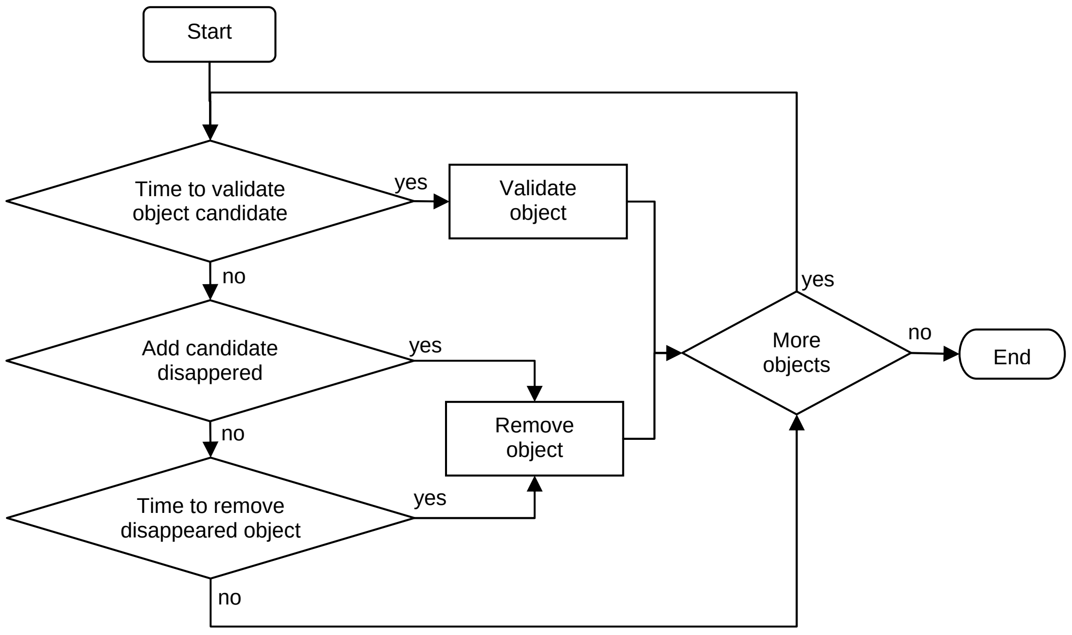

Figure 5.5: The validation process.

5.4 Execution time

The cameras' frame rate is 15 fps, giving a sample time of the acquisition system $1/15 \approx 67ms$. This means that the tracking algorithm's execution time should not exceed this.

In the global tracking process the execution time of the loop is measured. If the measured sample time is shorter than the expected, the system has to pause for the remaining of the sample time. If the execution time exceeds the expected, this may be partly compensated for in the Kalman filter since the velocity is calculated with the measured sample time. In cases where it exceeds the sample time much, the estimation process can miss one or more frames, for this reason it is important to optimize the computational load of the tracking algorithm.

5.5 Plotting

A function for plotting the tracking results in the captured images has been developed. The type of draw, color and the draw position are sent to the function as input arguments. The function uses the image transformation equations presented in section 3.2. The different types of plots are that can be shown in the images are:

- **DotUV –** Draws the measurements and the Kalman estimated center in the images.

- **DotXZ –** Draws the measurements and the estimation center in XZ-projection plane (tracking plane in the Cartesian space).

- **Circ –** Draws a circle around the Kalman estimated center, which represents the cylinder seen from above, in the XZ-projection plane.

- **Rect –** Draws a rectangle around the estimated center position, representing the cylinder seen from the side.

- **Text –** Draws a text in the image, giving information about the sample time and the number of the objects.

5.6 Improving the performance with velocity smoothing

In this section a performance improvement of standard tracking processes is described. The improvements with the object validation process and the predicting with “old velocity” have already been described.

The quality of the input measurements, produced by the stereo-vision system, can vary depending on the intensity level in the scene [1]. In some images the vision system can extract measurements around the whole object and at other times only at on side of the object. This can lead to a rather unstable behavior in the Kalman filter, since the modulus and direction of the velocity that is calculated from the measurements will vary depending on the vision process. In order to deal with this problem a *velocity smoothing* process is developed, see equation 5.1.

Equation 5.1

$$V_k = \frac{V_k + V_{k-1}}{2}$$

This smoothing produces a more even estimator behavior since changes in velocity are filtered.

6 Results

In this chapter the results obtained from the implemented tracking algorithm are presented. The following questions are going to be analyzed in order to see the quality of the designed tracker:

- Behaviour against different values for the measurement and state noise covariance matrices (R and Q) and finding a good start value of the estimation noise covariance (P_0), as described in section 4.1.3.

- Velocity smoothing, as described in section 5.2.

- Validation parameters, the validation and invalidation limits, as described in section 4.2.2.

- Cylinder radius, as described in section 3.3.

The issus analyzed are, in most cases, related to the different parameters the tracker has. The value of some that have been fixed rigorously through the theory (like Q, R and P_0), will be validated with these experiments presented in this chapter. The value of the rest will be tuned empirically, as shown here.

The quality factors used to validate the parameters are mainly: number of objects detected, estimation error and execution time, at each frame.

The exact position of the objects is not known. In order to get an idea about how good the estimations are, the distance to the mean of the measurements manually associated to an object is used. This is referred to as *manual background truth*. The number of objects will also be manually counted for each frame and used as manual background truth.

The images that are shown in this chapter are the image plane projection and the XZ-projection plane.

6.1 Adjusting the Kalman filter parameters

In order to test the performance of the tracking algorithm for the different Kalman filter parameters (Q, R and P_0) the following conditions are fixed in all the experiments in this section:

- Single object position estimation, during all experiments in this section.

- Fixed validation parameters, validation limit = 3 and invalidation limit = 10. Since it is an only object and it does not disappear nor reappear during the experiment the validation parameter has no importance.

- Fixed cylinder radius, 900 mm. The radius is discussed more in section 6.4, however this value has empirically shown to give good tracking results.

The quality factors to measure are the following:

- Estimation error (R and Q test).
- Convergence time (P_0 test).

To gain by the results of these experiments is:

1. The best approximation value of R.
2. The best approximation value of Q.
3. A initial value for the estimation covariance error, P_0.

6.1.1 Manipulation by changing the values in R

The mean variance for the input measurements from the only object, during the 15 measured iterations in the middle of the field-view scene, are found to be: σ_x^2 =30871 mm^2 and σ_z^2 =11226 mm^2. Using these values in the KF causes the filter to diverge. Instead a ratio between the variance and the cylinder radius, in which the variance is measured, is calculated as shown by equation 6.1 below.

Equation 6.1

$$\sigma_{x-ratio}^2 = \left(\frac{\sigma_x}{900}\right)^2 = \frac{30871}{900^2} \approx 0.038 \text{ , } \sigma_{z-ratio}^2 = \left(\frac{\sigma_z}{900}\right)^2 = \frac{11226}{900^2} \approx 0.014$$

The measurement noise covariance matrix then equals: $R = \begin{bmatrix} 0.038 & 0 \\ 0 & 0.014 \end{bmatrix}$.

Figure 6.1 shows how the estimation error changes with different values of R in the tracking experiment. The values of the estimation covariance error (the diagonal of P), for the tested R values are shown in table 6.1. Notice that when R is made bigger the estimation covariance error gets bigger, as explained in section 4.1.3.

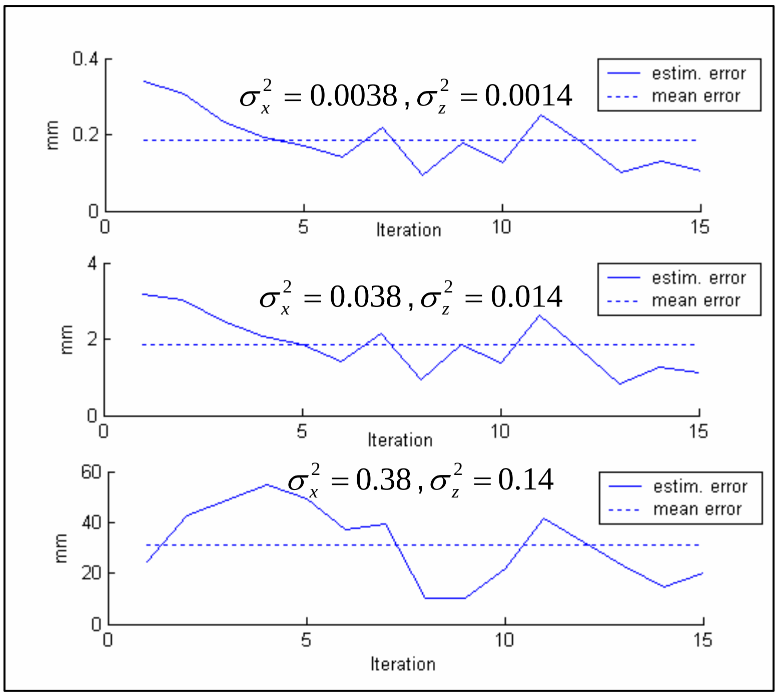

Figure 6.1: The estimation error when testing with different sizes in the R matrix.

Table 6.1: Estimation covariance error for different values in the R matrix.

R (mm^2)	**P** (mm^2)
$\sigma_x^2 = 0.0038, \sigma_z^2 = 0.0014$	0.0038
$\sigma_x^2 = 0.038, \sigma_z^2 = 0.014$	0.0381
$\sigma_x^2 = 0.38, \sigma_z^2 = 0.14$	0.4225

The results in figure 6.1 show a small and smooth estimation error for the smaller values of R. It is necessary to keep in mind that it is only the manual background truth, so the values are not to be trusted entirely.

The measured variance indicates that the σ_x^2 is bigger than σ_z^2 this is because the measurements are in a less comfortable situation for the x-coordinate than for the z-coordinate. Based on this knowledge and the results in the experiments above the following value is thought to be the most adequate value of the measurement noise covariance matrix.

$$R = \begin{bmatrix} 0.03 & 0 \\ 0 & 0.03 \end{bmatrix}$$

6.1.2 Manipulation by changing the values in Q

As explained in section 4.1.3, the state noise covariance matrix, Q, informs if the process is well known. It is possible to achieve acceptable estimation results when the process model is unknown, as in this case, if the values in Q is made big enough. The start value is thereby set to the unity matrix, both a bigger and a smaller value are also tested. Figure 6.2 shows the estimation error results of a series of testing with different Q-values and in table 6.2 the estimation covariance error (the diagonal) for the tested values are shown.

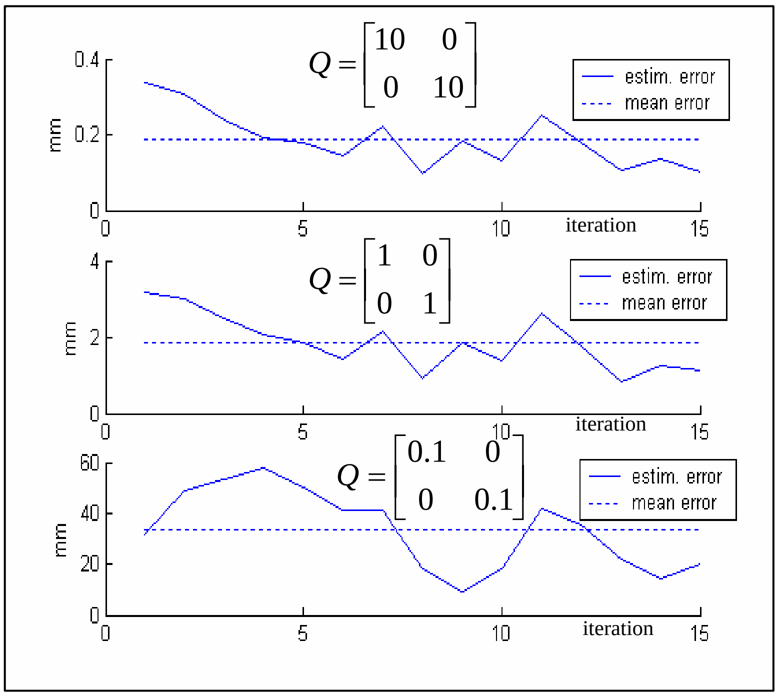

Figure 6.2: The estimation error when testing with different values in the Q matrix.

Table 6.2: Estimation covariance error for different values in the Q matrix.

Q (mm^2)	**P** (mm^2)
$0.1*I$	0.0298
$1*I$	0.0367
$10*I$	0.0379

These results, in figure 6.2, show a smooth and small estimation error.

As described in section 4.1.3, in scenes with much dynamics the magnitude of Q has to be bigger than in scenes with small dynamics. Figure 6.3 illustrates what happens when the values in Q is set too small. In this experiment Q is set to 0.1$*$unity matrix (left picture) respectively the unity matrix (right picture).

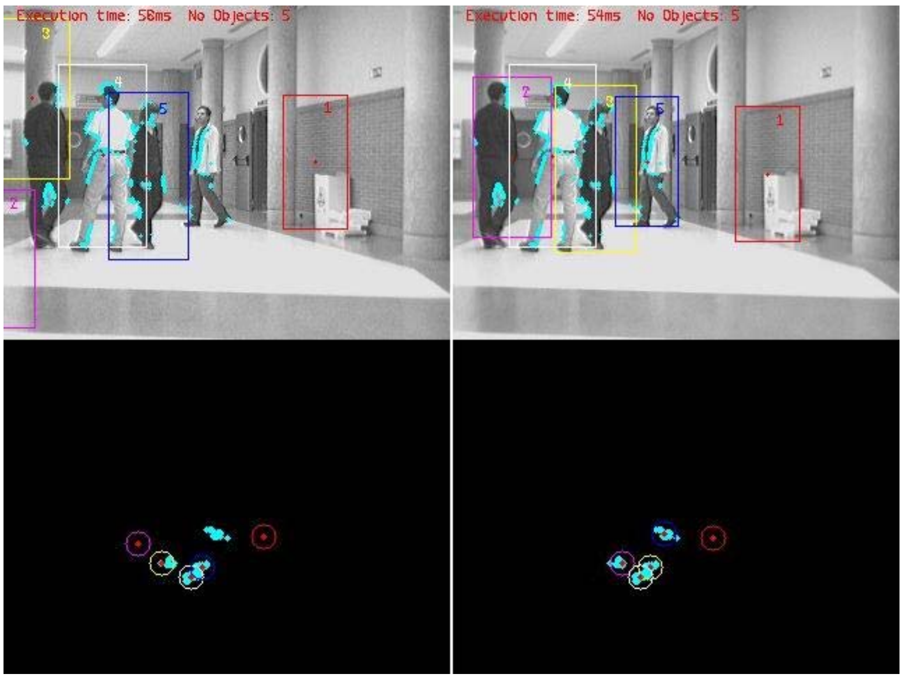
Execution time: 58ms No Objects: 5
Execution time: 54ms No Objects: 5

Figure 6.3: Result of the implemented system, showing that much dynamics in the image requires a bigger Q. The picture on the left has a too small Q.

Based on these experiments the process noise covariance matrix is fixed during the rest of the experiments to the following value:

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

6.1.3 Initial value of estimation covariance error

As stated in section 4.1.3 the initial value of estimation covariance error, P_0, should not be set too small. A test with the unity matrix gives a good filter convergence time, around 1-2 iterations. The initial value of P_0 is therby fixed to the following value:

$$P_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

6.2 Velocity smoothing

In order to test the effect of the tracking algorithm’s velocity smoothing the same conditions and parameters settings as in section 6.1 are used:

- Single object position estimation.
- R, Q and P_0, fixed to the values in section 6.1.

- Fixed validation parameters, validation limit = 3 and invalidation limit 10.

- Fixed cylinder radius, 900 mm.

The quality factors to measure:

- Estimation error.

The objective with this experiment is to get a more smooth estimation behaviour, as described in section 5.2.

Figure 6.4 and 6.5 below shows the effect of velocity smoothing for 35 iterations on a single object tracking. The smoothed velocity (dashed line) does not change as much as the non-smoothed one (solid line) between different iterations, resulting in a more even estimation error behaviour.

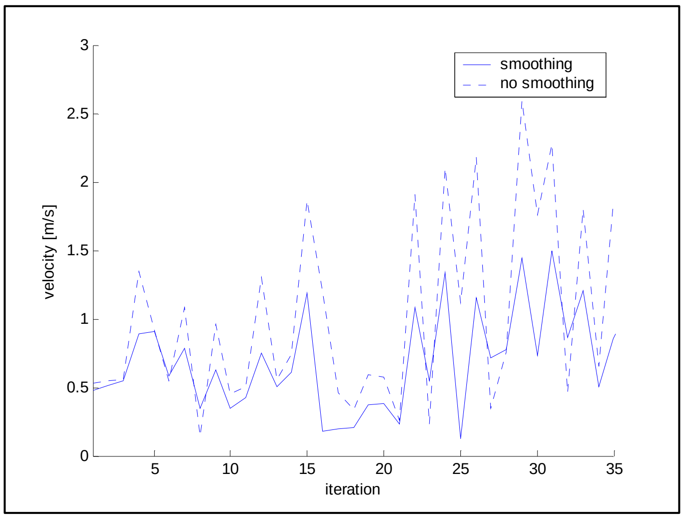

Figure 6.4: Velocity smoothing.

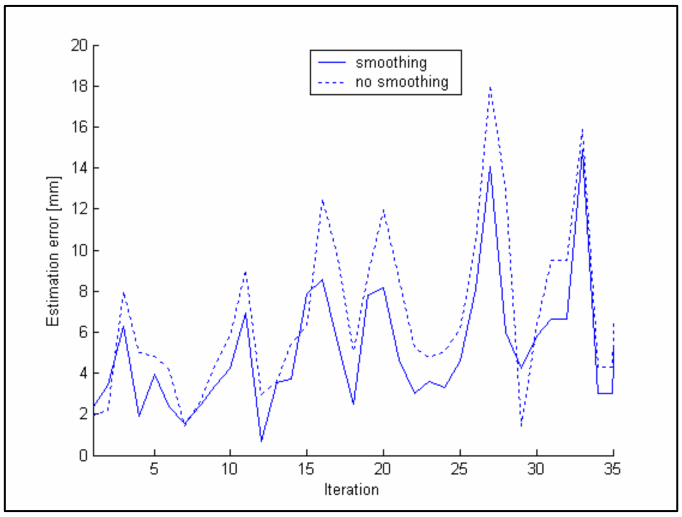

Figure 6.5: Estimation error with velocity smoothing.

6.3 Validation parameters

The following conditions are stated in order to test the tracking performance for different values of the validation parameters:

- Different objects' situations.
- R, Q and P_0 fixed to the values in section 6.1
- Radius fixed, 900 mm.

The quality factors to measure are the following:

- Outliers.
- Number of objects.
- Execution time.

The objective of this test is to gain a validation parameters range.

The object validation control is used in order to suppress outliers, as described in section 4.2.2. Figure 6.6 shows the effect of using the object validation control when there are outliers in the scene. The picture to the right is with the validation limit set to 5 iterations, the left one is without the object validation control.

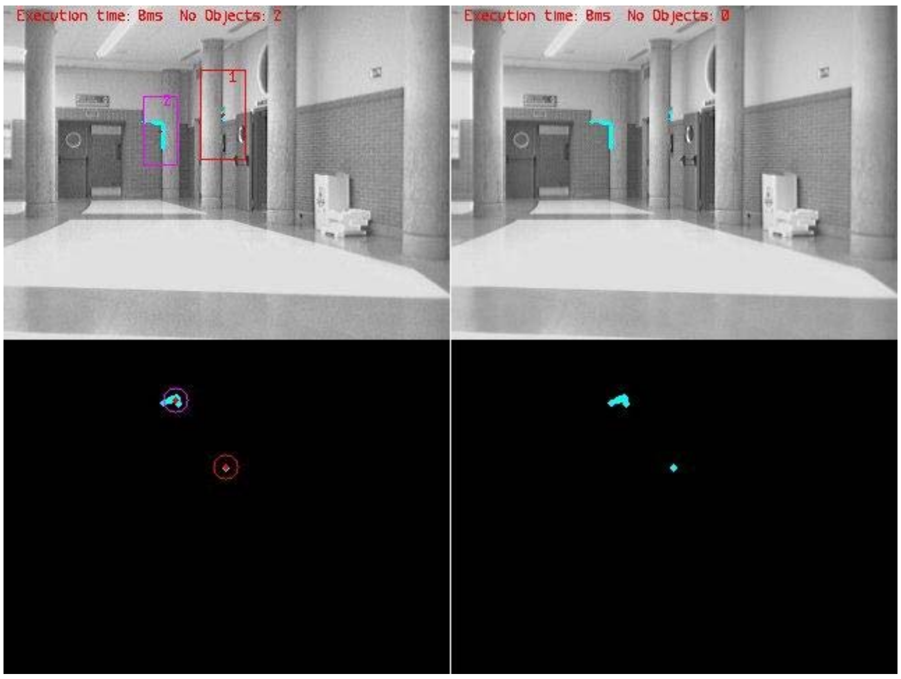
Execution time: 8ms No Objects: 2
2
1
Execution time: 8ms No Objects: 0

Figure 6.6: Two tracking results for the same scene. The picture to the right are the result obtained with the validation limit set to five iterations, and the one to the left is obtained without the validation control.

As described in section 4.2.2, in order to keep the tracking of an object while it is occluded invalidation limit has to be higher than the number of iterations it is occluded. In figure 6.7 the effect of the invalidation limit parameter is shown. In this experiment the object is occluded for 14 iterations and the invalidation parameter is set to 15 iterations, so the tracking is maintained.

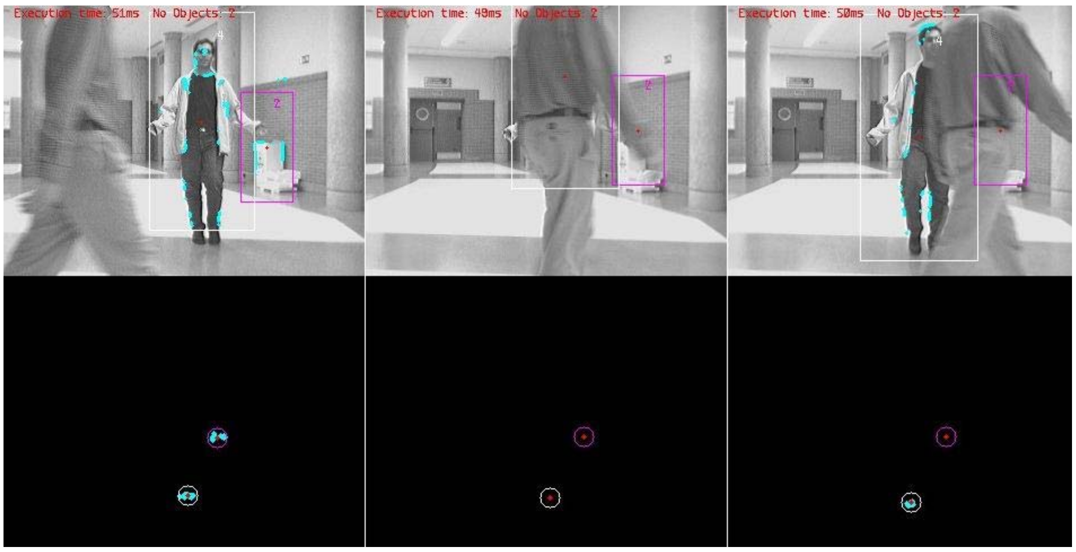
Execution time: 51ms No Objects: 2
Execution time: 49ms No Objects: 2
Execution time: 50ms No Objects: 2

Figure 6.7: An example of when an object is occluded, in this example the invalidation limit is set to 15 iterations.

The number of objects to track affects the execution time of the global algorithm, since there are more estimators to process when there are more objects to track. For this reason the validation process affects the execution time in the following way:

- No change when using the object validation functionality. Since the objects are processed in the global algorithm already when they are candidates.

- It increases when the invalidation limit is set higher. In the case where there are a lot of objects moving in and out of the scene, old estimators are kept longer with invalid limit high and meanwhile new ones are added. Resulting in many estimators to process.

In the experiments done to test the difference in the tracker execution time for different validation parameter settings, this value is not measurable since it is too low (mean execution time around 1ms).

A crossing is supposed to take less than 15 iterations (sample time 67 ms $\Rightarrow$1s). In other cases, for example when two objects are moving together and one gets occluded by the other one, the objects are considered as an only one. In this situation if one object starts moving away from the other it will appear as a new object.

Based on the statement above and the tests that are presented in this section the following validation parameters are used in the remaining experiments:

- When validation limit = 5 the object is validated.

- When invalidation limit = 15 the object is removed.

6.4 Cylinder radius

The following conditions are stated in order to test the tracking performance with different values for the cylinder radius:

- Different objects' situations.
- R, Q and P_0 fixed to the values in section 6.1.
- Validation parameters fixed, validation limit = 5 and invalidation limit = 15.

The quality factors to measure:

- Number of objects detected.
- Estimation error.
- Execution time.

The objective for this experiment is to find a range for the cylinder radius parameter.

Three different values of the cylinder radius have been tested: 600, 850 and 1000 mm. Figure 6.8 and 6.9 show the results of the three tests in two different situations. Figure 6.8 illustrates when the radius is set too big, then too few objects are found. However this particular situation is difficult since two persons' paths are crossed and one of them is occluded, only a few measurements are acquired from this person. Figure 6.9 shows an example where too many objects are detected when the radius is set small. The correct number of objects in both scenes is four; the number of objects at each study is shown by table 6.3.

Table 6.3: The number of objects detected in figure 6.9 and 6.10 for different sizes of the cylinder radius.

Radius	Detected objects in figure 6.9	Detected objects in figure 6.10
600 mm	4	7
850 mm	3	4
1000 mm	2	4

The mean estimation error during 40 iterations, in the middle of the field-view scene, with the different radius sizes has been tested and the results are shown in table 6.4. The estimation error with the different radii is not affected noticeable measuring the manual background truth.

Table 6.4: The mean estimation error for different sizes of the cylinder radius.

Radius (mm)	Estimation error (mm)
600	9.259
850	9.251
1000	9.256

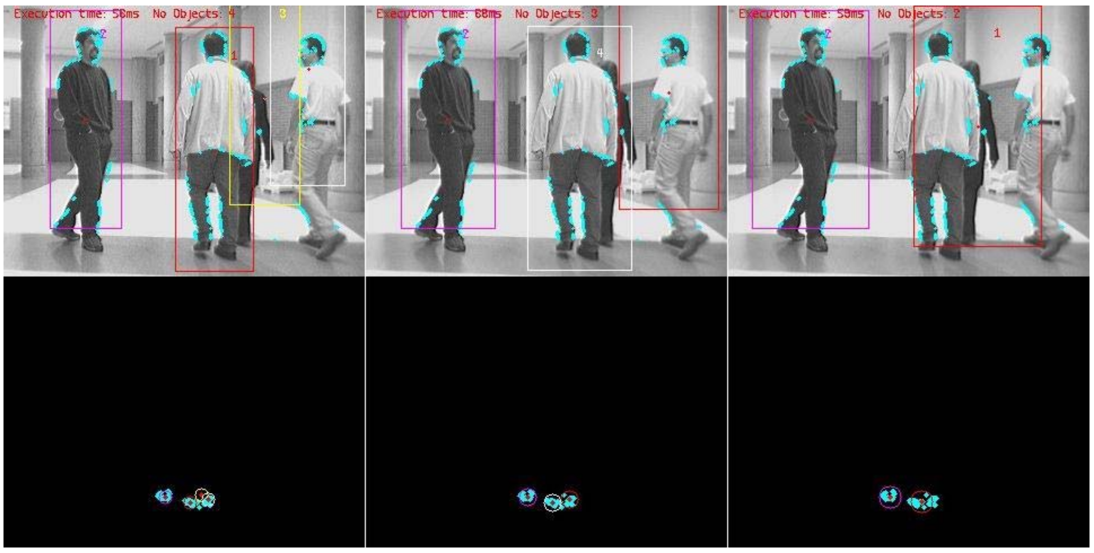

Figure 6.8: Result of a test with three different cylinder radiuses 600 mm to the left, 850mm in the middle and 1000 mm on the right.

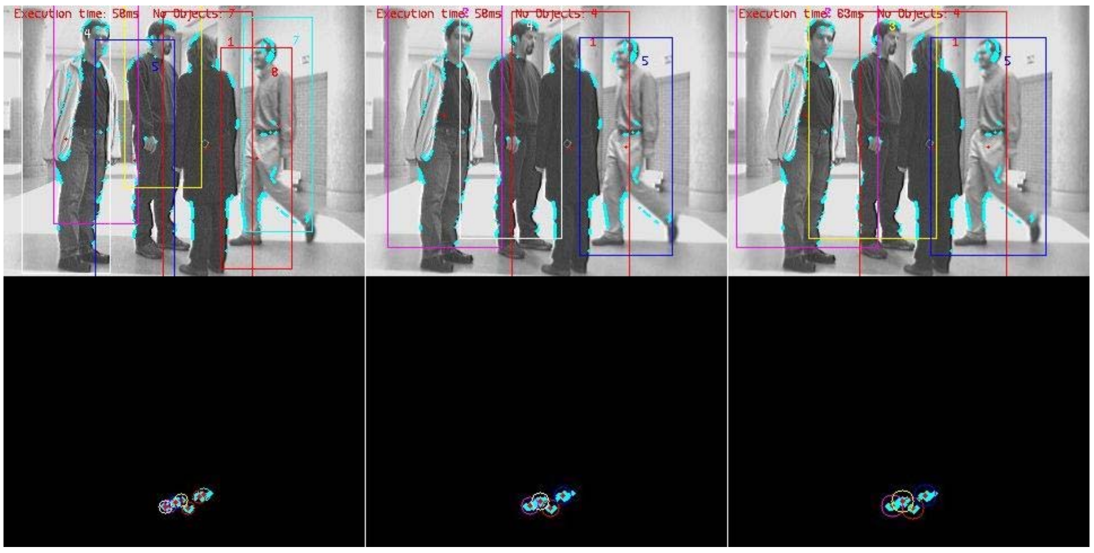

Figure 6.9: Result of a test with three different cylinder radiuses 600 mm to the left, 850mm in the middle and 1000 mm on the right.

The execution time increases when the cylinder radius is small, since more than one estimator is obtained for an only object. This effect cannot be measured due to the same reasons as described in section 6.3.

Based on these tests the cylinder radius is fixed to 850 mm.

6.5 Final results

In this chapter tests showing the performance of the tracking algorithm in different situations are shown. These tests are performed with the parameters tuned to the values in the previous sections.

6.5.1 One object

The result of a single object tracking task is shown in figure 6.10. A test has been performed during 35 iterations resulting in the following:

- A mean estimation error of: 4.464 mm.
- An estimation error covariance of: $\begin{bmatrix} 0.039 & 0 \\ 0 & 0.039 \end{bmatrix} mm^2$

This estimation error is very small but not very confident since it is not the background truth only the manual background truth that is measured.

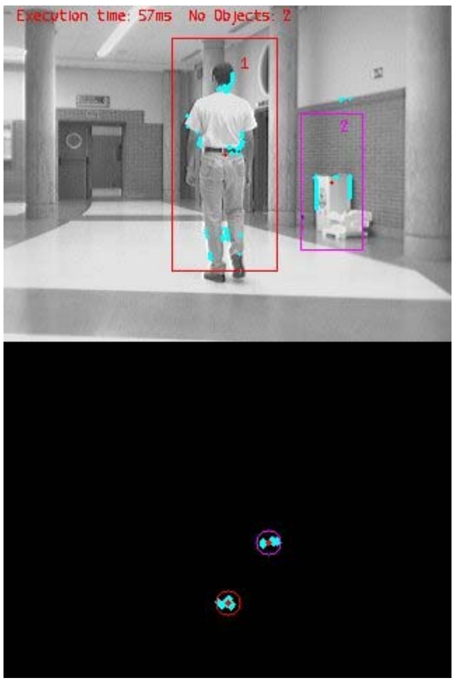

Figure 6.10: Single object tracking.

The resolution of the acquisition system decreases with z. This can lead to the behavior shown in figure 6.11, where two estimators are started to track an only object.

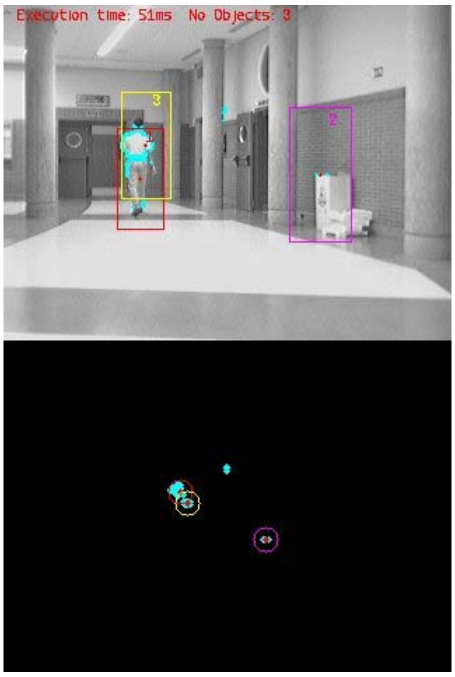

Figure 6.11: Single object tracking, two estimators started on the same object.

6.5.2 Multiple objects

Figure 6.12 below shows an example of a multi-object tracking task where the tracking algorithm’s performance is good. In this scene the five objects are tracked; one static object and four persons who are moving.

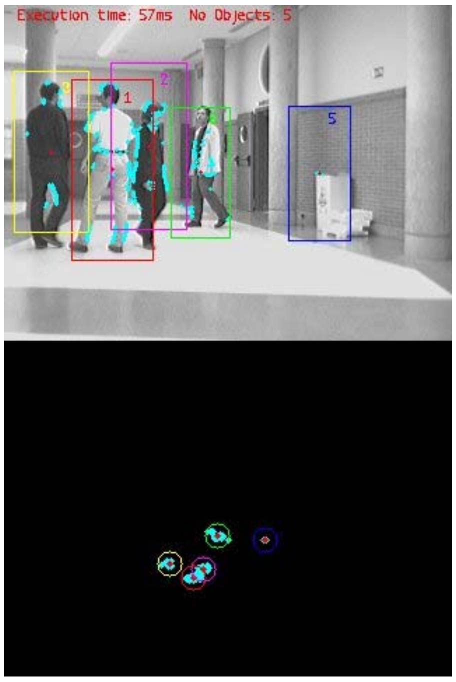

Figure 6.12: Multi-object tracking.

In figure 6.13 a less successful tracking process is shown. There are five persons in the scene. No measurements have been acquired from two of them, so it is impossible to track them. From the two other persons that stand closely three estimators are running, one tracker originates from the person moving out of the scene, which no more measurements is acquired from.

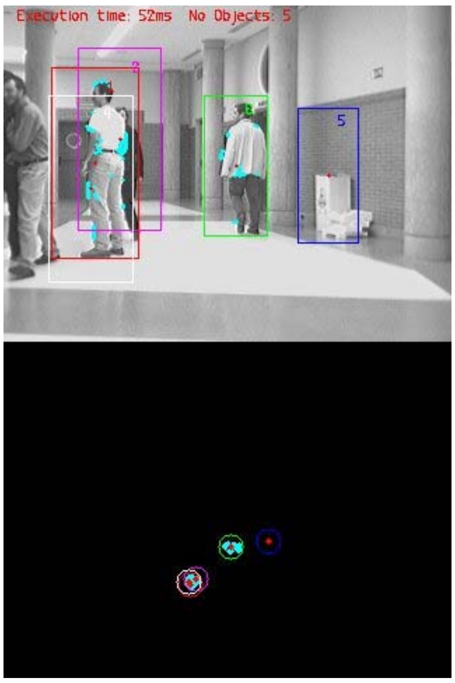

Figure 6.13: Multi-object tracking, less successful.

7 Conclusions and future work

7.1 Conclusions

An algorithm for multiple object tracking using Kalman filters has been implemented. The algorithm can be used in obstacle avoidance systems in autonomous robots, used in indoor environments. The tracking algorithm associates measurements, produced by a stereo-vision system, to the different estimators using the Nearest Neighborhood Data Association Algorithm.

Different performance tests in different situations have been presented. Considering that the manual background truth is used the results show that in most cases the tracking is performed well;

- The execution time demand is easily fulfilled.
- The number of objects can, in most cases, be obtained.
- The estimation error is small.

In complicated situations where there are many object crossing each others paths the tracking sometimes fails. The most complicated situation for the algorithm is when this happens far away from the measurement acquisition system, since the noise level of the produced measurements is higher there. The tracking failures might be because of the following two reasons:

1 The KF assumes all noise to be of Gaussian probability distribution. However, no tests identifying the input noise as Gaussian has been performed.

2 The NN, used for the association, does not give good results in high clutter density tracking environments, where there is more than one object and when tracks intersect.

In order to compensate the latter reason the cylinder radius, for which measurements are assumed to come from an object, has been set rather big. This can however lead to the misidentification of two objects as one.

7.2 Future work

In order to improve the performance of the tracking algorithm it is a good idea to use another data association algorithm. The NN was chosen because of its computationally inexpensiveness. However, since the execution time demands are fulfilled rather easily, a more advanced association algorithm could be afforded. Some are discussed in part I, such as the PDAF, which calculate probability between the association and each object and then uses it for a weighted update.

Another idea to improve the data association algorithm is to use a better segmentation method. The problem is that each time a new object is created all the measures should be reassigned since a previous assigned measure might actually be closer to the new object than to the one it has been assigned to. This would be a great improvement of the algorithm, but also time-consuming. Methods suggested for this can be found in [7].

An interesting thing to test would be to implement the tracking algorithm with an only estimator for tracking all the objects. The size of the estimator must in that case be changed dynamical since the number of objects is varying.

Since no noise identification has been made, to see if it is of Gaussian probability distribution, it is also of interest to test other estimation methods than the KF.

The parameters discussed in the algorithm (Q,R,P_0, cylinder radius and validation parameters) 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 cylinder radius could change with different sizes of objects.

III. User Manual

In this section all information needed to test the implementation of the presented tracking algorithm are described.

To be able to test the tracking algorithm some of the implementation assessments (see chapter IV) that are used in this project have to be used; a PC with a Linux operating system, with the openCV library installed.

The executable file *tracking* is built from the source code that can be found in chapter V. The file has three input arguments:

- *Argument 1* - decides whether or not to save the images showing the results of the tracking. The images will be stored in .JPG format. 0 = do not save, 1 = save.

- *Argument 2* – decides if to show a 2D likelihood histogram of the measurements, i.e. concentration of measurements in a discrete grid of the environment. 0 = no plot and 1 = plot.

- *Argument 3* – the number of the video file to test the tracking algorithm on.

The following video files are to choose from (argument 3):

- 001 – Two person walking towards the camera.
- 004 – Two person walking away from the camera.
- 010 – Many people in the scene, walking in different directions.
- 041 – One person walking away from the camera.
- 061 – One person walking towards the camera.

These files and the header file *tracking.h,* containing macros, must be in the same directory as the execution file, when executing.

To execute the file do the following:

- Open a terminal.
- Localize the directory of the execution file.
- Type ./tracking *arg.1 arg.2 arg.3* to run the tracking.

IV. Implementation assessments

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

1 Hardware

- **Stationary computer**

Microprocessor	Intel Pentium III
Speed	1.00 GHz
RAM	256 MB
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
Debian Linux	Kernel 2.6.5

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

- **Panda titanium antivirus 2005**
  Version 4.02.01.

- **OpenCV library**
  OpenCV-0.9.5.

- **Matlab**
  Version 6.5.0.180913a release 13.

V. Source code

In this section the source code for the implemented tracking algorithm is presented.

The following two files contain the code:

1. tracking.c - The realization of the tracking algorithm which is made up of the following functions:

• main	-	The global algorithm.
• sortData	-	The data association.
• mean	-	Objects' mean calculation.
• kman	-	Kalman filter.
• plotting	-	Used for drawing.
• hist	-	Plots a 2D likelihood histogram of the measurements.

2. tracking.h - Contains all macros used in the above functions.

1 tracking.c

```
// ================================================================
//                   Multi-object tracking
//                   By: Mikael Lindeborg
//
//                   tracking.c     2006-02-14
// ================================================================

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys/time.h>
#include <time.h>
#include "cv.h"
#include "highgui.h"
#include "tracking.h"

typedef struct{
       float   XZY[3];         // Coordinates
       int     Identify;       // -1=empty, 0=new object, 1=object
       int     remCand;        // Number of iterations without appearing
       int     addCand;        // 0 - no object candidate

}data;

int sortData(int ,float *,int *,int );
void meanCalc(int ,int, float *,int *);
void kman(float *,float,int);
void plotting(IplImage *,int ,int, float *,char *);
void hist(int ,float *,IplImage *);
void validation(int,int *);

// ============== Global variables =================================
data           *aCorrArr=NULL;
float          *aPredArr=NULL,*PPredArr=NULL,*mean=NULL,*PCorrArr=NULL,*velOld=NULL;
// ================================================================

int main( int argc,
        char *argv[] )
{
       IplImage
       *iplLeft,*iplLeftColor,*iplUndistLeftMap,*iplUndistLeft,*iplXZ,*iplBoth,
       *iplHist; //Images

       FILE           *pfLeft,*pfCoord;      //file pointers

       float          At=0,t=0,te=0;                 //time variables
       struct timeval tPrev,tNow,tExec;

       int            i,j,plotHist,saveImages;
       int            ny,hold=1,nFrame=1;
       int            objsAlloc=0,NoObjs=0;
       int            *objIndex=NULL;
       float          *pMeas=NULL,XYZ[3];
       char           text[MAX_STR_SIZE];

       saveImages = atoi(argv[1]);   //translate input arguments
       plotHist   = atoi(argv[2]);
       text[0]='\0';
       sprintf(text,"left%03d.str",atoi(argv[3]));
       pfLeft = fopen(text,"rb");
       sprintf(text,"data%03d.dat",atoi(argv[3]));
       pfCoord = fopen(text,"rb");

       //cvNamedWindow Creates a window(image placeholder)
       cvNamedWindow("1. Left 2D Image",CV_WINDOW_AUTOSIZE);
       if(plotHist)
              cvNamedWindow("2. Histogram",CV_WINDOW_AUTOSIZE);

       // Allocate space for images
       iplLeft        = cvCreateImage(cvSize(WIDTH,HEIGHT),IPL_DEPTH_8U,1);
       iplUndistLeft  = cvCreateImage(cvSize(WIDTH,HEIGHT),IPL_DEPTH_8U,1);
       iplLeftColor   = cvCreateImage(cvSize(WIDTH,HEIGHT),IPL_DEPTH_8U,3);
       iplUndistLeftMap= cvCreateImage(cvSize(WIDTH,HEIGHT),IPL_DEPTH_32S,3);
       iplXZ          = cvCreateImage(cvSize(WIDTH,HEIGHT),IPL_DEPTH_8U,3);
       iplBoth        = cvCreateImage(cvSize(WIDTH,HEIGHT*2),IPL_DEPTH_8U,3);
       iplHist        = cvCreateImage(cvSize(WIDTH,HEIGHT),IPL_DEPTH_8U,1);

       if (ferror(pfLeft))
       {
              printf("\nError when reading data...\n");
              fclose(pfLeft);
       }
```

```
	if (ferror(pfCoord))
	{
		printf("\nError when reading data...\n");
		fclose(pfCoord);
	}

	//The first value in the string is for the undistortionmap initialization
	if(!fread(iplLeft->imageData,1,FILE_SIZE,pfLeft))
	{
		printf("\nEnd of video\n");
		return(-1);
	}

	//Calculates arrays of distorted points indices and interpolation
	// coefficients, using known matrix of the camera intrinsic parameters and
	// distortion coefficients.
	cvUnDistortInit(iplLeft,iplUndistLeftMap,intrMatrixLeft,distCoeffsLeft,1);

	if(!fread(&ny,sizeof(int),1,pfCoord)) //N^o coordinates are read into ny
	{
		printf("\nNo more coordinates to read from file...\n");
		return(-1);
	}

	//Allocate space for the coordinates in an array
	pMeas=(float*)malloc(3*ny*sizeof(float));
	objIndex=(int*)malloc(ny*sizeof(int));
	fread(pMeas,sizeof(float),3*ny,pfCoord);

	if(gettimeofday(&tNow,NULL)) //start clock
		printf("Error in gettimeofday() !!\n");
	do
	{
		if (hold==1)
		{
			if(!fread(iplLeft->imageData,1,FILE_SIZE,pfLeft))
			{
				printf("\nEnd of video\n");
				break;
			}
			//cvUnDistort corrects camera lens distortion using previously
			// calculated undistortion map
			cvUnDistort(iplLeft,iplUndistLeft,iplUndistLeftMap,1);
			//cvCvtColor converts input img rom one color space to another
			cvCvtColor(iplUndistLeft,iplLeftColor,CV_GRAY2BGR);

			//Read coordinates
			if(!fread(&ny,sizeof(int),1,pfCoord))		{
				printf("\nThe End\n");
				break;
			}
			//Reallocate space for the coords
			pMeas=(float*)realloc(pMeas,3*ny*sizeof(float));
			objIndex=(int*)realloc(objIndex,ny*sizeof(int));
			fread(pMeas,sizeof(float),3*ny,pfCoord);

			objsAlloc=sortData(ny,pMeas,objIndex,objsAlloc);
			meanCalc(ny,objsAlloc,pMeas,objIndex);
			kman(mean,At,objsAlloc);
			validation(objsAlloc,&NoObjs);


			//Calculate execution time
			if(gettimeofday(&tExec,NULL))
				printf("Error in gettimeofday() !!\n");
			te=(tExec.tv_sec-tNow.tv_sec)*1000+(tExec.tv_usec-
tNow.tv_usec)/1000;
			cvZero(iplXZ); //clear image
			//Plot measurements
			for(i=0;i<(3*ny);i=i+3)
			{
				XYZ[0]=pMeas[i];
				XYZ[1]=pMeas[i+2];
				XYZ[2]=pMeas[i+1];
				//Plot in video
				plotting(iplLeftColor,DOTUV,TURQUOISE,XYZ,NULL);
				//XZ plot
				plotting(iplXZ,DOTXZ,TURQUOISE,XYZ,NULL);
			}

			if(plotHist)//if a "histogram" plot is wanted
				hist(ny,pMeas,iplHist);

			//plot kalman estimations
			for(i=0;i<objsAlloc;i++)
			{
```

```
                        //plot only validated objs
                        if(aCorrArr[i].Identify==1 && !aCorrArr[i].addCand)
                        {
                                XYZ[0]=aCorrArr[i].XZY[0];
                                XYZ[1]=aCorrArr[i].XZY[2];
                                XYZ[2]=aCorrArr[i].XZY[1];
                                plotting(iplLeftColor,DOTUV,RED,XYZ,NULL);
                                // %7 - 7 different colors
                                plotting(iplLeftColor,RECT,(RED+i)%7,XYZ,NULL);
                                sprintf(text,"%d",i+1);
                                plotting(iplLeftColor,TEXT,(RED+i)%7,XYZ,text);
                                plotting(iplXZ,DOTXZ,RED,XYZ,NULL);
                                plotting(iplXZ,CIRC,(RED+i)%7,XYZ,NULL);
                        }
                }
        }

        //cvWaitKey- Waits for pressed key
        if (cvWaitKey(10)>=0)
        {
                if (hold==1) hold = 2;
                else hold = 1;
        }

        tPrev=tNow;
        At=0;
        while(At < ts)//delay get the right sample time
        {
                if(gettimeofday(&tNow,NULL))
                        printf("Error in gettimeofday() !!\n");
                At=(tNow.tv_sec-tPrev.tv_sec)*1000+(tNow.tv_usec-
tPrev.tv_usec)/1000;
        }

        sprintf(text,"Execution time: %.0fms  No Objects: %d",te,NoObjs);
        //Print the execution time
        plotting(iplLeftColor,TEXT,RED,NULL,text);

        memcpy(iplBoth->imageData,iplLeftColor->imageData,3*FILE_SIZE);
        memcpy(iplBoth->imageData+3*FILE_SIZE,iplXZ->imageData,3*FILE_SIZE);

        if(saveImages) //Save images
        {
                sprintf(text,"Result%05d.jpg",nFrame);
                cvSaveImage(text,iplBoth);

        }
        nFrame++;

        cvShowImage("1. Left 2D Image",iplBoth);
        if(plotHist)
                cvShowImage("2. Histogram",iplHist);

    }while(1);


    free(pMeas);                //free allocated heap space
    free(objIndex);
    free(mean);
    free(velOld);
    free(aCorrArr);
    free(aPredArr);
    free(PPredArr);
    free(PCorrArr);

    cvReleaseImage(&iplLeft);     //releases header and image data
    cvReleaseImage(&iplUndistLeft);
    cvReleaseImage(&iplUndistLeftMap);
    cvReleaseImage(&iplLeftColor);
    cvReleaseImage(&iplXZ);
    cvReleaseImage(&iplBoth);
    cvReleaseImage(&iplHist);
    cvDestroyWindow("1. Left 2D Image");  //destroys windows
    if(plotHist)
        cvDestroyWindow("2. Histogram");

    fclose(pfLeft);             //close files
    fclose(pfCoord);

  return 0;
}


// ==================================================================
//      Function: SortData
//
//      Purpose : Assign measurements to objects
//      Input   : ny - N^o measurements
//                pMeas - Measurements
//                objsAlloc - N^o excisting allocated obj
//                aPredArr - KF-predicted pos. of objs
//      Output  : objIndex - Meas to Objs Association vector
//                objsAlloc -N^o allocated object's storage room

// ====================================================================

int sortData(int ny,float *pMeas,int *objIndex,int objsAlloc)
{
        int             i,j,k,p,minIndex,fill=0,newObj=1;
        float           CenterDist,minDist,*temp;

        while(newObj)
        {
                p=0;
                newObj=0;
                for(j=0;j<ny;j++)//loop through all meas
                {
                        minDist=3000.0; //big number for first iter
                        for(i=0;i<objsAlloc;i++)//Checks which object is closest
                        {
                                //only the "active" objects
                                if(aCorrArr[i].Identify!=-1)
                                        //euclidean distance calc
                                        CenterDist=sqrt( powf((pMeas[j*3]-
aPredArr[i*3]),2)+powf((pMeas[j*3+1]-aPredArr[i*3+1]),2) );
                                else
                                        CenterDist=4000.0;

                                if(CenterDist<minDist)
                                {
                                        minIndex=i;
                                        minDist=CenterDist;
                                }

                        }
                        //Check if minDist is too big, then create new object
                        // otherwise set the right index
                        if(minDist>RMAX)        //new object
                        {
                                k=0;fill=0;
                                newObj=1;       //set to restart the allocation
                                while(k<objsAlloc)
                                {
                                        //overwrite old object
                                        if(aCorrArr[k].Identify==-1)
                                        {
                                                fill=1;
                                                objIndex[p++]=k-1;
                                                aCorrArr[k].Identify=0;
                                                aCorrArr[k].addCand=1;
                                                aCorrArr[k].remCand=1;
                                                for(i=0;i<3;i++)
                                                        aPredArr[k*3+i]=pMeas[j*3+i];
                                                k=objsAlloc;//break
                                                j=ny;//break to restart the allocation

                                        }
                                        k++;
                                }
                                if(!fill)//reallocate for new  object
                                {
                                        objsAlloc++;//update the counter for new object
                                                aCorrArr = (data*)realloc(aCorrArr,
objsAlloc*sizeof(data));
                                        aPredArr = (float*)realloc(aPredArr,
objsAlloc*3*sizeof(float));
                                        PPredArr = (float*)realloc(PPredArr,
objsAlloc*9* sizeof(float));
                                        mean = (float*)realloc(mean,
objsAlloc*3*sizeof(float));
                                        PCorrArr = (float*)realloc(PCorrArr,
objsAlloc*9*sizeof(float));
                                        velOld = (float*)realloc(velOld,
objsAlloc*3*sizeof(float));


                                        objIndex[p++]=(objsAlloc-1);
                                        aCorrArr[objsAlloc-1].Identify=0;
                                        aCorrArr[objsAlloc-1].addCand=1;
                                        aCorrArr[objsAlloc-1].remCand=1;

                                        for(i=0;i<3;i++)
                                                //set value too compare with
                                                aPredArr[(objsAlloc-1)*3+i]=
pMeas[j*3+i];
```

```
                                j=ny;//break to restart the allocation
                        }
                }
                else
                {
                        objIndex[p++]=minIndex;
                }
        }
    }
    //Check if there were objects not appering
    for(i=0;i<ny;i++)
            aCorrArr[(objIndex[i])].remCand=0;

    return(objsAlloc);
}

// ==================================================================
//      Function: meanCalc
//      Purpose : Calculate mean of each objs meas
//      Input   : ny - N^o measurements
//                pMeas - Measurements
//                objsAlloc - N^o allocated object space
//                objIndex - Meas to Objs Association vector
//      Output  : mean - Vector containing objs mean
// ==================================================================

void meanCalc(int ny,int objsAlloc,float *pMeas,int *objIndex)
{
        int             i,j,objCount;
        float           sumX,sumY,sumZ;

        for(i=0;i<objsAlloc;i++)//calculate measured centers
        {
                //only when meas has been found
                if(aCorrArr[i].remCand==0 && aCorrArr[i].Identify!=-1 && ny!=0)
                {
                        sumX=0.0;sumY=0.0;sumZ=0.0;objCount=0;
                        for(j=0;j<ny;j++)
                        {
                                if(objIndex[j]==i)
                                {
                                        sumX=sumX+pMeas[j*3];
                                        sumY=sumY+pMeas[j*3+1];
                                        sumZ=sumZ+pMeas[j*3+2];
                                        objCount++;
                                }
                        }
                        mean[i*3]=sumX/(float(objCount));
                        mean[i*3+1]=sumY/(float(objCount));
                        mean[i*3+2]=sumZ/(float(objCount));
                }
        }

}

// ==================================================================
//      Function: kman
//      Purpose : Calculate kalman estimations
//      Input   : aCorrArr - Prev. position correction
//                aPredArr - Position prediction
//                PPredArr - Error cov. prediction
//                objsAlloc - No allocated obj spacr
//                objIndex - Meas to Objs Association vector
//      Output  : aCorrArr - Corrected position
//                aPredArr - Position prediction for next iter
//                PPredArr - Err cov. prediction for next iteration
// ==================================================================

void kman(float *mean,float At,int objsAlloc)
{
        int             i,j;
        float           Itemp[9]={1.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0},Rtemp[9];
        float           Qtemp[9],arr[9];;
        float           T[9]={ts,0.0,0.0,0.0,ts,0.0,0.0,0.0,ts};
        CvMat
        *aPredict,*aCorrect,*PPredict,*PCorrect,*Velo,*I,*temp1,*temp2,*temp3,*temp4,
*K,*Q,*R,*M,*H;

        aPredict=cvCreateMat(3,1,CV_32F);
        aCorrect=cvCreateMat(3,1,CV_32F);
        PPredict=cvCreateMat(3,3,CV_32F);
        PCorrect=cvCreateMat(3,3,CV_32F);
        I=cvCreateMat(3,3,CV_32F);
        temp1=cvCreateMat(3,3,CV_32F);
        temp2=cvCreateMat(3,1,CV_32F);
        temp3=cvCreateMat(3,1,CV_32F);
        temp4=cvCreateMat(3,3,CV_32F);
        Velo=cvCreateMat(3,1,CV_32F);
        K=cvCreateMat(3,3,CV_32F);
        R=cvCreateMat(3,3,CV_32F);
        M=cvCreateMat(3,1,CV_32F);
        Q=cvCreateMat(3,3,CV_32F);
        H=cvCreateMat(3,3,CV_32F);
        for(i=0;i<9;i++)
        {
                Qtemp[i]=Itemp[i]*1.0;
                Rtemp[i]=0.0;
        }
        Rtemp[0]=0.038;
        Rtemp[5]=0.038;
        Rtemp[8]=0.014;

        cvInitMatHeader(H,3,3,CV_32F,T);
        cvInitMatHeader(I,3,3,CV_32F,Itemp);
        cvInitMatHeader(R,3,3,CV_32F,Rtemp);
        cvInitMatHeader(Q,3,3,CV_32F,Qtemp);

// ================================================
//      Correction
// ================================================
        for(i=0;i<objsAlloc;i++)
        {
                if(aCorrArr[i].Identify==0) //if first appearence set X0 and P0
                {
                        aCorrArr[i].XZY[0]=mean[i*3]; // X0
                        aCorrArr[i].XZY[1]=mean[i*3+1];
                        aCorrArr[i].XZY[2]=mean[i*3+2];
                        aCorrArr[i].Identify=1;
                        aPredArr[i*3]=mean[i*3]; //X0+V0, V0=0
                        aPredArr[i*3+1]=mean[i*3+1];
                        aPredArr[i*3+2]=mean[i*3+2];

                        for(j=0;j<9;j++)        //P0
                                PPredArr[i*9+j]=Itemp[j];
                }
                else if(aCorrArr[i].Identify==1) // if it has “appeared” this iter
                {

                        aPredict->data.fl[0]=aPredArr[i*3];
                        aPredict->data.fl[1]=aPredArr[i*3+1];
                        aPredict->data.fl[2]=aPredArr[i*3+2];

                        aCorrect->data.fl[0]=aCorrArr[i].XZY[0];
                        aCorrect->data.fl[1]=aCorrArr[i].XZY[1];
                        aCorrect->data.fl[2]=aCorrArr[i].XZY[2];
                        arr[9];
                        M->data.fl[0] = mean[i*3];
                        M->data.fl[1] = mean[i*3+1];
                        M->data.fl[2] = mean[i*3+2];

                        for(j=0;j<9;j++)
                        {
                                arr[j]=PPredArr[i*3+j];
                        }
                        cvInitMatHeader(PPredict,3,3,CV_32F,arr);

                        //Correction only when measurments have been found.
                        if(aCorrArr[i].remCand==0)
                        {
                                //Calculate velo for prediction before old val. is
                                // overwritten.
                                Velo->data.fl[0]=(M->data.fl[0]-aCorrect->
data.fl[0])/At;
                                Velo->data.fl[1]=(M->data.fl[1]-aCorrect->
data.fl[1])/At;
                                Velo->data.fl[2]=(M->data.fl[2]-aCorrect->
data.fl[2])/At;

                                // Velocity smoothing
                                Velo->data.fl[0] = (Velo->data.fl[0]+velOld[i*3])
/2.0;
                                Velo->data.fl[1] = (Velo->data.fl[1]+velOld[i*3+1])
/2.0;
                                Velo->data.fl[2] = (Velo->data.fl[2]+velOld[i*3+2])
/2.0;

                                cvAdd(PPredict,R,temp1);
                                cvInv(temp1,temp4,CV_LU);
                                cvMatMul(PPredict,temp4,K);//Correction equation (1)

                                cvSub(M,aPredict,temp2);
                                cvMatMul(K,temp2,temp3);
                                cvAdd(aPredict,temp3,aCorrect);//Correction eq (2)
                                cvSub(I,K,temp1);
                                cvMatMul(temp1,PPredict,PCorrect);//Correction eq (3)

                                aCorrArr[i].XZY[0]=aCorrect->data.fl[0];// Save values
                                aCorrArr[i].XZY[1]=aCorrect->data.fl[1];
                                aCorrArr[i].XZY[2]=aCorrect->data.fl[2];
                                for(j=0;j<9;j++)
                                        PCorrArr[i*9+j]=PCorrect->data.fl[j];
                                velOld[i*3]=Velo->data.fl[0];
                                velOld[i*3+1]=Velo->data.fl[1];
                                velOld[i*3+2]=Velo->data.fl[2];

                        }
                        else //maintain old velocity if it disapperas
                        {
                                Velo->data.fl[0]=velOld[i*3];
                                Velo->data.fl[1]=velOld[i*3+1];
                                Velo->data.fl[2]=velOld[i*3+2];
                        }
// ================================================
//      Prediction
// ================================================

                        cvMatMul(H,Velo,temp2);
                        if(aCorrArr[i].remCand!=0)
                                cvAdd(aPredict,temp2,aPredict);//prediction eq (1)
                        else
                                cvAdd(aCorrect,temp2,aPredict);//prediction eq (1)
                        cvAdd(Q,PCorrect,PPredict);             //prediction eq (2)
                        aPredArr[i*3]=aPredict->data.fl[0];     //Save values
                        aPredArr[i*3+1]=aPredict->data.fl[1];
                        aPredArr[i*3+2]=aPredict->data.fl[2];
                        for(j=0;j<9;j++)
                                PPredArr[i*9+j]=PPredict->data.fl[j];
                }
            }

// ================================================
                cvReleaseMat(&aPredict);
                cvReleaseMat(&aCorrect);
                cvReleaseMat(&PPredict);
                cvReleaseMat(&PCorrect);

                cvReleaseMat(&temp1);
                cvReleaseMat(&temp2);
                cvReleaseMat(&temp3);
                cvReleaseMat(&temp4);
                cvReleaseMat(&Velo);
                cvReleaseMat(&I);
                cvReleaseMat(&K);
                cvReleaseMat(&R);
                cvReleaseMat(&M);
                cvReleaseMat(&Q);
                cvReleaseMat(&H);
}



// ================================================================
//      Function: validation
//      Purpose : remove or add object candidates
//      Input   : aCorrArr - Objects
//                objsAlloc - No allocated obj space
//                NoObjs    - Number of validated objects
//      Output  : aCorrArr - Objects
//                NoObjs    - Number of validated objects
// ================================================================
void validation(int objsAlloc,int *NoObjs)
{
                int             i;

                for(i=0;i<objsAlloc;i++)//loop trough all objs
                {
                //validate as object
                if(aCorrArr[i].addCand==ITERADD && aCorrArr[i].remCand==0)
                {
                        aCorrArr[i].addCand=0;
                        (*NoObjs)++;
                }
                //still a candidate
                else if(aCorrArr[i].addCand!=0 && aCorrArr[i].remCand==0)
                {
                        aCorrArr[i].addCand++;
                }
                //invalid candidate
                else if(aCorrArr[i].addCand!=0 && aCorrArr[i].remCand!=0)
                {
                        aCorrArr[i].addCand=0;          //empty
                        aCorrArr[i].Identify=-1;
                        aCorrArr[i].remCand=0;
                }
                //if x iterations without appearing, delete object
                if(aCorrArr[i].remCand>=ITERREM)
                {
                        aCorrArr[i].Identify=-1;
                        aCorrArr[i].remCand=0;
                        aCorrArr[i].addCand=0;
                        (*NoObjs)--;
                }
                //will be reseted in next iter if it appears
                if(aCorrArr[i].Identify!=-1)
                        aCorrArr[i].remCand++;
        }

}


// =================================================================
//      Function: plotting
//
//      Purpose : Plot
//      Input   : *ipl - Pointer to image in which to plot
//                type - Type of plot to be made
//                color - Wanted color of plot
//                XYX - Vector containing coordinates to plot
// =================================================================
void plotting(IplImage *ipl,int type,int color,float *XYZ,char *text)
{
        int             i,j;
        int             U,V,X1,Y1,X2,Y2;
        float           arr[3];
        double          setColor;
        CvMat           *matRotSCC,*matCoords,*matSCC;
        CvFont          font;
        switch(color)
        {
                case BLACK:
                        setColor=CV_RGB(0,0,0);
                        break;
                case BLUE:
                        setColor=CV_RGB(0,0,255);
                        break;
                case GREEN:
                        setColor=CV_RGB(0,255,0);
                        break;
                case TURQUOISE:
                        setColor=CV_RGB(0,255,255);
                        break;
                case RED:
                        setColor=CV_RGB(255,0,0);
                        break;
                case MAGNETA:
                        setColor=CV_RGB(255,0,255);
                        break;
                case YELLOW:
                        setColor=CV_RGB(255,255,0);
                        break;
                case WHITE:
                        setColor=CV_RGB(255,255,255);
                        break;
                default:
                        printf("Unknown color");
        }

        matRotSCC=cvCreateMat(3,3,CV_32F);              //Creates new matrix.
        cvInitMatHeader(matRotSCC,3,3,CV_32F,RotationSCC);//rotation matrix.
        matSCC=cvCreateMat(3,1,CV_32F);
        matCoords=cvCreateMat(3,1,CV_32F);

        switch(type)
        {
                case DOTUV:
                        arr[0]=XYZ[0];
                        arr[1]=TRAS_Y-XYZ[1];
                        arr[2]=XYZ[2];
                        cvInitMatHeader(matCoords,3,1,CV_32F,&arr);
                        cvMatMulAdd(matRotSCC,matCoords,0,matSCC);//make SCC matrix
                        U=(int)round(((float)FXL*(matSCC->data.fl[0]/matSCC->
data.fl[2]))+U0L); //x' to u
                        V=(int)round(((float)FYL*(matSCC->data.fl[1]/matSCC->
data.fl[2]))+V0L);//y' to v
                        cvCircle(ipl,cvPoint(U,V),1,setColor,-1);
                        break;
```

```
            case DOTXZ:
                   X1=(int)(round( (XYZ[0]+(float)XMAX) / CONVX ));
                   Y1=(int)(round( ((float)ZMAX-XYZ[2])/ CONVZXZ));
                   cvCircle(ipl,cvPoint(X1,Y1),2,setColor,-1);
                   break;

            case CIRC:
                   X1=(int)(round( (XYZ[0]+(float)XMAX) / CONVX ));
                   Y1=(int)(round( ((float)ZMAX-XYZ[2])/ CONVZXZ));
                   X2=(int)(round( RMAX/(2.0*CONVX) ));
                   cvCircle(ipl,cvPoint(X1,Y1),X2,setColor,1);
                   break;

            case RECT:
                   arr[0]=XYZ[0]-(RMAX/2.0);
                   arr[1]=-XYZ[1];
                   arr[2]=XYZ[2];
                   cvInitMatHeader(matCoords,3,1,CV_32F,&arr);
                   cvMatMul(matRotSCC,matCoords,matSCC);
                   X1=(int)round(((float)FXL*(matSCC->data.fl[0]/matSCC->
data.fl[2]))+U0L); //x' to u
                   Y1=(int)round(((float)FYL*(matSCC->data.fl[1]/matSCC->
data.fl[2]))+V0L);//y' to v
                   arr[0]=XYZ[0]+(RMAX/2.0);
                   arr[1]=2*TRAS_Y-XYZ[1];
                   arr[2]=XYZ[2];
                   cvInitMatHeader(matCoords,3,1,CV_32F,&arr);
                   cvMatMul(matRotSCC,matCoords,matSCC);
                   X2=(int)round(((float)FXL*(matSCC->data.fl[0]/matSCC->
data.fl[2]))+U0L); //x' to u
                   Y2=(int)round(((float)FYL*(matSCC->data.fl[1]/matSCC->
data.fl[2]))+V0L);//y' to v
                   cvRectangle(ipl,cvPoint(X1,Y1),cvPoint(X2,Y2),setColor,1);
                   break;

            case TEXT:
                   if(XYZ!=NULL)
                   {
                          arr[0]=XYZ[0]+150.0;
                          arr[1]=-XYZ[1]+250.0;
                          arr[2]=XYZ[2];
                          cvInitMatHeader(matCoords,3,1,CV_32F,&arr);
                          cvMatMul(matRotSCC,matCoords,matSCC);
                          U=(int)round(((float)FXL*(matSCC->data.fl[0]/matSCC->
data.fl[2]))+U0L); //x' to u
                          V=(int)round(((float)FYL*(matSCC->data.fl[1]/matSCC->
data.fl[2]))+V0L);//y' to v
                   }
                   else
                   {
                          U=10;
                          V=10;
                   }
                   cvInitFont(&font,CV_FONT_VECTOR0,0.3f,0.3f,0.0f,1);
                   cvPutText(ipl,text,cvPoint(U,V),&font,setColor);
                   break;

            default:
                   printf("\nNot a valid plotting type....\n");
     }

     cvReleaseMat(&matRotSCC);
     cvReleaseMat(&matCoords);
     cvReleaseMat(&matSCC);

}

// =================================================================
//     Function: hist
//     Purpose : Plot measurement histogram (2D)
//     Input   : iplHist - image to plot the hist. in
//               ny      - Number of measurements
//               pMeas   - Measurements
//               NoObjs   - Number of validated objects
// =================================================================

void hist(int ny,float *pMeas,IplImage *iplHist)
{
     int            i,j,X,Z,X1,X2,Y1,Y2;
     int            max=0;
     int            histMat[40][32];
     float          temp;

     for(i=0;i<40;i++)//rows
     {
            for(j=0;j<32;j++)//columns
            {
```

```
                        histMat[i][j]=0;
                }
        }
        for(i=0;i<ny;i++)
        {
                X = (int)floorf(((pMeas[i*3]+8000.0)/500.0));
                Z = (int)floorf(((pMeas[i*3+1]-500.0)/500.0));
                histMat[Z][X]++;
                if(histMat[Z][X]>max)
                        max=histMat[Z][X];
        }
        cvRectangle(iplHist,cvPoint(0,0),cvPoint(320,240),255,-1);//white
        for(i=0;i<40;i++)//rows
        {
                for(j=0;j<32;j++)//columns
                {
                        if(max)
                                temp=((float)histMat[i][j]/(float)max);
                        else
                                temp=0.0;
                        histMat[i][j]= (int)round(((1-temp)*255)); //sets greyscale
                        X1=j*10;
                        Y1=(40-i)*6;   // 6=240/40
                        X2=X1-10;
                        Y2=Y1-6;

        cvRectangle(iplHist,cvPoint(X1,Y1),cvPoint(X2,Y2),histMat[i][j],-1);
                }
        }
}
```

2 tracking.h

```
/* -------------------------------------------- */
#define TRUE 1
#define FALSE 0
/* -------------------------------------------- */
/* -------------------------------------------- */
/* Image parameters                          */
#define WIDTH 320
#define HEIGHT 240
#define FILE_SIZE (WIDTH*HEIGHT)
/* -------------------------------------------- */
/* -------------------------------------------- */
/* Intrinsic parameters, left camera         */
#define FXL 430.79014
#define FYL 431.72027
#define U0L 151.26555
#define V0L 117.03242
/* -------------------------------------------- */
/* -------------------------------------------- */
/* Matrix calibration of the intrinsic cameras- */
float intrMatrixLeft[9] = {FXL,0.0,U0L,0.0,FYL,V0L,0.0,0.0,1.0};
float distCoeffsLeft[4] = {-0.06108,-0.14348,0.00345,-0.00526};

/* -------------------------------------------- */
/* -------------------------------------------- */
/* Camera rotation angels i radians          */
#define alpha (0.94972*3.14159/180)   // X-axis rotation
#define beta 0.019508         // Y-axis rotation
#define phi -0.014053         // Z-axis rotation
/* -------------------------------------------- */
/* -------------------------------------------- */
/* Rotation matrix                           */
float RotationSCC[9] = {cos(beta)*cos(phi),-
cos(beta)*sin(phi),sin(beta),sin(alpha)*sin(beta)*cos(phi)+cos(alpha)*sin(phi),sin(a
lpha)*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)};
/* -------------------------------------------- */
/* -------------------------------------------- */
/* translation of the Y-axis                 */
#define TRAS_Y 970
/* -------------------------------------------- */
/* -------------------------------------------- */
/* Borders X-Y-Z (in mm)                     */
#define XMIN -8000
#define XMAX 8000
#define YMIN 100
#define YMAX 2100
#define ZMIN 500
#define ZMAX 20500
#define ZMAX2 16500
/* -------------------------------------------- */
/* -------------------------------------------- */
/* Transformation constants                  */
#define CONVX  ((float)(XMAX-XMIN)/WIDTH)
#define CONVZXZ ((float)(ZMAX-ZMIN)/HEIGHT)
#define CONVZXZ2 ((float)(ZMAX2-ZMIN)/HEIGHT)
#define CONVY  ((float)(YMAX-YMIN)/HEIGHT)
/* -------------------------------------------- */
/* -------------------------------------------- */
/* Object parameters                         */
#define RMAX           850.0
#define ITERREM                15
#define ITERADD        5
/* -------------------------------------------- */
/* -------------------------------------------- */
/* Object representation                     */
#define WHITE          0
#define BLUE           1
#define GREEN          2
#define TURQUOISE      3
#define RED            4
#define MAGNETA                5
#define YELLOW         6
#define BLACK          7

#define DOTUV          0
#define DOTXZ          1
#define CIRC           2
#define RECT           3
#define TEXT           4
/* -------------------------------------------- */
/* -------------------------------------------- */
/* Sample time                               */
```

```
#define ts             66.0
/* --------------------------------------------- */
/* --------------------------------------------- */
/* String                                     */
#define MAX_STR_SIZE   40
```

VI. Budget

In this section the different costs for this project are described.

- **Cost for laboratory equipment**

Items	**Cost per hour**	**Hours of usage**	**Total**
PC	0.4 €	920 h	368 €
Software for PC	1.4 €	920 h	1288 €
Printer paper	-	-	10 €

Total cost for laboratory equipment	**1666.00 €**

- **Cost for manual work**

Function	**Number of hours**	**€/h**	**Total**
Engineering	700	60.00	42000 €
Writing	220	12.00	2640 €

Total cost for manual work	**44640 €**

- **Total cost for execution material**

Items	Total
Cost for laboratory equipment	1666.00 €
Cost for manual work	44640.00 €

Total cost for execution material	**46306.00 €**

- **Contracting cost**

Items	Total
Total cost for execution material	46306.00 €
Industrial benefit (30%)	13891.80 €

Cost for contracting	**60197.80 €**

- **Writing remunerations**

Writing remunerations (7%)	**3124,80 €**

- **Grand total cost**

Items	Total
Cost for contracting	60197.80
Writing remunerations	3124,80

Grand total cost	**63322,60 €**

VII. References

Part I J. Broddfelt, “Tracking of multiple objects with Kalman filters - Part I” 2006.

[1] 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,pp:234-239, Edmonton, August 2005.

[2] D. Schulz, W. Burgard, D. Fox, and A. B Cremers. ”Tracking multiple moving targets with a mobile robot using particle filters and statistical data association”, In Proc. of the IEEE International Conference on Robotics & Automation (ICRA), 2001.

[3] Greg Welch, Gary Bishop, “An introduction to the Kalman Filter”, ACM, Inc. 2001.

[4] E.R. Davies, “Machine Vision – Theory Algorithms Practicalities”, 3rd edition, Elsevier Inc., ISBN: 0-12-206093-8, 2005, pp. 595-623.

[5] Sebastian Thrun, “Lecture 2, Lenses and Calibration”, CS223B Computer Vision, Stanford University, 2005 http://robots.stanford.edu/cs223b05/notes/CS%20223-B%20L2%20Lenses&Calibration.ppt)

[6] F. Gustafsson, L. Ljung, M. Millnert, ”Signalbehandling”, Studentlitteratur, Lund, ISBN: 91-44-01500-3, 2000.

[7] 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, pp. 2017-2022, Raleigh, November 2005.